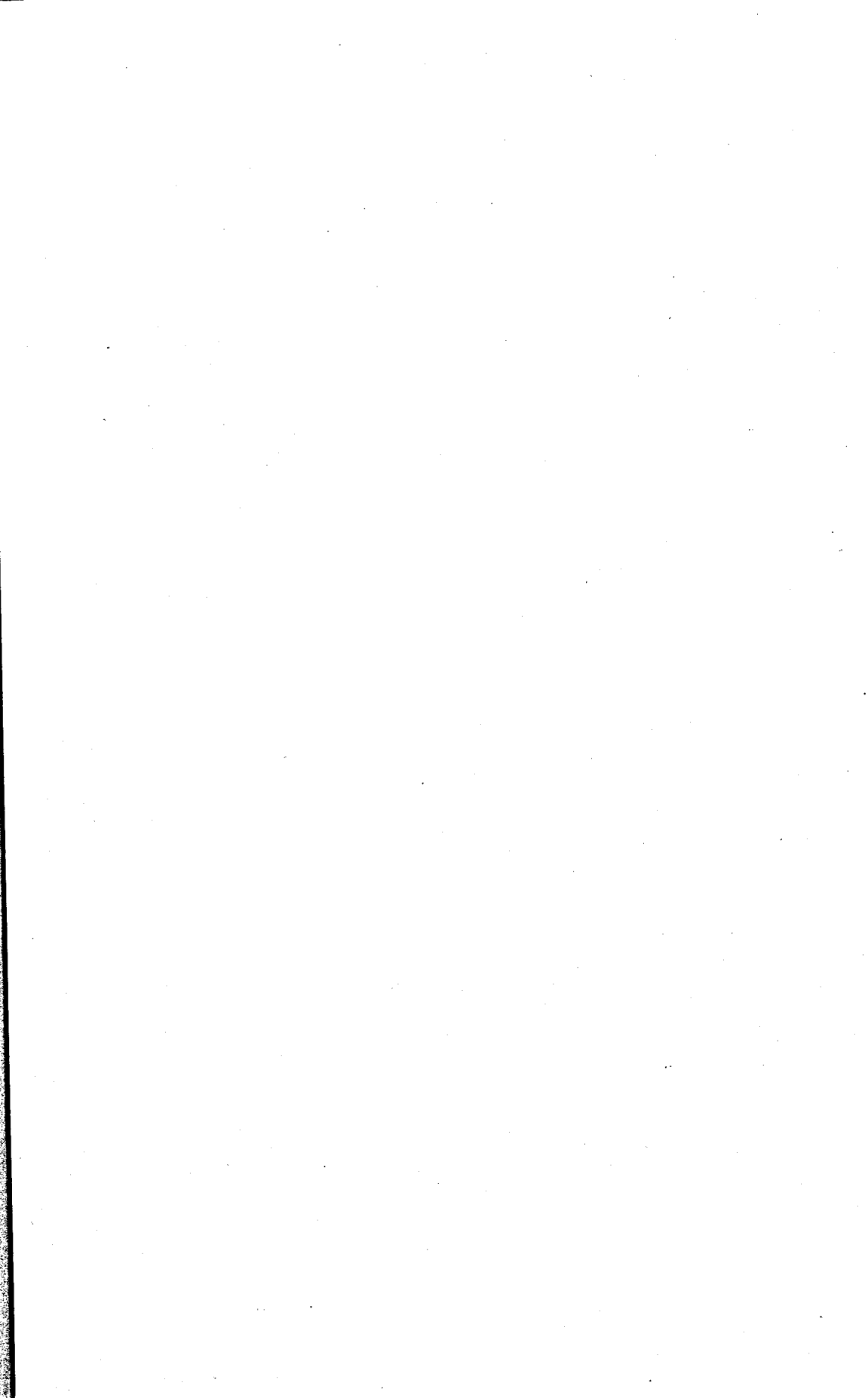


Meltzer
& Michie

and
MELTZER
MICHIE
MELTZER
MICHIE

Edinburgh



**MACHINE
INTELLIGENCE 4**



MACHINE INTELLIGENCE 4

edited by

BERNARD MELTZER

Metamathematics Unit
University of Edinburgh

and

DONALD MICHIE

Department of Machine Intelligence and Perception
University of Edinburgh

with a preface by

MICHAEL SWANN, FRS

Principal and Vice-Chancellor
University of Edinburgh

EDINBURGH at the University Press

© 1969 Edinburgh University Press
22 George Square, Edinburgh

Printed in Great Britain at
The Aberdeen University Press

85224 062 7

Library of Congress
Catalog Card Number
67-13648

PREFACE

In reviewing the first three *Machine Intelligence* volumes, *Nature* (14 December 1968) wrote: 'The enterprise of Edinburgh University in fostering many of the developments reported has been amply rewarded'. It also noted that several of the contributions were speculative, and seemed 'to grope in the darkness for some signs of a breakthrough towards a true Machine Intelligence'.

A Vice-Chancellor may be forgiven these days for seizing every chance to stress the obvious. The practice of research is proper and vital to a university. Such pursuits – speculative or experimental – often lead, penultimately, to a blind alley; and such negative results are themselves positive contributions to knowledge. John Donne put it well:

Doubt wisely On a huge hill,
Cragged, and steep, Truth stands, and he that will
Reach her, about must, and about must go

Furthermore, the practice of such research, as is described in this volume, has a markedly beneficial effect on the quality of teaching; not just on the teaching of those directly involved in it, but also of their colleagues who share so stimulating an intellectual environment.

In the eyes of the Greek bureaucrats, the teaching of Socrates corrupted the youth of Athens. In this sense, research is also, and literally, a subversive activity. Its results tend to overthrow established opinions and accepted truths, and totalitarian states therefore control its liberty. Thus the quality of university research – its intensity, its freedom, and its courage – is a good indication of a nation's self-confidence and psychological health. . . .

A particularly encouraging aspect of research in the field of Artificial Intelligence is its inter-disciplinary nature. The subject, one might hazard, is a new colloidal solution, with computer scientists, mathematicians, biologists, psychologists, metaphysicians, logicians, and linguistics scholars, beginning to form a new gel. No two cultures here. Indeed, the career of one distinguished contributor to this as to previous volumes suggests the emergence of a higher synthesis; for this particular scholar, after graduating in Classics at Cambridge, and in Philosophy at Princeton, and after holding, in succession, Chairs of Philosophy and Computer Science, is now the occupant of a new Chair of Logic *and* Computer Science.

PREFACE

The quality and success of this combined operation is readily evident in the present volume. Most notable is the long-predicted arrival of mathematical logic and its mechanisation at the centre of the machine intelligence arena. The fourth Machine Intelligence Workshop was noteworthy for a series of important theoretical contributions; in particular, several which apply the new insights to the design principles of intelligent computing systems. A further feature which distinguishes the present volume is a section devoted to the programming of robots so as to confer on them the capability to reason and plan.

We have here a nascent technology which provides a meeting ground for theorists, engineers, software innovators, and students of cognition in biological systems.

The speedy and accurate communication of research is a subordinate but still important part of scholarship. That this is where a University Press can often be most effective, is testified by the publication of this volume within five months of the receipt of the contributions.

MICHAEL SWANN

March 1969

CONTENTS

MATHEMATICAL FOUNDATIONS

- | | | |
|---|--|----|
| 1 | Program scheme equivalences and second-order logic. D. C. COOPER | 3 |
| 2 | Programs and their proofs: an algebraic approach.
R. M. BURSTALL and P. J. LANDIN | 17 |
| 3 | Towards the unique decomposition of graphs. C. R. SNOW and
H. I. SCOINS | 45 |

THEOREM PROVING

- | | | |
|---|---|-----|
| 4 | Advances and problems in mechanical proof procedures. D. PRAWITZ | 59 |
| 5 | Theorem-provers combining model elimination and resolution.
D. W. LOVELAND | 73 |
| 6 | Semantic trees in automatic theorem-proving. R. KOWALSKI and
P. J. HAYES | 87 |
| 7 | A machine-oriented logic incorporating the equality relation.
E. E. SIBERT | 103 |
| 8 | Paramodulation and theorem-proving in first-order theories with
equality. G. ROBINSON and L. WOS | 135 |
| 9 | Mechanizing higher-order logic. J. A. ROBINSON | 151 |

DEDUCTIVE INFORMATION RETRIEVAL

- | | | |
|----|--|-----|
| 10 | Theorem proving and information retrieval. J. L. DARLINGTON | 173 |
| 11 | Theorem-proving by resolution as a basis for question-answering
systems. C. CORDELL GREEN | 183 |

MACHINE LEARNING AND HEURISTIC PROGRAMMING

- | | | |
|----|--|-----|
| 12 | Heuristic dendral: a program for generating explanatory hypotheses
in organic chemistry. B. BUCHANAN, G. SUTHERLAND and
E. A. FEIGENBAUM | 209 |
| 13 | A chess-playing program. J. J. SCOTT | 255 |
| 14 | Analysis of the machine chess game. I. J. GOOD | 267 |
| 15 | PROSE - Parsing Recogniser Outputting Sentences in English.
D. B. VIGOR, D. URQUHART and A. WILKINSON | 271 |
| 16 | The organization of interaction in collectives of automata.
V. I. VARSHAVSKY | 285 |

CONTENTS

COGNITIVE PROCESSES: METHODS AND MODELS

- 17 Steps towards a model of word selection. G. R. KISS 315
18 The game of hare and hounds and the statistical study of literary
vocabulary. S. H. STOREY and M. A. MAYBREY 337
19 The holophone - recent developments. D. J. WILLSHAW and
H. C. LONGUET-HIGGINS 349

PATTERN RECOGNITION

- 20 Pictorial relationships - a syntactic approach. M. B. CLOWES 361
21 On the construction of an efficient feature space for optical character
recognition. A. W. M. COOMBS 385
22 Linear skeletons from square cupboards. C. J. HILDITCH 403

PROBLEM-ORIENTED LANGUAGES

- 23 Absys 1: an incremental compiler for assertions; an introduction.
J. M. FOSTER and E. W. ELCOCK 423

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

- 24 Planning and generalisation in an automaton/environment system.
J. E. DORAN 433
25 Freddy in toyland. R. J. POPPLESTONE 455
26 Some philosophical problems from the standpoint of artificial
intelligence. J. MCCARTHY and P. J. HAYES 463

- INDEX 505

**MATHEMATICAL
FOUNDATIONS**



Program Scheme Equivalences and Second-Order Logic

D. C. Cooper
Computation Department
University College of Swansea

The equivalence problem for program schemes, or for programs, is reduced to the proving of a theorem in second-order logic. This work extends Manna's first-order logic reductions. Some examples of the technique are given together with a suggested method for obtaining proofs in special cases by first-order methods.

INTRODUCTION

Several workers in recent years have considered using techniques and ideas of various mathematical theories of computation for proving interesting results about computer programs. This paper is concerned with two of these approaches.

Floyd stressed the practicability of, and formalised, an idea due to several people (Floyd 1967a, Floyd 1967b, Naur 1966). In this approach, predicates are associated with various parts of the program (including the exit point). These predicates may be thought of as expressing the relevant relations between the values of the variables of interest at the given point; in particular, the predicate on the exit point represents the property of the program we wish to prove. For example, in a sort program the final predicate could be: the values of A_1, \dots, A_n are some permutation of their original values and, for all i from 1 to $n-1$ we have $A_i \leq A_{i+1}$. By a mechanical procedure relations between these predicates may be found in the form of a formula of the first-order predicate calculus. If we can prove this is a theorem then we have proved that, assuming the program does not get stuck in a loop, the final values of the variables will indeed satisfy the exit predicate.

Floyd (1967a) also interprets his approach as a formal way of specifying the semantics of a programming language and discusses a technique for proving the non-looping of a program with given data. He discusses various features of ALGOL and their effect on the relations between the predicates; this paper will, however, only consider assignment statements and two-way tests.

Manna (1968) introduced the idea of associating predicate symbols, rather than specific predicates, with points in the program. He then showed how to construct a formula of first-order logic whose unsatisfiability proved the convergence of the program for all inputs, and a similar idea was used to prove convergence and equivalence of two programs. Knowledge of the content of Floyd's and Manna's papers is not required for an understanding of this paper. However, it is clear that the spirit of Manna's thesis permeates this paper, which is an extension of his work.

The second idea goes back to Ianov (*see* Rutledge 1964) and it is used in a form due to Luckham, Park and Paterson (1967). (*See also* Paterson 1967 and Paterson 1968.) In this approach it is recognized that for many of the properties to be proved the full detail of the precise functions used in the program is irrelevant. For example, one may only need to use the fact that a particular part of the program does not alter the contents of certain registers. One is then led to consider *program schemes* in which in place of actual functions and tests pure names are used. Two program schemes are then, for example, defined to be equivalent if and only if, whatever actual functions and tests are substituted for the names, the resulting programs are equivalent.

The program schemes of this paper are the same as those defined in Paterson's papers except for one inessential difference: in this paper the result of a program scheme will be regarded as the values in all (or some) of the registers rather than just 'yes' or 'no'.

PROGRAM SCHEMES

In a program scheme we have a set of registers L_1, L_2, \dots, L_n , a set of function names f_1, f_2, \dots, f_p , and a set of test names t_1, t_2, \dots, t_q , all these sets being finite. The function names may take several arguments, but without loss of generality the tests will be assumed to have only one argument. A program scheme consists of a series of statements each of which is either:

- (a) an assignment statement, e.g. $L_3 \leftarrow f(L_1, L_6)$
- or (b) a test statement, e.g. $t(L_6) \ 3,4$
- or (c) the halt statement
- or (d) a goto statement, e.g. goto 6.

Any statement may be labelled with an integer, e.g. (1) $L_3 \leftarrow f(L_1)$. In the test statement control goes to the statement with the first label if the predicate is false, otherwise to the statement with the second label.

Two examples follow, in which the predicates before the bar | may initially be ignored:

PS_1	PS_2
$L = \alpha$	$L = \alpha$
$A(L)$ (1) $t_1(L)$ 1,2	$B(L)$ (1) $t_1(L)$ 1,2
(2) $L \leftarrow f(L)$	(2) $L \leftarrow f(L)$
(3) $t_2(L)$ 1,4	(3) $t_2(L)$ 4,7

PS_1	PS_2
$Z(L) \mid (4) \text{ halt}$	$C(L) \mid (4) t_1(L) 4,5$
	$(5) L \leftarrow f(L)$
	$(6) t_2(L) 4,7$
	$Z(L) \mid (7) \text{ halt}$

These are both program schemes with one register (L), one function (f) and two tests (t_1 and t_2). The first is a simple loop within a loop, while the second is the same except that the outer loop has been unwound once (as can be seen from a flowchart). They are clearly equivalent, being a special case of the equivalence-preserving transformation figure 3(1v) of Paterson (1968) in which program scheme P_1 has itself been taken to be a loop. (It will be proved later that equivalence of PS_1 and PS_2 proves equivalence of Paterson's schemes for any P_1 ; this is why these examples with their unnatural 'loop stops' are used.) This example will be used to illustrate the technique.

By an *interpretation* of a program scheme we mean some universe of objects and some assignment of functions and predicates over this universe to the function and test names of the program scheme, and also a set of n objects from the universe to be taken as the initial values of the n registers. Thus, an interpretation of a program scheme gives a program in an obvious way together with a set of starting values for that program. The value of the program scheme under an interpretation is either undefined (if the program never terminates) or is the set of values of the registers when the program halts.

Two program schemes are *equivalent* if for every interpretation either both values are undefined or both are defined and equal. This equivalence problem is shown to be undecidable by Luckham, Park and Paterson (1967). Furthermore, they show that equivalence is not even partially decidable, that is, we cannot enumerate all pairs of equivalent program schemes (or, alternatively, even if we know two schemes are equivalent we cannot mechanically produce a proof). This is in contrast to the proving of theorems of the first-order predicate calculus.

A *control path* of a program scheme is a sequence of instructions of the program such that if one member of the sequence is an assignment statement then the next member is the next instruction of the program sequence; if one member is a test statement then the next member is one of the instructions with a label occurring in the test statement; if one member is a *goto* statement then the next member is that instruction with the label named in the *goto* statement; and *halt* can only be the last member of the sequence.

These notions are treated more formally by Paterson (1967). Another definition we need is PS_1 extends PS_2 , where PS_1 and PS_2 are program schemes. By this is meant that under any interpretation for which the value of PS_2 is defined, the value of PS_1 is also defined and is equal to the value of PS_2 . Clearly PS_1 extends PS_2 and PS_2 extends PS_1 if and only if PS_1 and PS_2 are equivalent.

DEFINITION OF PREDICATES

Predicate names are associated with some (or all) of the instructions of a program scheme in such a way that every loop includes at least one such name; these are to be n -ary predicates for an n -register scheme. Intuitively, they represent conditions which will be satisfied just before that instruction is obeyed. With **halt** statements associate the special name Z . Such names have been indicated in the column before the bar of the above examples.

Now consider any control path of a program scheme which starts and ends at statements with associated predicate names, and which has no intermediate statement with an associated predicate. Consider also control paths which start at an additional 'initialising' statement before the first instruction with which is associated the predicate $L_1=\alpha_1 \ \& \ L_2=\alpha_2 \ \& \ \dots \ \& \ L_n=\alpha_n$. For PS_1 all possible such sequences are 0-1, 1-1, 1-2-3-1, 1-2-3-4 and for PS_2 they are 0-1, 1-1, 1-2-3-4, 1-2-3-7, 4-4, 4-5-6-4, 4-5-6-7, where 0 refers to the extra initialising statement while other statements are referred to by their labels.

For each of these control paths let P_i be the predicate on the first statement and P_j the predicate on the last statement. Let $\phi_{ij}(\{L\})$ be the condition for this control path to be traversed and let $\psi_{ij}(\{L\})$ be the final set of values of the registers, $\{L\}$ being the set of values at the start of this control path. Both ϕ_{ij} and ψ_{ij} will be defined in terms of the function names and test names occurring on the control path in an obvious way. Define C_{ij} by:

$$C_{ij} \text{ is } (\forall L_1)(\forall L_2) \dots (\forall L_n)[P_i(\{L\}) \ \& \ \phi_{ij}(\{L\}) \rightarrow P_j(\psi_{ij}(\{L\}))],$$

where we assume $\&$ is more binding than \rightarrow .

For example, for the path 4-5-6-7 of PS_2 we have

$$C_{ij} \text{ is } (\forall L)[C(L) \ \& \ t_1(L) \ \& \ t_2(f(L)) \rightarrow Z(f(L))],$$

and for the path 0-1 of PS_2 we have

$$C_{ij} \text{ is } (\forall L)[L=\alpha \rightarrow B(L)] \quad (\text{or the logically equivalent } B(\alpha)).$$

Let C be the conjunction of all possible C_{ij} (there can only be a finite number of C_{ij} because of the restriction that every loop includes a statement with an associated predicate).

Finally define

$$PS \text{ does } Z \text{ by } (\exists P_1)(\exists P_2) \dots (\exists P_k) C,$$

where P_1, P_2, \dots, P_k are all the predicate names associated with statements, excepting the predicate on the **halt** statement, Z .

The predicate PS does Z depends on Z , on the test and function names occurring in the program scheme and on the initial values; that is, given an interpretation and a predicate Z , PS does Z is completely defined and will be either true or false. This is the relation to be investigated.

For the previously defined program schemes we have:

PS_1 does Z is

$$(\exists A)(L)[A(\alpha) \& \\ A(L) \& \neg t_1(L) \rightarrow A(L) \& \\ A(L) \& t_1(L) \& \neg t_2(f(L)) \rightarrow A(f(L)) \& \\ A(L) \& t_1(L) \& t_2(f(L)) \rightarrow Z(f(L))] \dots\dots 1.$$

PS_2 does Z is

$$(\exists B)(\exists C)(L)[B(\alpha) \& \\ B(L) \& \neg t_1(L) \rightarrow B(L) \& \\ B(L) \& t_1(L) \& \neg t_2(f(L)) \rightarrow C(f(L)) \& \\ B(L) \& t_1(L) \& t_2(f(L)) \rightarrow Z(f(L)) \& \\ C(L) \& \neg t_1(L) \rightarrow C(L) \& \\ C(L) \& t_1(L) \& \neg t_2(f(L)) \rightarrow C(f(L)) \& \\ C(L) \& t_1(L) \& t_2(f(L)) \rightarrow Z(f(L))] \dots\dots 2.$$

THE PS 'DOES' Z RELATION

PS does Z is a well-formed formula of second-order logic (as it involves quantification over predicates). The existentially quantified predicates correspond to the predicates which Floyd (1967a, 1967b) and Manna (1968) associate with points in a program. Once these have been guessed, PS does Z becomes a formula of first-order logic; and if for some particular Z it can be proved true then it has been shown that if the program terminates Z is true. This is the principle of the verifying compiler (Floyd 1967b) and is expressed by Theorem 1 below. In this paper only a trivial kind of programming language is being considered, whereas Floyd also deals with more sophisticated constructs. Manna (1968) introduced the notion of taking Z to be the constant predicate **false**, in order to prove results about non-termination; these results may be deduced from Theorem 2 below. One may then consider what the relation between two program schemes must be if PS does Z is the same for both. Theorem 4 states that in fact such program schemes are equivalent.

Given a particular interpretation of a program scheme we have:

Theorem 1 If the scheme terminates with value $L_1^F, L_2^F, \dots, L_N^F$ then PS does $Z \equiv Z(L_1^F, L_2^F, \dots, L_N^F)$ is true for any Z .

and

Theorem 2 The scheme loops if and only if PS does **false** is true. (An alternative form of Theorem 2 is: The scheme loops if and only if P does Z is true for all predicates Z .)

Relations between two program schemes are expressed by:

Theorem 3 PS_1 extends PS_2 if and only if PS_1 does $Z \rightarrow PS_2$ does Z is a theorem of second-order logic.

with the obvious corollary:

Theorem 4 $PS_1 \equiv PS_2$ if and only if PS_1 does $Z \equiv PS_2$ does Z is a theorem.

These theorems relate the looping and equivalence of program schemes (or programs) to second-order logic, while Manna (1968) relates these issues to first-order logic. The difference is that Manna assumes one already has available the condition for the program not to loop. If this condition is expressible in first-order logic and is known, then proof of non-looping, of equivalence or of satisfaction of given conditions on termination are all reduced to first-order logic problems by Manna. With actual programs the properties of the functions and tests occurring may well only be expressible in second-order logic, or only more clumsily or incompletely in first-order logic. However, if one can express these relations in first-order logic and if, as is often the case, the condition for non-looping is known and easily expressible, then Manna's approach is very attractive.

The fact that equivalence of program schemes is not partially decidable shows that this problem cannot be reduced to the proving of a theorem of first-order logic, as this is partially decidable.

PROOF OF THEOREMS

Theorem 1. Given an interpretation of a program scheme, PS , which causes the scheme to terminate with value $\{L^F\}$, then, with this interpretation,

$$PS \text{ does } Z \equiv Z(\{L^F\}).$$

($\{L\}$ denotes the set of registers of the program scheme, $\{L^F\}$ their final values.)

Proof (a) Assume $PS \text{ does } Z$ is true.

Take the predicates, which exist by definition of $PS \text{ does } Z$, and consider the control path of the scheme under the given interpretation. By the definition of the C_{ij} (see definition of $PS \text{ does } Z$) as we follow the control path all predicates will take the value true and in particular Z will be true at the end.

(b) Assume $Z(\{L^F\})$ is true.

Suppose with the given interpretation control reaches a node p times with $\{L\} = \{L_1\}, \{L_2\}, \dots, \{L_p\}$. Associate with this node the predicate

$$\{L\} = \{L_1\} \vee \{L\} = \{L_2\} \vee \dots \vee \{L\} = \{L_p\}.$$

We show with this choice for P_1, P_2, \dots, P_k , $PS \text{ does } Z$ is true.

Consider C_{ij} , defined as

$$(\forall L_1)(\forall L_2) \dots (\forall L_n)[P_i(\{L\}) \ \& \ \phi_{ij}(\{L\}) \rightarrow P_j(\psi_{ij}(\{L\}))].$$

If $\{L\}$ is a set of values of the registers on the control path at the i th node, and if with this set control does then go to the j th node, then all three terms of the quantifier-free part of C_{ij} are true, and therefore so is the implication. If $\{L\}$ is such a set and control does not then go to the j th node, then the ϕ_{ij} term is false; if $\{L\}$ is not such a set then the P_i term is false. In either case the implication is true and so we have proved C_{ij} to be true; the argument is also valid if i is the start node or j the terminating node.

Hence C , the conjunction of the C_{ij} is true, and hence PS does Z is true.

Theorem 2. Given an interpretation of a program scheme, PS , the scheme loops if and only if in that interpretation PS does false is true.

Proof (a) Assume the scheme loops.

The proof is exactly the same as for Theorem 1, part (b). The predicates are defined as before, but now for some node or nodes the disjunction will be infinite, these predicates being still well defined but in a non-constructive manner. As control never reaches the final node the truth value of Z will never be needed in proving PS does Z true.

(b) Assume the scheme terminates with $Z(\{L\}) = \{L^F\}$.

Use Theorem 1 with Z replaced by false.

Minor variants of these proofs also prove the alternative formulation of Theorem 2, that the scheme loops if and only if PS does Z is true for all predicates Z .

Theorem 3. PS_1 extends PS_2 if and only if PS_1 does $Z \rightarrow PS_2$ does Z is a theorem.

Proof (a) Assume PS_1 extends PS_2 .

Take any interpretation and any predicate Z . Suppose under this interpretation PS_2 terminates with $\{L\} = \{L^F\}$, then so does PS_1 by the definition of extension. Then PS_1 does $Z \equiv PS_2$ does Z , as by Theorem 1 both are equivalent to $Z(\{L^F\})$. This is a stronger result than required. Suppose under this interpretation PS_2 does not terminate, then by the alternative Theorem 2 PS_2 does Z is true, and trivially PS_1 does $Z \rightarrow PS_2$ does Z .

(b) Assume PS_1 does $Z \rightarrow PS_2$ does Z is a theorem.

Take any interpretation under which PS_2 converges, to $\{L_{F2}\}$, say. By Theorem 2 PS_2 does false is false; the hypothesis then shows that PS_1 does false is false and by Theorem 2 again PS_1 converges, to $\{L_{F1}\}$, say.

Now take $Z(\{L\})$ to be $\{L\} = \{L_{F1}\}$. With this Z , PS_1 does Z is true, and by hypothesis PS_2 does Z is true. Theorem 1 then gives $\{L_{F1}\} = \{L_{F2}\}$, hence proving PS_1 extends PS_2 .

Theorem 4. PS_1 is equivalent to PS_2 if and only if PS_1 does $Z \equiv PS_2$ does Z is a theorem. The proof of this is immediate by Theorem 3.

PROVING EQUIVALENCE OF PROGRAM SCHEMES

To prove two program schemes, PS_1 and PS_2 , equivalent one can proceed by proving both PS_1 extends PS_2 and its converse. By Theorem 3 this is equivalent to proving PS_1 does $Z \rightarrow PS_2$ does Z . This is a formula of second-order logic. Theoremhood in second-order logic is not even partially decidable; however, in particular cases this theoremhood is provable within first-order logic.

The formula PS_1 does $Z \rightarrow PS_2$ does Z is of the form:

$$(\exists P_1)(\exists P_2) \dots C \rightarrow (\exists Q_1)(\exists Q_2) \dots D,$$

or equivalently

$$(\forall P_1)(\forall P_2) \dots (\exists Q_1)(\exists Q_2) \dots \{C \rightarrow D\}.$$

If we can guess the predicates Q_1, Q_2, \dots , (in terms of the predicates P_1, P_2, \dots , the test and function names of the program schemes, Z and the initial values), then with this guess we have eliminated the existential quantifiers on the predicates, the initial universal quantifiers may be dropped, and we have a formula of first-order logic.

In the previous example, equations 1 and 2, we first have to guess B and C , then drop all existential quantifiers and prove formula 2 follows from formula 1. Then the process is repeated guessing A and proving 1 follows from 2. In this case the guessing is trivial; for the first proof take both B and C to be A . Each of the conjuncts of 2 is then a conjunct of 1 and so 2 follows from 1. For the second proof take A to be the disjunct $B \vee C$, each of the conjuncts of 1 may then be proved true by using the conjuncts of 2. It has therefore been proved that the program schemes PS_1 and PS_2 are equivalent; more complex examples will follow later.

Paterson (1967) considers the problem of finding 'adequate rule books' for proving equivalences or for simplification of schemes by successive transformations. Under a very weak hypothesis these rule books cannot exist. The fact that equivalences are not even partially decidable is the stumbling block. However, if we consider restricted definitions of equivalence and if this restricted equivalence is partially decidable, then perhaps one can find rule books or produce mechanical equivalence-provers. Suppose the predicates to be guessed in the above technique are chosen from some recursively enumerable set, for example all those which may be defined by well-formed formulae of first-order logic using the given predicates, test and function names. Then, with this choice, a theorem of first-order logic has to be proved and this itself is partially solvable. The set of pairs of programs which can be proved equivalent by this method is then recursively enumerable and mechanical provers or adequate rule books can be looked for. Such equivalent program schemes will be equivalent in a strong sense; intuitively, the conditions for being at a particular place in one scheme can be stated in terms of the conditions of the other scheme.

GIVEN RELATIONS

Suppose in the program schemes there are given relations between the tests and functions, and equivalence has to be proved only on the assumption that the relations are satisfied. The previous theorems may be easily extended to cover this case. Let Q denote all the given conditions; define PS_1 extends PS_2 over Q by restricting the interpretations to those for which Q is true. Theorem 3, for example, may then be generalised to:

Theorem 3A. PS_1 extends PS_2 over Q if and only if
 $Q \ \& \ PS_1$ does $Z \rightarrow PS_2$ does Z

In the limit, if Q contains sufficient restrictions completely to define the functions and tests then equivalence of programs themselves is being considered.

SUB-PROGRAM SCHEMES

Suppose, as in figure 3 (iv) of Paterson (1968), we have two program schemes each with an unspecified sub-program scheme, and it is desired to prove equivalence whatever particular sub-program scheme is substituted into the two schemes. The following theorem shows that we need then only consider one particular substitution. Suppose the program schemes have one register:

Theorem 5. Let $PS_1(P)$ and $PS_2(P)$ be two program schemes each with an unspecified sub-program scheme P . Then $PS_1(P) \equiv PS_2(P)$ for all P if and only if $PS_1(Q) \equiv PS_2(Q)$, where Q is the particular program scheme

- (1) $t(L)$ 1,2
- (2) $L \leftarrow f(L)$

and it is assumed that labels (1) and (2), test name t and function name f do not occur in the given schemes.

Proof. Assume $PS_1(Q) \equiv PS_2(Q)$, where Q is the given particular program scheme, and take any program scheme P .

Consider any interpretation and starting value for $PS_1(P)$ and $PS_2(P)$. If the t and/or f of program scheme Q occur in P , rename them to get a new Q and still have $PS_1(Q) \equiv PS_2(Q)$. Now consider the interpretation of $PS_1(Q)$ and $PS_2(Q)$, which is the chosen interpretation, extended by letting t be the condition for P to loop and f the function relating P 's final value to its initial value if it does not loop (this test and function will, of course, depend on the interpretation already given to the tests and functions of P). It is clear that $PS_1(Q)$ and $PS_1(P)$ must behave the same way in this interpretation, and $PS_2(Q)$ and $PS_2(P)$ similarly. But $PS_1(Q) \equiv PS_2(Q)$, hence $PS_1(P)$ and $PS_2(P)$ must also both loop or both have the same value. Since we chose any interpretation it follows that $PS_1(P) \equiv PS_2(P)$.

The converse is trivial.

The extension to many-register schemes or to the occurrence of several sub-program schemes in one or both given schemes is easy.

Thus our previous equivalence proof and this theorem prove that the transformation of Paterson is valid for any program scheme P_1 .

A RESTRICTED SET OF ANSWERS

It may be that only the final value in some of the registers is relevant, so that one must prove of two program schemes that the final values of registers in a specified set are the same. The previous technique can be easily extended to this case, the final predicate Z being taken to depend only on the chosen registers. As an example, consider:

PS_3 $L = \alpha \ \& \ M = \beta \mid$ $M \leftarrow f(M)$ $L \leftarrow g(L)$ $Z(M) \mid \text{halt}$	PS_4 $L = \alpha \ \& \ M = \beta \mid$ $P(L, M) \mid (3) \ t(L) \ 1, 2$ $(1) \ L \leftarrow f(L)$ $\text{goto } 3$ $(2) \ M \leftarrow f(M)$ $Z(M) \mid \text{halt}$
---	---

Clearly PS_3 extends PS_3 so long as M alone is regarded as the answer, the final L values being different. This is also an example of an extension which is not an equivalence.

PS_3 does Z is

$$(\forall L)(\forall M)[L = \alpha \ \& \ M = \beta \rightarrow Z(f(M))].$$

PS_4 does Z is

$$(\exists P)(\forall L)(\forall M)[P(\alpha, \beta) \ \& \\ P(L, M) \ \& \ \neg t(L) \rightarrow P(f(L), M) \ \& \\ P(L, M) \ \& \ t(L) \rightarrow Z(f(M))].$$

If $P(L, M)$ is 'guessed' to be $M = \beta$ then PS_4 does Z may easily be deduced from PS_3 does Z .

FURTHER EXAMPLES

As a more complex example consider the following three program schemes. They are all two-register and consist of two independent loops, one working on one register (L) and the other on the other (M). In PS_5 the L loop is first; in PS_6 the M loop is first; and in PS_7 operations in the two registers are intermingled (as may be seen by drawing flowcharts). They are all equivalent; the predicates to be guessed will be exhibited later. The letters t and s are test names, f and g function names.

PS_5 $L = \alpha \ \& \ M = \beta \mid$ $A(L, M) \mid (1) \ t(L) \ 2, 4$ $(2) \ L \leftarrow f(L)$ $(3) \ \text{goto } 1$ $B(L, M) \mid (4) \ s(M) \ 5, 7$ $(5) \ M \leftarrow g(M)$ $(6) \ \text{goto } 4$ $Z(L, M) \mid (7) \ \text{halt}$	PS_6 $L = \alpha \ \& \ M = \beta \mid$ $V(L, M) \mid (1) \ s(M) \ 2, 4$ $(2) \ M \leftarrow g(M)$ $(3) \ \text{goto } 1$ $U(L, M) \mid (4) \ t(L) \ 5, 7$ $(5) \ L \leftarrow f(L)$ $(6) \ \text{goto } 4$ $Z(L, M) \mid (7) \ \text{halt}$
PS_7 $L = \alpha \ \& \ M = \beta \mid$ $C(L, M) \mid (1) \ t(L) \ 2, 6$ $(2) \ s(M) \ 3, 9$ $(3) \ L \leftarrow f(L)$ $(4) \ M \leftarrow g(M)$ $(5) \ \text{goto } 1$	

$E(L,M)$	(6) $s(M)$ 7,12
	(7) $M \leftarrow g(M)$
	(8) goto 6
$D(L,M)$	(9) $t(L)$ 10,12
	(10) $L \leftarrow f(L)$
	(11) goto 9
$Z(L,M)$	(12) halt

The following are easily produced:

PS_5 does Z is

$$(\exists A)(\exists B)(\forall L)(\forall M)[A(\alpha, \beta) \& \\ A(L, M) \& \neg t(L) \rightarrow A(f(L), M) \& \\ A(L, M) \& t(L) \rightarrow B(L, M) \& \\ B(L, M) \& \neg s(M) \rightarrow B(L, g(M)) \& \\ B(L, M) \& s(M) \rightarrow Z(L, M)].$$

PS_6 does Z is

$$(\exists V)(\exists U)(\forall L)(\forall M)[V(\alpha, \beta) \& \\ V(L, M) \& \neg s(M) \rightarrow V(L, g(M)) \& \\ V(L, M) \& s(M) \rightarrow U(L, M) \& \\ U(L, M) \& \neg t(L) \rightarrow U(f(L), M) \& \\ U(L, M) \& t(L) \rightarrow Z(L, M)].$$

PS_7 does Z is

$$(\exists C)(\exists D)(\exists E)(\forall L)(\forall M)[C(\alpha, \beta) \& \\ C(L, M) \& \neg t(L) \& \neg s(M) \rightarrow C(f(L), g(M)) \& \\ C(L, M) \& \neg t(L) \& s(M) \rightarrow D(L, M) \quad \& \\ C(L, M) \& t(L) \quad \quad \quad \rightarrow E(L, M) \quad \quad \& \\ D(L, M) \& \neg t(L) \quad \quad \quad \rightarrow D(f(L), M) \quad \& \\ D(L, M) \& t(L) \quad \quad \quad \rightarrow Z(L, M) \quad \quad \& \\ E(L, M) \& \neg s(M) \quad \quad \quad \rightarrow E(L, g(M)) \quad \& \\ E(L, M) \& s(M) \quad \quad \quad \rightarrow Z(L, M)].$$

To prove equivalence we need to prove:

1. PS_5 does $Z \rightarrow PS_6$ does Z .

'Guess' V and U to be

$$V(L, M) \text{ is } L = \alpha \& (\forall L')[B(L', \beta) \rightarrow B(L', M)]$$

$$U(L, M) \text{ is } A(L, \beta) \& (\forall L')[B(L', \beta) \rightarrow Z(L', M)].$$

With the aid of this substitution the fact that PS_5 does $Z \rightarrow PS_6$ does Z is a theorem may be proved within first-order logic. This is left to the reader to verify.

2. PS_6 does $Z \rightarrow PS_5$ does Z .

This follows from 1. by symmetry.

3. PS_5 does $Z \rightarrow PS_7$ does Z .

With V and U defined as in 1. guess C , D , and E to be:

$$C(L, M): A(L, \beta) \ \& \ V(\alpha, M)$$

$$D(L, M): U(L, M)$$

$$E(L, M): B(L, M)$$

The theorem follows.

4. PS_7 does $Z \rightarrow PS_5$ does Z .

This is a difficult proof, as there seems no way to express the predicates of PS_5 in terms of those of PS_7 by definitions of the type considered so far, that is, by a formula of first-order logic. However, assuming the existence of predicates expressing the condition that control is in the C , D , and E loops of PS_7 , and assuming some obvious properties of these predicates, 4 may be proved true.

Use a superscript to denote functional composition, e.g. $f^2(\alpha)$ stands for $f(f(\alpha))$ and in general $f^n(\alpha)$ for n compositions ($f^0(\alpha)$ is defined to be α).

Let p and q be the smallest integers such that $t(f^p(\alpha))$ and $s(g^q(\beta))$ are true (or ∞ if there is not such an integer).

Define: $\phi(L, M)$ is $(\exists n)[n \leq p \ \& \ n \leq q \ \& \ L = f^n(\alpha) \ \& \ M = g^n(\beta)]$

$\psi(L, M)$ is $q < p \ \& \ M = g^q(\beta) \ \& \ (\exists n)[L = f^n(\alpha) \ \& \ q \leq n \leq p]$

$\Gamma(L, M)$ is $p \leq q \ \& \ L = f^p(\alpha) \ \& \ (\exists n)[M = g^n(\beta) \ \& \ p \leq n \leq q]$

$\chi(L, M)$ is $(\exists n)[L = f^n(\alpha) \ \& \ n \leq p] \ \& \ (\exists n)[M = g^n(\beta) \ \& \ n \leq q]$

so that ϕ , ψ and Γ are the conditions on L and M when control of PS_7 is in the C , D , and E loops respectively, and χ is the condition that the L and M are both possible values in PS_7 .

Now guess:

$$A \text{ is } \chi(L, M) \ \& \ M = \beta \ \& \ (\forall M')[\phi(L, M') \rightarrow C(L, M') \ \& \ \psi(L, M') \rightarrow D(L, M')].$$

$$B \text{ is } \chi(L, M) \ \& \ (\forall M')[\phi(L, M') \rightarrow E(L, M') \ \& \ \psi(L, M') \rightarrow Z(L, M')] \ \& \ (\forall L')[\phi(L', M) \rightarrow C(L', M) \ \& \ \Gamma(L', M) \rightarrow E(L', M)] \ \& \ t(L).$$

These guesses enable the proof of 4 to be carried out. This proof will require several properties of ϕ , ψ , Γ and χ to be used, for example

$$\phi(\alpha, M) \rightarrow M = \beta \ \& \ g(M) \neq \beta \ \& \ \phi(L, g(M)) \rightarrow (\exists L')[L = f(L') \ \& \ \neg t(L') \ \& \ \phi(L', M)].$$

A complete list will not be given here (with the author's approach sixteen such properties were required), but it is clear that the above-defined ϕ , ψ , Γ , and χ have all the required properties. Another way of putting this is that $\{(\exists\phi)(\exists\psi)(\exists\Gamma)(\exists\chi) \text{ all these properties}\}$ is a theorem of second-order logic. On this assumption, 4 can then be proved within first-order logic.

It is discouraging that a simple example seems to need such complex

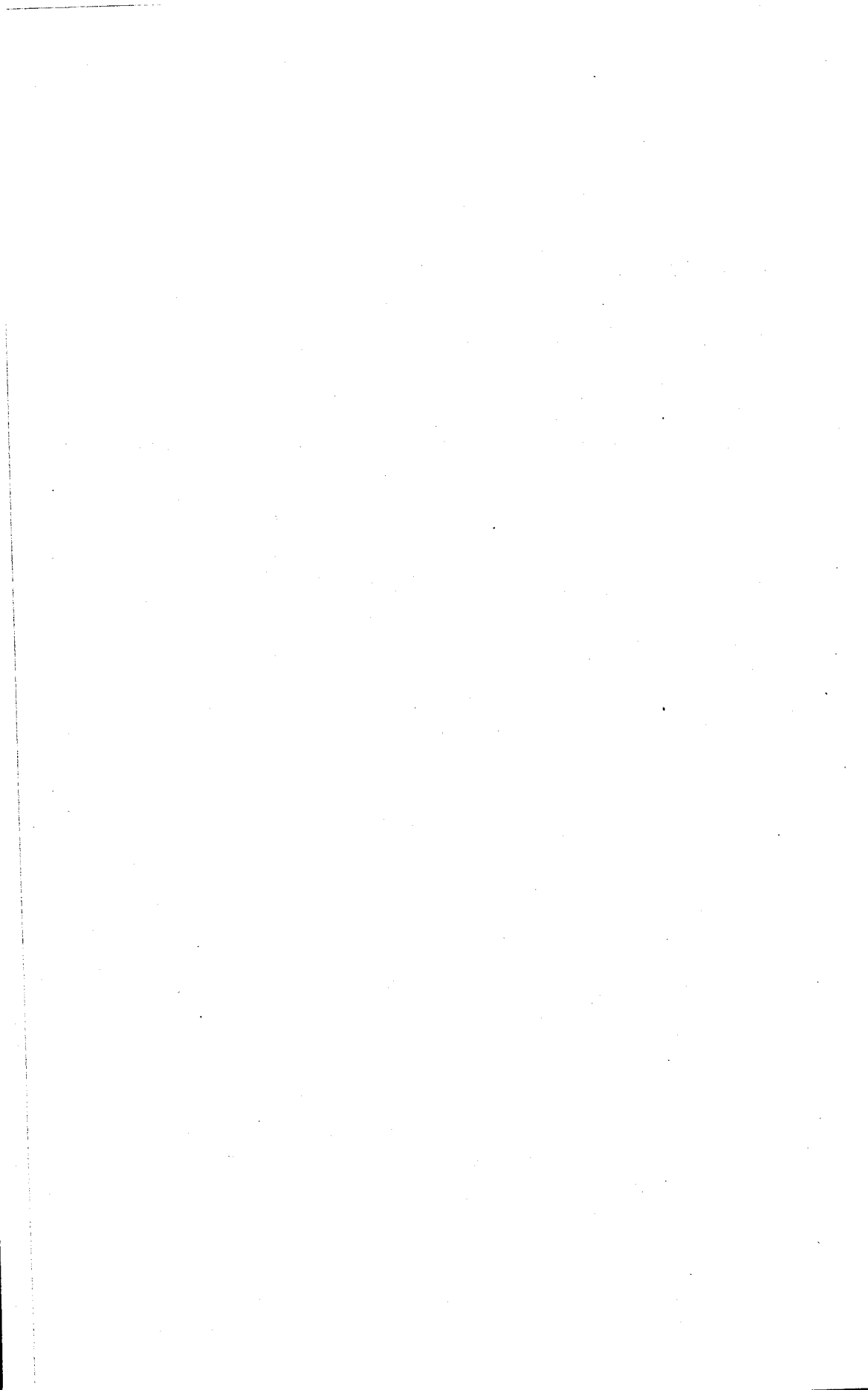
treatment. If our proof techniques do not work easily on simple examples, how shall we ever deal with real programs? The ϕ , etc., introduced, implied some knowledge about the program scheme; they are very like the predicates which have to be supplied by the user in Floyd's verifying compiler. However, 1, 2, and 3 did not need such guesses, and proof of these extensions fell into the partially solvable class in which the predicates were expressible by formulae of first-order logic.

Acknowledgements

The ideas used in this paper are clearly slight extensions of those put forward by Floyd in his verifying compiler and by Manna in his use of predicate symbols to reduce termination and equivalence problems to the first-order predicate calculus. I also wish to acknowledge helpful and productive conversations which I have had with D. Park and M. S. Paterson.

REFERENCES

- Floyd, R.W. (1967a) Assigning meanings to programs. *Proc. Amer. Math. Soc. Symposia in Applied Mathematics*, **19**, 19-31.
- Floyd, R.W. (1967b) The verifying compiler. *Computer Science Research Review*, pp. 18-19. Carnegie-Mellon University Annual Report, 1967.
- Luckham, D.C., Park, D.M.R. & Paterson, M.S. (1967) On formalised computer programs. Programming Research Group, Oxford University.
- Manna, Z. (1968) Termination of Algorithms. Ph.D. thesis. Carnegie-Mellon University.
- Naur, P. (1966) Proof of algorithms by general snapshots. *B.I.T.*, **6**, 310-16.
- Paterson, M.S. (1967) Equivalence problems in a model of computation. Ph.D. thesis. Trinity College, Cambridge.
- Paterson, M.S. (1968) Program schemata. *Machine Intelligence 3*, pp. 19-31 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
- Rutledge, J.D. (1964) On Ianov's program schemata. *J. Ass. Comput. Mach.*, **11**, 1-9.



Programs and their Proofs: an Algebraic Approach

R. M. Burstall

Department of Machine Intelligence and Perception
University of Edinburgh

P. J. Landin

Queen Mary College
University of London

1. INTRODUCTION

In this paper we try to show how an algebraic point of view can be helpful in the discussion of computer programs and their correctness proofs. There are a number of instances where algebraic concepts fit naturally on to programming phenomena and they bring a certain conciseness and uniformity to the treatment of programming matters. We have made an effort to understand some of these concepts and discover their connection with programming. Here we present the results of a preliminary skirmish in this field, namely an algebraic treatment of the proof of correctness of a simple compiler for expressions.

Programming is essentially about certain 'data structures' and functions between them. Algebra is essentially about certain 'algebraic structures' and functions between them. Starting with such familiar algebraic structures as groups and rings algebraists have developed a wider notion of algebraic structure (or 'algebra') which includes these as examples and also includes many of the entities which in the computer context are thought of as data structures.

Almost any branch of mathematics faces the problem of 'multum in parvo', how to describe an infinite set of arbitrarily long calculations or arbitrarily large structures using only a finite description. Programming manages this by the device of iteration (loops in programs, stars in regular expressions) or alternatively by recursion (recursive definitions of functions, recursive syntax definitions).

MATHEMATICAL FOUNDATIONS

Algebra seems to rely on different methods of description, largely the notion of homomorphism between algebras and the notion of the closure of a set under certain operations. The method of description is closely related to the methods of proving theorems about the objects described. The algebraic frame of discourse lends itself to certain kinds of inductive proof and sometimes enables one to 'wrap up' the conclusions of certain fundamental inductive arguments in convenient and very general lemmas.

Our business then is to define some useful algebraic concepts, indicate their points of contact with programming and illustrate their application in a simple correctness proof. We aim to be rigorous but not formal. Naturally, correctness proofs are more attractive if they can be formalised so as to make them susceptible to computer checking and even computer discovery. The reasoning here can appropriately be formalised in a higher-order logic rather than a first-order one, and one may hope that such logics will soon be mechanised.

2. NOTATION

When we speak of a function (or mapping) f we associate with it a domain X and a co-domain Y , writing $f: X \rightarrow Y$. This means that f is defined for each x in X and takes a value y in Y . We write $X \rightarrow Y$ for the set of all functions from X to Y . Thus $f: X \rightarrow Y$ is short for $f \in (X \rightarrow Y)$.

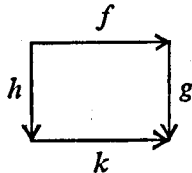
We write $f x$ for the result of applying f to x , instead of the commoner but cumbersome $f(x)$. Function applications associate to the *left* thus $f x y = (f x)y \neq f(x y)$, that is, $f x$ applied to y . For functions of two or more arguments we write $f(x, y)$, etc.

The image of f , written $im f$, is the set of all y in Y such that $y = f x$ for some x in X . Plainly, $im f \subseteq Y$; if $im f = Y$ we say that f is *onto* Y . If for each y there is at most one x in X such that $y = f x$, we say that f is *one-one* (it may or may not be onto Y).

If $f: X \rightarrow Y$ and $g: X' \rightarrow Y$ and $X' \subseteq X$ and for each x in X' $f x = g x$ then we say that g is a *restriction* of f and f is an *extension* of g .

The composition of two functions $f: X \rightarrow Y$ and $g: Y \rightarrow Z$ is denoted by $f \circ g$, that is, $f \circ g: X \rightarrow Z$ and $(f \circ g)x = g(f x)$. Note the inversion of order.

We say that a diagram such as the one below *commutes* if $f \circ g = h \circ k$



We denote n -tuples or lists by (x_1, \dots, x_n)

The following symbols denote special sets.

- N^+ : the set of positive integers
- N : the set of non-negative integers
- Z : the set of integers
- R : the set of reals.

3. BASIC ALGEBRAIC CONCEPTS

Those familiar with algebraic ideas may wish to skip this section, referring back later to clarify points of terminology. Others may find the definitions lacking in motivation until they have read the applications which follow. Our treatment is largely derived from that of Cohn (1965), which may be consulted for proofs omitted here.

Algebras

By an *operator set with arity* shall we mean a set Ω of objects called operators with a function from operators to non-negative integers, *arity*: $\Omega \rightarrow N$, indicating how many arguments each operator needs, that is, whether it is nullary, unary, binary, etc. Let Ω_n be the subset of Ω containing operators of arity n .

By an Ω -*algebra* A_Ω we mean an operator set Ω with arity and a set A , called the *carrier* of the algebra, together with a function for each n in N assigning to each operation ω in Ω_n a function from n -tuples of elements of A to A , that is, a function in $A^n \rightarrow A$. Call this function opn_n : $\Omega_n \rightarrow (A^n \rightarrow A)$.

Example. Any group is an algebra with operator set $\Omega = \{\omega', \omega'', \omega'''\}$ and arity, *arity* $\omega' = 2$, *arity* $\omega'' = 1$, *arity* $\omega''' = 0$. We take ω' to mean multiplication, ω'' to mean inversion and ω''' to mean the constant identity. The set Ω and the function *arity* are common to all groups. A particular group, say the cyclic group of order 4 with elements $\{I, e_1, e_2, e_3\}$, is specified by defining particular functions opn_2 , opn_1 , opn_0 as follows:

$$opn_2 \omega' = m, \text{ where } m(I, I) = I, m(I, e_1) = e_1, \dots, m(e_1, e_1) = e_2, \\ m(e_1, e_2) = e_3, \text{ etc.}$$

$$opn_1 \omega'' = inv, \text{ where } inv I = I, inv e_1 = e_3, \text{ etc.}$$

$$opn_0 \omega''' = id, \text{ where } id(\) = I.$$

Abbreviation. We habitually write ω instead of $opn_n \omega$, for example $\omega'(I, e_1)$ instead of $m(I, e_1)$. This causes no confusion so long as the algebra being referred to is understood.

By the *closure* of a set X with respect to Ω we mean the least set A containing X such that for each ω of arity n in Ω and each a_1, \dots, a_n all in A , $\omega(a_1, \dots, a_n)$ is in A . Note that if Ω_0 is non-null then so is the closure of $\{ \}$.

We say that a set X *generates* an algebra A_Ω if A is the closure of X with respect to Ω .

Example. $\{I, e_1\}$ generates the group G_Ω^4 .

If X is any subset of A then the algebra A'_Ω generated by X is said to be a *subalgebra* of A_Ω .

Example. $\{I, e_2\}$ generates a subalgebra of G_Ω^4 whose carrier is just those two elements.

Homomorphisms

By a *homomorphism* from A_Ω to B_Ω , where A_Ω and B_Ω are algebras with the same operator set Ω and the same arity, we mean a function from the carrier of A_Ω to the carrier of B_Ω , $\phi: A \rightarrow B$, such that for each ω in Ω of arity n

$$\phi(\omega(a_1, \dots, a_n)) = \omega(\phi a_1, \dots, \phi a_n)$$

We write such a homomorphism as $\phi: A_\Omega \rightarrow B_\Omega$.

Example. If G_Ω^4 is the cyclic group of order 4 defined above and G_Ω^2 is the cyclic group of order 2, with elements $\{I, e\}$, then there is a homomorphism $\phi: G_\Omega^4 \rightarrow G_\Omega^2$ defined by $\phi I = I$, $\phi e_1 = e$, $\phi e_2 = I$, $\phi e_3 = e$. It is easy to verify that ϕ preserves multiplication, inversion and identity in the sense of the equality just given.

Certain kinds of homomorphisms are dignified with special names.

If $\phi: A_\Omega \rightarrow B_\Omega$ is one-one then ϕ is a *monomorphism*.

If $\phi: A_\Omega \rightarrow B_\Omega$ is onto B then ϕ is an *epimorphism*.

If $\phi: A_\Omega \rightarrow B_\Omega$ is one-one and onto then ϕ is an *isomorphism*.

If $\phi: A_\Omega \rightarrow A_\Omega$ then ϕ is an *endomorphism*.

We consider a set A as an algebra with an empty set Ω of operators, so that formally any function is a homomorphism, if only of sets. We shall usually use Greek letters for homomorphisms of algebras other than sets.

We now note some fundamental properties of homomorphisms and explain a convenient means of defining them. The proofs are by straightforward induction and we omit them.

Product of homomorphisms lemma. If $\phi: A_\Omega \rightarrow B_\Omega$ and $\psi: B_\Omega \rightarrow C_\Omega$ are homomorphisms then so is their composition $\phi \circ \psi: A_\Omega \rightarrow C_\Omega$.

Unique extension lemma. If X generates A_Ω and $Y \subseteq B$ then for any function $f: X \rightarrow Y$ there is at most one homomorphism $\phi: A_\Omega \rightarrow B_\Omega$ such that for all x in X , $\phi x = f x$. We say that ϕ , if it exists, is the extension of f to a homomorphism. Note that f might usefully be the null function if Ω_0 is non-null.

Definition of homomorphisms. Since ϕ is uniquely determined by f , given A_Ω and B_Ω we may define a partial function 'Extend' such that

$$\phi = \text{Extend}(A_\Omega, B_\Omega)f$$

We shall frequently define a homomorphism ϕ by writing

$$\phi: A_\Omega \rightarrow B_\Omega \text{ and } \phi x = E(x) \text{ for } x \text{ in } X$$

where $E(x)$ is some expression involving x . This is an elliptic way of saying

$$\phi = \text{Extend}(A_\Omega, B_\Omega)f \text{ where } f: X \rightarrow Y \text{ is defined by } fx = E(x)$$

We note that

- (i) Such a definition must be justified by a proof that a homomorphism ϕ does indeed exist.
- (ii) Such a definition specifies a unique ϕ , if one exists, but it gives us no means of computing ϕa for an arbitrary a in A . We shall see later that to compute ϕa we also need a rule for deriving any a from the elements of X in terms of the operations of Ω .

Extension of composition lemma. Given algebras A_Ω , B_Ω and C_Ω with X generating A_Ω , and Y generating B_Ω , $Z \subseteq C$ and functions $f: X \rightarrow Y$ and $g: Y \rightarrow Z$, then

$$\text{Extend}(A_\Omega, C_\Omega)(f \circ g) = (\text{Extend}(A_\Omega, B_\Omega)f) \circ (\text{Extend}(B_\Omega, C_\Omega)g)$$

Free algebras and word algebras

If \mathcal{C} is any class of Ω -algebras for a fixed Ω , then A_Ω in \mathcal{C} is said to be a *free algebra* on a generating set X with respect to \mathcal{C} if any function $f: X \rightarrow Y$, where $Y \subseteq B$, the carrier of some B_Ω in \mathcal{C} , can be extended to a homomorphism $\phi: A_\Omega \rightarrow B_\Omega$.

A class \mathcal{C} does not necessarily have a free algebra on X . However, if \mathcal{C} is the class of all groups there is a free group on any set X ; similarly for semigroups (one associative binary operation) and some other common kinds of algebra. For example, the free semigroup on X is isomorphic with the set of all strings on the alphabet X . We shall discuss semigroups further below.

In particular, if \mathcal{C} is the class of all Ω -algebras for a given operator set Ω with given arity, then the free algebra on X with respect to \mathcal{C} is called the Ω -*word algebra* on X and is denoted by $W_\Omega(X)$.

The word algebra always exists; so any function $f: X \rightarrow Y$, where $Y \subseteq A$, the carrier of A_Ω , can be extended to a homomorphism $\phi: W_\Omega(X) \rightarrow A_\Omega$.

One way of representing the word algebra is to associate an object w' with each word w in $W_\Omega(X)$ thus

$$(i) w' = w, \text{ if } w \text{ is in } X$$

$$(ii) w' = (\omega, w'_1, \dots, w'_n), \text{ an } n+1\text{-tuple, if } w = \omega(w_1, \dots, w_n)$$

and associate with each ω in Ω an operation ω' defined by

$$\omega'(w'_1, \dots, w'_n) = (\omega, w'_1, \dots, w'_n).$$

Programmers will recognise this as a common kind of data structure (plex, record, structure) with n components and a marker ω to indicate what kind of structure is intended. In LISP, for example, there is only one ω in Ω , say 'cons', with arity 2, and X is the set of all 'atoms'. Thus, LISP lists, in pure LISP without assignment, are $W_{\{\text{cons}\}}(\text{atoms})$; with assignment, more complex structures occur, e.g. circular lists.

MATHEMATICAL FOUNDATIONS

Derived operations

We can define new operations by composition and the use of constants.

If A_Ω is any Ω -algebra and X is an infinite set of 'variables' we may associate a *derived* operation, say ω' of arity n , with an ordered set x_1, \dots, x_n and a word w in $W_\Omega(A \cup \{x_1, \dots, x_n\})$ as follows:

$$\omega'(a_1, \dots, a_n) = (\text{Extend}(W_\Omega(x), A_\Omega) f) w, \text{ where } f: A \cup \{x_1, \dots, x_n\} \rightarrow A \text{ is defined by } f x_i = a_i \text{ and } f a = a. \text{ In } \lambda\text{-calculus notation}$$
$$\omega' = \lambda(x_1, \dots, x_n). w.$$

For example, if A_Ω is a group with operations m and inv then $c(x, y) = m(inv\ y, m(x, y))$, that is, $y^{-1} \cdot x \cdot y$, is a binary derived operation and $c_a x = a^{-1} \cdot x \cdot a$ a unary derived operation. A set Ω' of derived operations forms an algebra $A_{\Omega'}$ with carrier A . Such an algebra is said to be a *restriction* of A_Ω .

We note the following useful lemma.

Homomorphism of restrictions lemma

Given A_Ω and B_Ω and $\phi: A_\Omega \rightarrow B_\Omega$, suppose that $A_{\Omega'}$ is a restriction of A_Ω and $B_{\Omega'}$ a restriction of B_Ω , where the operations of $B_{\Omega'}$ are derived in the same way as those of $A_{\Omega'}$ replacing a by ϕa , then ϕ is a homomorphism $\phi: A_{\Omega'} \rightarrow B_{\Omega'}$.

For example if ϕ is a homomorphism of groups with $\phi a = b$, then it is also a homomorphism from the algebra with unary operations $c_a x = a^{-1} \cdot x \cdot a$ to that with unary operations $c_b x = b^{-1} \cdot x \cdot b$.

Semigroups

A *semigroup* is an algebra with one binary associative operator, which we shall denote by an infix '·' rather than a prefixed ω .

Examples

- (i) N_+ , the semigroup of positive integers under addition.

A *semigroup with identity*, also called a monoid, has an extra nullary operator 1, the identity. For any element a we have $a \cdot 1 = 1 \cdot a = a$.

Examples

- (i) N_+ , the monoid of non-negative integers under addition with the number zero as identity.
(ii) T_{or}^1 , the semigroup with identity of truth values, $\{true, false\}$ under logical *or*, with *false* as identity. We have $false \vee a = a \vee false = a$.
(iii) T_{and}^1 , the same replacing *or* by *and* and interchanging *true* and *false*.

A *semigroup with identity and a zero*, has in addition to '·' and 1 a nullary operator 0, the zero. For any a we have $a \cdot 0 = 0 \cdot a = 0$. A semigroup can have only one zero, and we can always add a zero to a semigroup without one.

Examples

- (i) N_\times the semigroup with identity and zero of non-negative integers under multiplication, with the number one as identity and the number zero as zero (hence the nomenclature).

(ii) The semigroup with identity T_{or}^1 can be made into one with identity and zero $T_{or}^{1_0}$ by taking zero to be *true*. We have $true \vee a = a \vee true = true$.

(iii) Similarly for T_{and}^1 .

We shall denote the free semigroup on the set X by $\sum X$. Recall that any function $f: X \rightarrow \text{Carrier of } S$, where S is another semigroup defines a unique homomorphism $\phi: \sum X \rightarrow S$. We shall often define ϕ by just saying

$$\begin{aligned} & \phi: \sum X \rightarrow S \\ & \phi x = E(x) \text{ for } x \text{ in } X, \end{aligned}$$

using implicitly the justification that $\sum X$ is free so that ϕ exists.

The free semigroup $\sum X$ can be represented by the set of all finite sequences of elements of X under concatenation.

Programmers will note that many list-processing operations act on lists considered as a semigroup rather than as a word algebra, that is, the word algebra is being used to represent a semigroup. This is even more apparent in string processing languages such as SNOBOL.

We sometimes need the free semigroup with identity, which we shall denote by $(\sum X)^1$, and the free semigroup with both identity and zero, which we shall denote by $(\sum X)^{1_0}$.

A particularly important kind of semigroup is a *mapping semigroup* over a set X , that is, a set of functions in $(X \rightarrow X)$ with composition as operation and closed under composition. Thus if $f: X \rightarrow X$ and $g: X \rightarrow X$ then $f.g = f \circ g$. We denote this semigroup of all functions from X to X under composition by $F_R(X)$ (Functions under Right multiplication). We write $(F_R(X))^1$ for the semigroup with identity of all functions from X to X with the identity function as identity.

We shall see that an automaton can be characterised by a homomorphism from a free semigroup of 'inputs' to a mapping semigroup of 'state transformations'. A treatment of semigroups and their application to automata may be found in Arbib (1968).

4. ELEMENTARY EXAMPLES IN PROGRAMMING

Let us take a look at a few functions commonly met with in programming and express them as homomorphisms. Our examples are mostly taken from list-processing (considering single-level lists rather than 'list-structures') and involve homomorphisms of semigroups. Many simple list functions, written as, say, LISP recursive functions, show a repetitive pattern of conditionals and recursion. This pattern makes them easy to write once it is recognised. Expressing the functions as homomorphisms abstracts this pattern as a mathematical entity.

We shall use X and Y for arbitrary sets. In the examples we shall often take these to be integers or reals. We recall the following names for semigroups.

- $(\sum X)^1$, the free semigroup with identity over X (lists over X , including nil)
- N_+ , the semigroup with identity of non-negative integers under addition
- T_{or}^1 , the semigroup with identity of truth values under logical or
- $F_R(X)$, the semigroup of functions in $X \rightarrow X$ under composition
- $(F_R(X))^1$, the semigroup with identity of functions $X \rightarrow X$ under composition with the identity function as identity.

We define each of our functions first informally, then as a recursive function in the notation of Landin (1966), then as a homomorphism using the notation described in section 3. We shall presently exhibit a more explicit notation suitable for presenting the definitions to a compiler.

- (a) *map*, which applies a function f to each member of a list, for example
 $map(sqrt, (1,2,3,4)) = (1.00, 1.41, 1.73, 2.00)$
 let recursive $map(f,x) = \text{if } null\ x \text{ then } nil$
 else $cons(f(hd\ x), map(f, tl\ x))$

Homomorphism. If $f: X \rightarrow Y$, the unary function map_f is
 $map_f: (\sum X)^1 \rightarrow (\sum Y)^1$
 $map_f x = f\ x$ for x in X

- (b) *sub*, which selects all elements of a list which satisfy a predicate p , for example
 $sub(prime, (3,2,4,7,8)) = (3,2,7)$
 let recursive $sub(p,x) = \text{if } null\ x \text{ then } nil$
 else if $p(hd\ x)$ then $cons(hd\ x, sub(p, tl\ x))$
 else $sub(p, tl\ x)$

Homomorphism. The function sub_p of one argument is
 $sub_p: (\sum X)^1 \rightarrow (\sum X)^1$
 $sub_p x = \text{if } p\ x \text{ then } x \text{ else } 1$ for x in X
 (1 being the empty list, nil)

- (c) *sigma*, which applies a given function f to each member of a list of integers and adds the results, for example

$$sigma(square, (1,2,3,4)) = 30$$

Recursive definition. Obvious.

Homomorphism.

$$sigma_f: (\sum N)^1 \rightarrow N_+$$

$$sigma_f n = f\ n \quad \text{for } n \text{ in } N$$

Now if we define a function *sum* so that, e.g., $sum((1,4,9,16)) = 30$

$$sum: (\sum N)^1 \rightarrow N_+$$

$$sum\ n = n \quad \text{for } n \text{ in } N,$$

we see that $sigma_f = map_f \circ sum$. Proof: For any n in N , $sigma_f\ n = (map_f \circ sum)n$, and therefore $sigma_f = map_f \circ sum$ for any element in $(\sum N)^1$ by the 'extension of composition lemma'. It is worth re-

marking that this little proof involves inspection of the effect on the generator set, and then avoids any inductive argument by appealing to a standard lemma.

- (d) *exists*, which tests whether any member of a set satisfies a given predicate, for example $exists(prime, (4,5,6,8)) = true$

Recursive definition. Obvious.

Homomorphism.

$$exists_p: (\sum X)^1 \rightarrow T_{or}^1$$

$$exists_p x = p x \text{ for } x \text{ in } X$$

The function *all* can be defined similarly using T_{and}^1 .

- (e) *exec*, a function which takes a list of elements (x_1, x_2, \dots, x_n) together with a function f which turns each element x into a function $f x$, producing as its result the composition of these functions, $f x_1 \circ f x_2 \circ \dots \circ f x_n$. For example, if $add\ i\ j = i + j$, so that $add\ i$ is a unary function, then $exec(add, (1,2,3))\ 0 = (add\ 1 \circ add\ 2 \circ add\ 3)\ 0 = 3 + (2 + (1 + 0))$

Suppose that $f: X \rightarrow Y$ and I is the identity function in $Y \rightarrow Y$.

let recursive $exec(f, x) =$

if null x then I

else $f(hd\ x) \circ exec(f, tl\ x)$

Homomorphism.

$$exec_f: X \rightarrow (F_R(Y))^1$$

$$exec_f x = f x$$

- (f) *Functions of integers*. Primitive recursive functions of integers can be expressed using homomorphisms. We let N_{s0} be the algebra of non-negative integers with a nullary operation, 0, and a unary operation, *successor*. A restricted form of the primitive recursion schema, equivalent to the usual schema given primitive functions to manipulate pairs, can be written with y as a parameter thus

$$f_y 0 = a_y, \quad f_y(\text{successor } x) = h_y(f_y x)$$

Let N_{ha} be the algebra of integers with a nullary operation a and a unary operation h_y . Then f_y is

$$f_y: N_{s0} \rightarrow N_{ha}$$

This defines f_y since N_{s0} is generated by the empty set.

It is a fundamental property of non-negative integers that N_{s0} is a free algebra and hence this always defines a homomorphism f_y for any h_y and a .

- (g) *Substitution in expressions*. In symbol manipulation tasks, such as theorem-proving, we often deal with a set of 'expressions' involving a set Ω of operators and a set X of variables, that is, a word algebra $W_\Omega(X)$. Consider a function to substitute expressions for variables

in a given expression, for example to put $\omega_1(y,y)$ for x and ω_2x for y in $\omega_1(\omega_2x_2,\omega_1y)$ giving $\omega_1(\omega_2(\omega_1(y,y)), \omega_1(\omega_2x))$.

Suppose $f: X \rightarrow \text{carrier of } W_\Omega(X)$ is the function specifying the substitutions to be made, then the function to perform them on a given expression is

$$\begin{aligned} \phi &: W_\Omega(X) \rightarrow W_\Omega(X) \\ \phi x &= f x. \end{aligned}$$

that is, the extension of f to a homomorphism of $W_\Omega(X)$

These examples suggest that definitions by homomorphisms are quite versatile, indeed it is possible to do quite a lot of programming without resorting to recursive definitions or iteration, as we shall show in our expression compiler.

5. PROGRAMMING IN TERMS OF HOMOMORPHISMS

We remarked earlier that, given algebras A_Ω and B_Ω and a set X which generates A_Ω , a function $f: X \rightarrow B$ can be extended to a homomorphism ϕ in at most one way, and that this enables us to define a partial function *Extend* such that $\text{Extend } f = \phi$, if the homomorphism exists. We have just shown how a number of common functions can be defined as the extensions of simpler functions to homomorphisms. We now show how to make these definitions more explicit, indeed suitable for presentation to a computer, by giving an algorithmic definition of the function *Extend*.

We shall start by programming a less general function than *Extend*, one which deals only with homomorphisms of semigroups. We hope in this way to make the definition of *Extend* easier to follow.

Recall the function *map* defined in the last section by

$$\begin{aligned} \text{let recursive } \text{map}(f,x) &= \text{if null } x \text{ then nil} \\ &\quad \text{else cons}(f(\text{hd } x), \text{map}(f,\text{tl } x)) \end{aligned}$$

If we generalise this by making *null*, *hd*, *tl*, *nil* and *cons* parameters we get a more general function:

$$\begin{aligned} \text{let } \text{genmap}(g\text{null},g\text{hd},g\text{tl},g\text{nil},g\text{cons}) &= g\text{map} \\ \text{where recursive } g\text{map}(f,x) &= \text{if } g\text{null } x \text{ then } g\text{nil} \\ &\quad \text{else } g\text{cons}(f(g\text{hd } x), g\text{map}(f,g\text{tl } x)) \end{aligned}$$

Now $\text{genmap}(\text{null},\text{hd},\text{tl},\text{nil},\text{cons})$ is simply *map*. The reader can verify that $\text{genmap}(\text{null},\text{hd},\text{tl},\text{false},\text{or})$ is *exists* and that $\text{genmap}(\text{null},\text{hd},\text{tl},0,+)$ is *sigma*. Thus *genmap* is capable of producing a wide class of functions.

However, we want the still more general function *Extend*, which can be used to produce any homomorphism of Ω -algebras, with the semigroup homomorphisms produced by *genmap* as a special case.

We want $\text{Extend}(A_\Omega, B_\Omega)f$ to be ϕ , the unique extension of f to a homomorphism. How are we to represent the algebras A_Ω and B_Ω ?

To compute ϕa for a in A we need to decompose a into a construction in terms of Ω and X the generating set of A . For this we may use a decomposition function d_{AX} to produce from any a in A but not in X an operator ω and an n -tuple (a_1, \dots, a_n) such that $a = \omega(a_1, \dots, a_n)$

$$d_{AX}: A - X \rightarrow \bigcup_{n=0 \text{ to } \infty} (\Omega_n \times A^n)$$

$$d_{AX} a = (\omega, (a_1, \dots, a_n)) \text{ such that } a = \omega(a_1, \dots, a_n)$$

We also need a predicate p_X to test a for membership of X

$$p_X: A \rightarrow \{true, false\}$$

$$p_X a = true \text{ iff } a \text{ in } X.$$

These two functions, d_{AX} and p_X , suffice to represent A_Ω . To represent B_Ω we just need the function opn_B which takes any operator ω in Ω of arity n into the corresponding operation of B_Ω , that is, into a function from B^n to B .

$$opn_B: \Omega \rightarrow \bigcup_{n=0 \text{ to } \infty} (B^n \rightarrow B)$$

We may as well represent each of the algebras A_Ω and B_Ω by a triple (d, p, opn) even though we do not use all the components.

Now we define *Extend* and give an example of its use. *

let $Extend(A_\Omega, B_\Omega) f =$

let $(d_{AX}, p_X, opn_A) = A_\Omega$

and $(d_{BY}, p_Y, opn_B) = B_\Omega$

ϕ where recursive $\phi a =$ if $p_X a$ then $f a$

else let $(\omega, (a_1, \dots, a_n)) = d_{AX} a$

$(opn_B \omega)(\phi a_1, \dots, \phi a_n)$

For example, since lists of atoms are $(\sum atoms)^1$ we could define the function *map* as $Extend((\sum atoms)^1, (\sum atoms)^1)$ since it extends a function from atoms to atoms to a function from lists to lists. We represent $(\sum atoms)^1$ by (d, p, opn) , where

$d a =$ if null a then ("NIL", ())
else ("CONCAT", (hd a , tl a))

$p a =$ atom a

$opn a =$ if $a =$ "NIL" then nilf

else if $a =$ "CONCAT" then concat

where nilf() = nil and concat joins lists, concat((1,2), (3,4)) = (1,2,3,4)

$Extend((d, p, opn), (d, p, opn))$ square applied to a , which is a list of numerical atoms or an atom or nil, first tests a with p to see whether it is an atom. If not it decomposes a with d obtaining either the operator NIL and no elements of a or the operator CONCAT and two shorter lists a_1 and a_2 . If NIL then nilf is applied to produce nil, if CONCAT and (a_1, a_2) then ϕ (that is, map) is applied recursively to a_1 and a_2 and concat is applied to the results.

* Note added in proof: We have tested *Extend* by translating and running it in Pop-2 (Burstall & Popplestone 1968).

We have demanded only that the decomposition function d satisfy

$$d = \omega(a_1, \dots, a_n), \text{ where } (\omega, (a_1, \dots, a_n)) = d a$$

Provided that f really is extendable to a homomorphism it does not matter which function d we use, so long as it satisfies this relationship and converges, that is, repeated applications of d eventually produce elements in X . An example may make this clearer.

Consider *logarithm* which is a homomorphism from the semigroup of positive reals under multiplication to the semigroup of reals under addition, since $\log(x \times y) = \log x + \log y$. Suppose we know a function *logtable* which gives $\log x$ for $x < 2$. Then we write

$$\begin{aligned} \text{let } \log x = & \text{if } x < 2 \text{ then } \text{logtable}(x) \\ & \text{else let } (y, z) = \text{factors}(x) \\ & \log y + \log z \end{aligned}$$

It does not matter which function *factors* we use so long as $y \times z = x$ and the computation terminates, for example it does not matter whether *factors* $(12) = (3, 4)$ or *factors* $(12) = (2, 6)$.

In general for any a define two decomposition functions d and d'

$$d a = (\omega, (a_1, \dots, a_n)) \text{ and } d' a = (\omega', (a'_1, \dots, a'_n))$$

$$\text{Then } a = \omega(a_1, \dots, a_n) = \omega'(a'_1, \dots, a'_n)$$

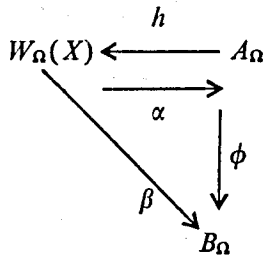
But since ϕ is a homomorphism

$$\omega(\phi a_1, \dots, \phi a_n) = \phi a = \omega'(\phi a'_1, \dots, \phi a'_n)$$

These are precisely the expressions occurring in *Extend* when we supply d or d' as parameter; since they are equal the result of *Extend* is unaffected.

In general it may be difficult or impossible to find a function d which converges for any element of a (see the discussion of the word problem in Cohn 1965). What is a sufficient condition for d to converge?

If X generates A_Ω , there is a unique homomorphism from $W_\Omega(X)$ to A_Ω , $\alpha: W_\Omega(X) \rightarrow A_\Omega$. Consider any function h which is a left inverse of α , that is, $h \circ \alpha = I_A$, the identity function on A . Now let β be the unique extension of f to a homomorphism from $W_\Omega(X)$ to B_Ω . Then $\alpha \circ \phi = \beta$, and hence $h \circ \beta = h \circ \alpha \circ \phi = I_A \circ \phi = \phi$, the required homomorphism.



This gives us a way of computing ϕ which avoids problems about convergence by using the word algebra. Suppose that given h we choose d so that

$$h a = \omega(h a_1, \dots, h a_n) \quad \text{where } (\omega, (a_1, \dots, a_n)) = d a,$$

a sufficient condition for d to converge. Since α has in general more than one h there is more than one possible d , and any such d is guaranteed to converge.

6. CORRECTNESS OF A SIMPLE COMPILER

As an illustration of the use of an algebraic point of view we shall prove the correctness of a simple compiler for expressions, comparable to the one whose correctness was proved by McCarthy and Painter (1967) using the method of recursion induction. For other research on correctness proofs see Painter (1967), Kaplan (1967), Floyd (1967), Burstall (1969) and Cooper (1969), with the references therein. We shall do this proof in several stages by proving the correctness of compilers for various intermediate machines.

Table 1 shows an example of the action of these machines. The problem is to evaluate expressions composed of variables from a set X and operators from a set Ω (Table 1a). We assume that the expressions have already been syntactically analysed into their operator-operand structure. The operators may include nullary operators, that is, constants. We are given the value for each variable. In general we consider the expressions as elements of some Ω -word algebra and the values as the elements of an arbitrary Ω -algebra.

Table 1. Example of various methods of evaluating an expression.

(a) Set of operators = +, \times , 0, 1, 2, 3, etc.

Set of variables $X = u, v$, etc.

Values of variables: value $u = 5$, value $v = 4$, etc.

(b) Expression = $u + (6 \times v)$

Value = $5 + (6 \times 4) = 29$

(c) Stack machine

Program Stack sequence

empty

u

5

6

5.6

v

5.6.4

\times

5.24

+

29

result = 29

(d) Store-pointer machine

Program Store-pointer sequence

{{(1,*),(2,*),(3,*),...},0}

u

{{(1,5),(2,*),(3,*),...},1}

6

{{(1,5),(2,6),(3,*),...},2}

v

{{(1,5),(2,6),(3,4),...},3}

\times

{{(1,5),(2,24),(3,4),...},2}

+

{{(1,29),(2,24),(3,4),...},1}

result = 29

(e) Address-program machine

Program	Store sequence
	$\{(1,*),(2,*),(3*), \dots\}$
$(u,0)$	$\{(1,5),(2,*),(3*), \dots\}$
$(6,1)$	$\{(1,5),(2,6),(3*), \dots\}$
$(v,2)$	$\{(1,5),(2,6),(3,4), \dots\}$
$(\times,3)$	$\{(1,5),(2,24),(3,4), \dots\}$
$(+,2)$	$\{(1,29),(2,24),(3,4), \dots\}$
result=29	

(f) Conventional machine

Program	Store-accumulator sequence
	$\{(1,*),(2,*),(3*), \dots\},*$
<i>cla u</i>	$\{(1,*),(2,*),(3*), \dots\},5)$
<i>sto 1</i>	$\{(1,5),(2,*),(3*), \dots\},5)$
<i>cla 6</i>	$\{(1,5),(2,*),(3*), \dots\},6)$
<i>sto 2</i>	$\{(1,5),(2,6),(3*), \dots\},6)$
etc.	

Table 1 (contd.) Note: * means an arbitrary value

The simplest method (Table 1b) is to use the evaluation rule directly, that is, to use an 'interpreter'. The evaluation rule is specified as a homomorphism between the expressions and the values.

The next method (Table 1c) is to compile a 'reversed Polish' program for a machine with a 'stack' (sometimes called a nest or pushdown). The stack is a finite sequence of values, considered as an element of the free semigroup over the values. Execution of an instruction in this program transforms the stack, affecting the right-hand end of it by either loading the value of a variable or performing a k -argument operation on the last k elements and replacing them by the single element which results; if $k=0$ the effect is to load the value of a constant.

The third method (Table 1d) is to compile the same 'reversed Polish' program, but to execute it on a machine with a 'virtual' stack represented by some area of addressable store (we choose locations 1 onwards) and a

pointer, that is, a register containing at any time the serial number of the last location used for the stack. In Table 1d this is shown as a pair consisting of a store and a pointer; the store itself is a set of pairs each consisting of an address (a positive integer) and its contents (a value). The store is thought of as a function from addresses (positive integers) to values. It is used only to represent the stack.

The fourth method (Table 1e) is to compile an 'address-program' for a machine whose instructions consist of a variable or an operation together with an address showing where in the store the value of the variable is to be loaded or the operation performed. The address is the value, now computed at compile-time, of an imaginary pointer. Thus '(v,2)' causes the value of v to be loaded into the store at address $2+1$. The essential difference is that computation of the value of the pointer is now done at compile-time, saving work at execute-time. This is an example of the general technique used in efficient compilers of computing as much as possible at compile-time, for example determining the types of expressions at compile-time rather than testing them at execute-time.

The fifth method (Table 1f) is to compile for a 'conventional' machine with an accumulator and instructions such as '*clau*' to load the value of u into the accumulator, '*sto 3*' to store the contents of the accumulator in address 3, '*add 2*' to add the contents of address 2 to the accumulator. This machine only differs in a rather trivial way from the previous one. The instructions of the address-program machine may be thought of as 'macros' which have to be expanded to obtain the program for the conventional machine.

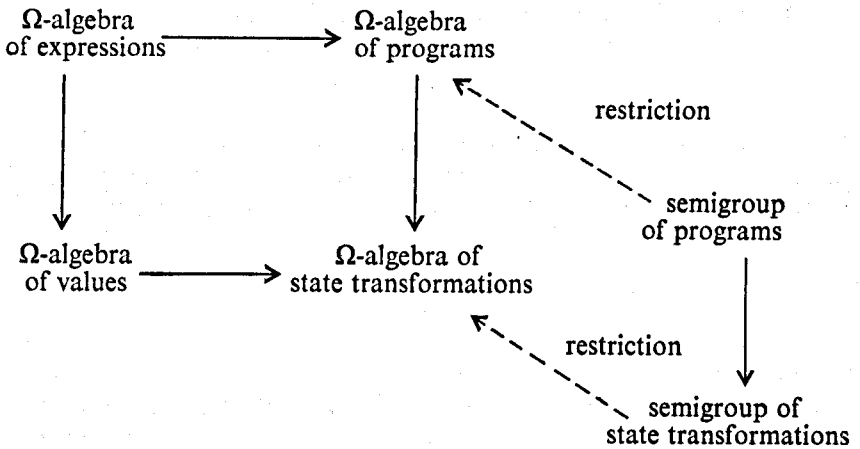
Our final aim is to develop a compiler for evaluating expressions on this conventional machine. This compiler uses the compilers for the intermediate machines as components and the proof of its correctness depends on the proofs of correctness for the intermediate compilers.

To effect the transition from interpretation of expressions to execution of a reversed Polish program on a stack machine we shall need some elementary algebraic results about embedding Ω -algebras in semi-groups. To use a store-pointer machine to simulate the stack machine we need a simple induction lemma about homomorphisms of semigroups which we can conveniently state in the language of automata theory. To go over to the address-program machine we need a further result about semigroups, again expressed in automata terms. In each case we develop the abstract theory first and then apply the result to the compilation problem. This shows up the generality of the treatment and leaves us free to apply the lemmas to more than one problem in the area of programming.

Transition to a stack machine

Let us consider the transition from expressions and values (Table 1b) to reversed Polish program and stack transformations (Table 1c) in an algebraic context.

Expressions and their corresponding value algebra are Ω -algebras with some arbitrary set Ω of operators. We wish to represent them using sequences of instructions (for expressions) and sequences of state transformations (for the value algebra). These are both semigroups and so there can be no homomorphism to them from the Ω -algebras. The device used to obtain an Ω -algebra is to define some derived operations over the corresponding semigroup, one operation for each ω in Ω . We can then get a homomorphism from the original Ω -algebra to the Ω -algebra constructed out of the semigroup. What is more, the homomorphism from the expression algebra to the value algebra can be mirrored by a homomorphism between the corresponding algebras constructed from semigroups. This homomorphism of the constructed algebras is itself a restriction of a homomorphism of the underlying semigroups.



Embedding an Ω -algebra in a semigroup

In this section we develop the algebraic results needed for the transition to a stack machine. Suppose that S is a semigroup, Y a subset of its carrier, Ω a set of operators and that there is a function $f: \Omega \rightarrow \text{Carrier of } S$. Then we can define a derived algebra of S which is an Ω -algebra, say $E_\Omega(S, Y, f)$, as follows. Take Y as the set of generators of the Ω -algebra and let its operations be defined by derived operations of the semigroup, thus:

$$\omega(s_1, s_2, \dots, s_k) = s_1 \cdot s_2 \cdot \dots \cdot s_k \cdot f\omega, \text{ where } s_i \in S \text{ and } k = \text{arity } \omega$$

Here $f\omega$ is the semigroup element corresponding to ω . In choosing f we determine which Ω -algebra we get.

Looking at it from another point of view we might ask whether given any Ω -algebra we can find a means of constructing an algebra isomorphic to it from a semigroup; that is, given an Ω -algebra A_Ω can we find a semigroup S , a subset Y of its carrier and a function f such that $E_\Omega(S, Y, f)$ has a sub-algebra isomorphic to A_Ω , that is, there is a monomorphism

$$\xi: A_\Omega \rightarrow E_\Omega(S, Y, f)$$

We shall call this *embedding* A_Ω in the semigroup and show that it is always possible (cf. Cohn 1965, Section IV.4).

Consider first the special case where A_Ω is the word algebra $W_\Omega(X)$. Then we can use the semigroup $S_W = \sum(X \cup \Omega)$, and put $f_W \omega = \omega, f_W x = x$. From these we define $E_\Omega(S_W, X, f_W)$ as above. Now put

$$\begin{aligned} \xi_W : W_\Omega(X) &\rightarrow E_\Omega(S_W, X, f_W) \\ \xi_W(x) &= x \quad \text{for } x \text{ in } X \end{aligned}$$

Since $W_\Omega(X)$ is a word algebra with generator set X the definition of $\xi_W(x)$ serves to define ξ_W as a homomorphism. Cohn shows that it is an isomorphism, but we shall not need this fact here.

Consider now the general case where we are given any Ω -algebra A_Ω . To embed this in a semigroup we need a slightly more elaborate construction, using a semigroup of state transformations.

Let A be the carrier of A_Ω . We choose as our semigroup for the embedding S_A , the semigroup of transformations of sequences of elements of A . We include in the sequences the empty sequence 1 and a zero element 0. The zero may be thought of as an 'error' indication signifying that an operation ω has been applied to too few arguments. Thus we put

$$S_A = F_R((\sum A)^{10})$$

that is, the mapping semigroup over the carrier of the free semigroup with identity and zero.

To construct the Ω -algebra we choose as generators $Y_A \subseteq S_A$ the set of transformations ρ_a

$$\begin{aligned} \rho_a : (\sum A)^{10} &\rightarrow (\sum A)^{10} \\ \rho_a u &= u \cdot a \quad \text{for each } a \text{ in } A \text{ and } u \text{ in } (\sum A)^{10} \end{aligned}$$

and define a function f_A , giving a semigroup element for each ω .

$$\begin{aligned} f_A \omega &= \sigma_\omega \\ \text{where } \sigma_\omega u &= \text{if } u = a_1 \cdot a_2 \cdot \dots \cdot a_{n-k} \cdot a_{n-k+1} \cdot \dots \cdot a_n \\ &\quad \text{then } a_1 \cdot a_2 \cdot \dots \cdot a_{n-k} \cdot \omega(a_{n-k+1}, \dots, a_n) \\ &\quad \text{otherwise } 0. \end{aligned}$$

The Ω -algebra is $E_\Omega(S_A, Y_A, f_A)$.

We shall now show that there is a monomorphism

$$\xi_A : A_\Omega \rightarrow E_\Omega(S_A, Y_A, f_A)$$

defined by

$$\xi_A a = \rho_a$$

This is a homomorphism because

$$\begin{aligned} \xi_A(\omega(a_1, \dots, a_k)) &= \rho_{\omega(a_1, \dots, a_k)} \\ \text{and } \omega(\xi_A a_1, \dots, \xi_A a_k) &= \omega(\rho_{a_1}, \dots, \rho_{a_k}) \\ &= \rho_{a_1} \cdot \dots \cdot \rho_{a_k} \cdot \sigma_\omega \\ &= \rho_{\omega(a_1, \dots, a_k)} \end{aligned}$$

ξ_A is a monomorphism because to each ρ_a there corresponds a unique a , namely $\rho_a 1$, where 1 is the empty sequence. So there is an isomorphism

$$\begin{aligned} \eta_A: im \xi_A &\rightarrow A & im \xi_A \text{ being a subalgebra of } E_\Omega(S_V, Y_V, f_V) \\ \eta_A \rho_a &= \rho_a 1 \end{aligned}$$

We shall need to make use of one further fact: a homomorphism of semi-groups induces a homomorphism of the Ω -algebras constructed from them (by the homomorphism of restrictions lemma), that is, if S_1 and S_2 are semi-groups then a homomorphism of semigroups $\phi: S_1 \rightarrow S_2$ can be restricted to a homomorphism of Ω -algebras $\eta: E_\Omega(S_1, Y_1, f_1) \rightarrow E_\Omega(S_2, Y_2, f_2)$, provided that ϕ maps Y_1 into Y_2 .

Compiling expressions for a stack machine

We now use these techniques to prove the correctness of a compiler for evaluating expressions using a stack machine (see Table 1c).

We regard the expressions as elements of a word algebra $W_\Omega(X)$ with a set of operators Ω and a set of variables X . We call $W_\Omega(X)$ the expression algebra and assume that the value of an expression is given by a homomorphism

$$\alpha: W_\Omega(X) \rightarrow V_\Omega,$$

where V_Ω , which we shall call the 'value algebra', has as its carrier V , the set of all values of expressions. For any variable x in X , αx is the value of x .

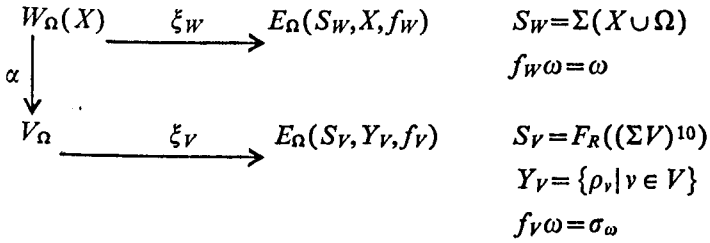
Direct computation of the value of an expression using this homomorphism may be thought of as interpreting the expression. We now show how to compile it into a reversed Polish program, and then execute the instructions in this program by carrying out appropriate transformations of the stack.

We first embed $W_\Omega(X)$, the expression algebra, in the free semigroup $S_W = \Sigma(X \cup \Omega)$, as explained in the last section. The elements of this semigroup are programs, that is, sequences of instructions of two kinds, namely 'x' to load the value of the variable x on the stack and ' ω ' to perform the operation ω on the top elements of the stack. The algebra $E_\Omega(S_W, X, f_W)$ has as elements a subset of these programs, the 'well-formed programs', but its operations are not simple concatenation. They are the appropriate compiling operations corresponding to concatenating a number of programs representing subexpressions and concatenating the appropriate operator, for example, if '+' is in Ω then the corresponding derived operation of the semigroup is 'compileplus', where *compileplus* $(u_1, u_2) = u_1 \cdot u_2 \cdot +$.

Next we embed V_Ω , the value algebra, in the semigroup $S_V = F_R((\Sigma V)^{10})$ as explained above. ΣV is the set of all stacks, that is, sequences of values, 1 is the empty stack and 0 is the error state, that is, 'stack underflow'. Thus S_V is the semigroup of stack transformations under functional composition.

Y_V and f_V are formed in the same way as Y_A and f_A in the previous section, that is, Y_V is the set of 'load value v on the stack' transformations $\{\rho_v | v \in V\}$ and $f_V \omega = \sigma_\omega$, the 'do ω to the top of the stack' transformation. The algebra $E_\Omega(S_V, Y_V, f_V)$ has as elements a subset of these stack transformations, the 'well-formed' transformations, but its operations are not simple composition. They are the operations of composing transformations and then composing the result with σ_ω .

The situation so far may be pictured thus

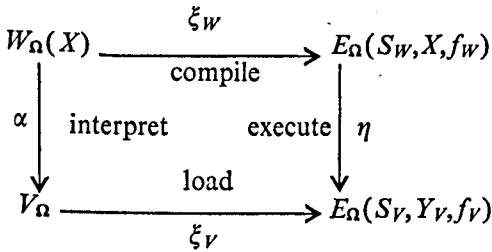


To complete the diagram we define a homomorphism of semigroups from S_W to S_V , knowing that this will induce a homomorphism of the corresponding Ω -algebras.

$$\begin{aligned}
 \phi : S_W &\rightarrow S_V \\
 \phi x &= \rho_{xx} \quad \text{and} \quad \phi \omega = \sigma_\omega
 \end{aligned}$$

Since S_W is the free semigroup on $X \cup \Omega$ this does define a homomorphism.

We now restrict ϕ to a function η from the carrier of $E_\Omega(S_W, X, f_W)$, a subset of the carrier of S_W , to the carrier of $E_\Omega(S_V, Y_V, f_V)$. We showed in the last section that η is a homomorphism of the Ω -algebras. This gives the diagram

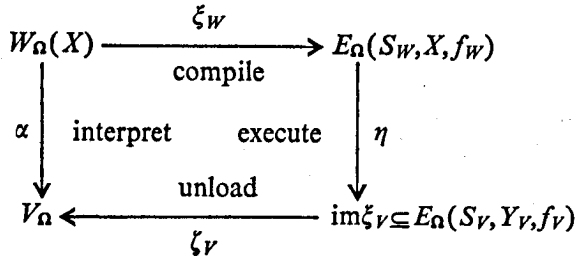


To show that this diagram commutes, that is, that $\xi_W \circ \eta = \alpha \circ \xi_V$ we need only verify that this holds for the generators X . Both $\xi_W \circ \eta$ and $\alpha \circ \xi_V$ take x to ρ_{xx} . Hence the diagram commutes.

This is the required correctness result except that the homomorphism ξ_V really goes in the wrong direction for our purpose. However, we showed in the last section that ξ_V is a monomorphism and hence there is an isomorphism, say $\zeta_V: im \xi_V \rightarrow V_\Omega$. Now we may assume that α is an epimorphism (onto V_Ω)

MATHEMATICAL FOUNDATIONS

and hence $im \xi_V = im (\alpha \circ \xi_W) = im (\xi_W \circ \eta)$. In fact it is easy to show that $im \xi_V = E_\Omega(S_V, Y_V, f_V)$. We obtain the commutative diagram



This expresses the correctness of compilation, execution and unloading for the stack machine.

Simulating one state machine by another

In order to describe the transition from a stack machine to a machine with a store and a pointer, it will be convenient to introduce some terminology and definitions from the algebraic theory of automata and prove a simple lemma. This apparatus will continue to be useful later on when we consider the transitions to an address-program machine and to a conventional machine.

First we define a *state machine* as a set of inputs each of which causes a specified transformation of states. Given an input and the current state we know the following state. The initial state of the machine is specified and we are interested in the final state after supplying a sequence of inputs. Expressing this algebraically:

A state machine M is a set Q of states with a distinguished initial state q_0 , a set I of inputs and a homomorphism $\phi : \sum I \rightarrow F_R(Q)$. Schematically:



We shall call ϕ the execution homomorphism since it tells us the action of each instruction and hence the action of each sequence of instructions.

Given a machine M we perform computations by giving it a sequence of inputs t in $\sum I$ and looking at the resulting state $q = \phi t q_0$. We may wish to perform the same computations using some other machine M' to simulate M . We use Q', q'_0, I' and ϕ' to denote the components of M' . M' can simulate M if we have a function θ to translate a sequence of inputs t for M into a sequence t' for M' . We also need a function h to translate the resulting state $q' = \phi' t' q'_0$ of M' into a resulting state q of M . This leads to the definition:

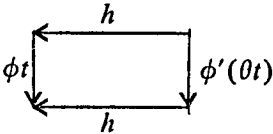
A machine M can be simulated by a machine M' if there is a homomorphism $\theta : \sum I \rightarrow \sum I'$ and a function $h : \bar{Q}' \rightarrow Q$, onto Q , where \bar{Q}' is the

subset of Q' consisting of $\phi'(\theta t)q'_0$ for all t in ΣI , such that

- (i) $h q'_0 = q_0$
- (ii) $\phi t q_0 = h(\phi'(\theta t)q'_0)$ for any t in ΣI .

We now consider the conditions under which one state machine can be simulated by another. First we note that if M' simulates the action of individual inputs to M then it simulates the action of sequences of inputs.

Lemma. If $h \circ \phi i = \phi'(\theta i) \circ h$ for all i in I , then $h \circ \phi t = \phi'(\theta t) \circ h$ for all t in ΣI , that is, the following diagram commutes:



Proof. If $t=i$, the result is immediate.

Assume as induction hypothesis that $h \circ \phi t = \phi'(\theta t) \circ h$.

Then $h \circ \phi(t \cdot i) = h \circ \phi t \circ \phi i = \phi'(\theta t) \circ h \circ \phi i = \phi'(\theta t) \circ \phi'(\theta i) \circ h = \phi'(\theta(t \cdot i)) \circ h$, as required.

Corollary. If $h \circ \phi i = \phi'(\theta i) \circ h$, then M can be simulated by M'

Proof. By the lemma $h \circ \phi t = \phi'(\theta t) \circ h$, therefore $\phi t(h q_0) = h(\phi'(\theta t)q'_0)$

We make two remarks, which are easy to prove. Although we shall not need these results they may serve to clarify the situation.

Remark. If M can be simulated by M' then $h \circ \phi i = \phi'(\theta i) \circ h$

Remark. If S is the subsemigroup of $F_R(Q)$ whose carrier is $im \phi$ and \bar{S}' is the subsemigroup of $F_R(Q')$ whose carrier is $im(\theta \circ \phi')$ then if M can be simulated by M' there is a homomorphism $\psi : \bar{S}' \rightarrow S$.

This last remark suggests an analogy between the homomorphisms ξ_w and ζ used in embedding Ω -algebras in semigroups and the homomorphisms θ and ψ used in simulating one semigroup state machine by another.

Transition from the stack machine to the store-pointer machine

We now use the corollary established in the last section to move from the stack machine (Table 1c) to the store-pointer machine (Table 1d). We must define these machines as state machines and then show how the first can be simulated by the second.

The stack machine. This machine has already been described when we showed how to evaluate expressions by embedding them in semigroups. In state machine terms it is as follows:

Input set: $I = \Omega \cup X$

State set: $Q = (\sum V)^{10}$, $q_0 = 1$ (that is, all stacks and an error state, the identity 1 being the empty stack)

Execution homomorphism: ϕ defined by

$\phi \omega = \omega'$,

MATHEMATICAL FOUNDATIONS

$$\text{where } \omega'(v_1 \cdot v_2 \cdot \dots \cdot v_{n-k} \cdot v_{n-k+1} \cdot \dots \cdot v_n) = \begin{array}{l} v_1 \cdot v_2 \cdot \dots \cdot v_{n-k} \cdot \omega(v_{n-k+1}, \dots, v_n) \\ \text{or } 0 \text{ if } n < k \\ \text{where } k = \text{arity } \omega \end{array}$$

$$\phi x(v_1, \dots, v_n) = v_1 \cdot v_2 \cdot \dots \cdot v_n \cdot \alpha x$$

The store-pointer machine. This machine has a store, that is, an infinite set of locations numbered from 1 upwards, and a pointer, that is, the number of one of these locations.

First we need to define stores as members of a set $S = (N^+ \rightarrow V)$, regarded as a function from positive integers to values. Thus, if s is in S , the contents of location number n is $s n$. We define a function to perform assignment of the value to a store location with a given number

$$\text{assign} : N^+ \times V \rightarrow (S \rightarrow S)$$

that is, an integer and a value specify a store transformation

$$\text{assign}(n, v)s = s' \text{ where } s'n' = \text{if } n' = n \text{ then } v \text{ else } s n'$$

For example, $\text{assign}(2, 3.14)$ applied to a store whose function table is $\{(1, 5.41), (2, 0.01), (3, 6.77), \dots\}$ is $\{(1, 5.41), (2, 3.14), (3, 6.77), \dots\}$

The pointer initially has a zero value, thereafter positive values. We allow an error value e , analogous to the zero element of the stack semigroup, in case the program tries to take more things off the stack than are on it.

In state machine terms we describe the store-pointer machine by

$$\text{Input set: } I = \Omega \cup X$$

$$\text{State set: } Q = S \times (N \cup \{e\}) \text{ (that is, the set of all store and pointer pairs)}$$

Execution homomorphism: ϕ defined by

$$\begin{aligned} \phi \omega(s, n) &= (s', n') \\ &\text{where } s' = \text{assign}(n+1-k, \omega(s(n+1-k), \dots, s n))s \\ &\quad \text{or } s \text{ if either } n < k \text{ or } n = e \\ &\text{and } n' = n+1-k \text{ or } e \text{ if either } n < k \text{ or } n = e \\ &\text{where } k = \text{arity } \omega \end{aligned}$$

$$\phi x(s, n) = (s', n') \text{ where } s' = \text{assign}(n, \alpha x) \text{ and } n' = n+1$$

Simulating the stack machine by the store-pointer machine

Lemma. The stack machine can be simulated by the store-pointer machine, using as homomorphism θ the identity function and as function h

$$h(s, n) = s \ 1. \ s \ 2. \ \dots \ .s \ n \text{ or } \phi \text{ if } n = e$$

Proof. For simulation it is sufficient to show that $h \circ \phi i = \phi'(\theta i) \circ h$ where h , ϕ , ϕ' and θ have the appropriate values for the two machines in question.

First if $i = \omega \in \Omega$, put $k = \text{arity } \omega$. If $n < k$ then both expressions have value zero when applied to (s, n) . Otherwise

$$\begin{aligned} (h \circ \phi \omega)(s, n) &= \phi \omega(s \ 1. \ s \ 2. \ \dots \ .s \ n) \\ &= s \ 1. \ s \ 2. \ \dots \ .s(n-k). \ \omega(s(n-k+1), \dots, s \ n) \end{aligned}$$

$$\begin{aligned} (\phi'(\theta\omega) \circ h)(s,n) &= h(\text{assign}(n+1-k, \omega(s(n+1-k)), \dots, s n))s, \\ & \hspace{15em} n+1-k) \\ &= s\ 1.\ s\ 2.\ \dots.\ s(n-k).\ \omega(s(n-k+1), \dots, s n) \\ &= (h \cdot \phi\omega)(s,n) \end{aligned}$$

If $i = x \in X$

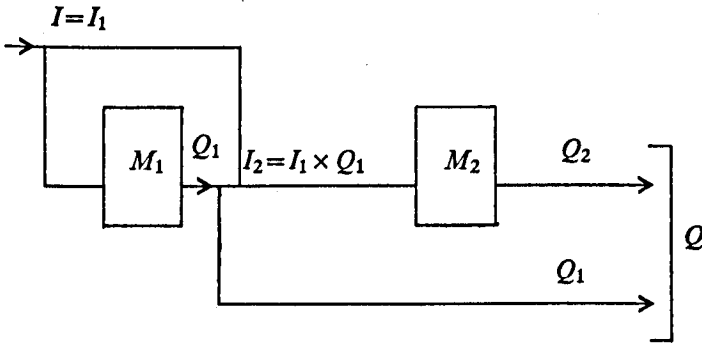
$$\begin{aligned} (h \circ \phi x)(s,n) &= \phi x(s\ 1.\ s\ 2.\ \dots.\ s\ n) \\ &= s\ 1.\ s\ 2.\ \dots.\ s\ n.\ \alpha\ x \\ (\phi'(\theta x) \circ h)(s,n) &= h(\text{assign}(n, \alpha x), n+1) \\ &= s\ 1.\ s\ 2.\ \dots.\ s\ n.\ \alpha\ x \\ &= (h \circ \phi x)(s,n) \end{aligned}$$

Cascade of two state machines

We shall now see how one machine can sometimes be replaced by a pair of interconnected machines, and thus develop a lemma about state machines which will enable us to effect the transition from a store-pointer machine to an address-program machine. The replacement can take place if the state q has two components q_1 and q_2 such that q_1 affects q_2 but q_2 does not affect q_1 (cf. the Krohn-Rhodes decomposition into a semidirect product, Arbib, 1968).

Consider machines M_1 and M_2 where $I_2 = I_1 \times Q_1$. We can connect them 'in cascade' to define a machine M as follows:

Put $I = I_1$, $Q = Q_2 \times Q_1$ and $\phi i(q_2, q_1) = (\phi_2(i, q_1)q_2, \phi_1 i q_1)$. Schematically:



Conversely, consider a machine M whose state set Q can be expressed as a direct product $Q_2 \times Q_1$ in such a way that there are a pair of functions f_2 and f_1 with $\phi i(q_2, q_1) = (f_2 i(q_2, q_1), f_1 i q_1)$. That is, the new value of the second component of the state depends only on the previous value of the second component and not on the previous value of the first component. Then we can always define homomorphisms ϕ_2 and ϕ_1 by

$$\phi_2(i, q_1)q_2 = f_2 i(q_2, q_1) \text{ and } \phi_1 = f_1.$$

This enables the machine M to be represented as a cascade of two simpler machines.

MATHEMATICAL FOUNDATIONS

Since ϕ is a homomorphism and we are given ϕi for each i we can evaluate $\phi t(q_2, q_1)$. However, if ϕ has the special property just mentioned we can obtain another expression for $\phi t(q_2, q_1)$ which enables us to separate the q_2 computation and the q_1 computation into two distinct phases, a 'translation' involving only q_1 followed by 'execution' involving only q_2 .

Lemma. Suppose that $\phi i(q_2, q_1) = (\phi_2(i, q_1) q_2, \phi_1 i q_1)$.

Put $I_2 = I_1 \times Q_1$ and define a 'translation' homomorphism τ

$$\tau : \sum I \rightarrow F_R((\sum I_2)^1 \times Q_1)$$

$$\tau i(t_2, q_1) = (t_2 \cdot (i, q_1), \phi_1 i q_1)$$

Then $\phi t(q_2, q_1) = (\phi_2 t_2 q_2, q_1)$ where $(t_2, q_1') = \tau t(1, q_1)$,

that is, we translate t into t_2 using q_1 and ϕ_1 , then execute t_2 on initial state q_2 using ϕ_2 .

Proof. First, if $t = i$, then

$$\tau i(1, q_1) = ((i, q_1), \phi_1 i q_1)$$

The right-hand side = $(\phi_2(i, q_1) q_2, \phi_1 i q_1)$

$$= \phi i(q_2, q_1) \text{ as required.}$$

To complete the induction we put

$$(t_2, q_1') = \tau t(1, q_1) \text{ and } (t_2^*, q_1'^*) = \tau(t \cdot i)(1, q_1)$$

From the definition of τ we get

$$t_2^* = t_2 \cdot (i, q_1') \text{ and } q_1'^* = \phi_1 i q_1'$$

Now we assume that $\phi t(q_2, q_1) = (\phi_2 t_2 q_2, q_1')$, then

$$\begin{aligned} \phi(t \cdot i)(q_2, q_1) &= \phi i(\phi t(q_2, q_1)) \\ &= \phi i(\phi_2 t_2 q_2, q_1') \\ &= (\phi_2(i, q_1')(\phi_2 t_2 q_2), \phi_1 i q_1') \\ &= (\phi_2(t_2 \cdot (i, q_1')) q_2, \phi_1 i q_1') \\ &= (\phi_2 t_2^* q_2, q_1'^*), \text{ as required.} \end{aligned}$$

Replacing the store-pointer machine by a cascade involving the address-program machine

We now show how the store-pointer machine (Table 1d) can be replaced by an address-program machine (Table 1e). This transition involves computing the current value of the pointer at compile-time and including it as the address of the next address instruction. We express this in state machine terms.

The execution homomorphism for the store-pointer machine has the form

$$\phi i(s, n) = (f_2 i(s, n), f_1 i n)$$

where $f_1 \omega n = n + 1 - k$, or e if $n < k$ or $n = e$, where $k = \text{arity } \omega$

$$f_1 x n = n + 1 \text{ or } e \text{ if } n = e$$

and $f_2 \omega(s, n) = \text{assign}(n + 1 - k, \omega(s(n + 1 - k), \dots, s n))s$ or s if either $n < k$ or $n = e$.

$$f_2 x(s, n) = \text{assign}(n + 1, \alpha x) \text{ or } s \text{ if } n = e.$$

This is the appropriate form for a cascade decomposition into M_1 and M_2 where

$$\begin{aligned}
 M_1 \text{ has } I_1 &= \Omega \cup X \\
 Q_1 &= N \cup \{e\}, \quad q_{01} = 0 \\
 \phi_1 &= f_1 \\
 M_2 \text{ has } I_2 &= (\Omega \cup X) \times (N \cup \{e\}) \\
 Q_2 &= S, \quad q_{02} = s_0 \quad \text{where } s_0 \text{ is an arbitrary store} \\
 \phi_2(i, n) &= f_2 i(s, n)
 \end{aligned}$$

Then we may define

$$\tau i(t_2, n) = (t_2 \cdot (i, n), f_1 i n)$$

We know from the lemma proved about cascades that $\phi t(s, n) = (\phi_2 t_2 s, n')$, where $(t_2, n') = \tau t(1, n)$, and 1 is the null address program.

Note that τ is a homomorphism which translates from a reversed Polish program t to an address program t_2 . It carries along as an extra result n' , the value of the pointer which would result from running that program. Applying $\phi_2 t_2$ to s corresponds to executing the address program starting with store s .

Using a conventional machine instead of the address-program machine

We defined a conventional machine (Table 1f) whose state comprises an accumulator as well as a store and a pointer, and whose instruction code has instructions like 'load contents of n into accumulator', 'add contents of n to accumulator', 'store accumulator in n '. The address-program machine is easily seen to be simulated by the conventional machine, using the expansion of one 'macro' instruction into several accumulator instructions as input homomorphism, and using the function which simply ignores the accumulator to examine the resulting states. This follows immediately from the corollary about simulation.

7. CONCLUSION

We have shown how instead of evaluating an expression by direct interpretation we can compile it, through various intermediate stages, into a program for a 'conventional' machine. We specify the method of execution for this machine and show how to use the resulting state to obtain the value of the expression, again through several intermediate stages (a rather trivial process). Each stage involves 'representing' an algebra in some other algebra. The correctness of our procedure follows from the correctness of each stage. For reference we have collected the various stages and written out the whole compiler in the Appendix.

The proof involves some general lemmas about algebra and algebraic machine theory. These are proved by appeal to standard lemmas of algebra or by easy induction. We then apply them to the compilation situation with

MATHEMATICAL FOUNDATIONS

no more ado than some substitutions to verify that the computing machines and the transitions between them satisfy the conditions specified in the lemmas.

This exercise is offered as an example of what we hope will be a wider range of applications of algebraic techniques to programming problems.

Acknowledgements

This work was supported by Queen Mary College, University of London, by the University of Edinburgh and by Radio Corporation of America Laboratories, Princeton (under AFOSR contract No. F44620-68-c-0012).

One of the authors (R. M. B.) would like to thank R.C.A. for enabling him to do a large part of this work during a summer visit, and Dr S. Amarel and members of the Computer Research Group at R.C.A. for stimulating discussions. Thanks are also due to the Science Research Council for meeting travel expenses in attending the Workshop, and to Miss Lenore Bonofiglio and Miss Eleanor Kerse for typing.

APPENDIX: SUMMARY OF THE COMPILER

It may be helpful to collect the whole compiler on to a single page. The instruction sets are

$I_1 = \Omega \cup X$ (reversed Polish instructions)

$I_2 = (\Omega \cup X) \times (N \cup \{e\})$ (address-instructions, 'macros')

$I_3 = (\Omega' \cup X) \times (N \cup \{e\})$ (conventional accumulator instructions)

The algebras needed are

$A_1 = W_\Omega(X)$, $A_2 = E(S_W, X, f_W)$, $A_3 = \sum I_1$, $A_4 = F_R((\sum I_2)^1 \times N)$,

$A_5 = \sum I_2$, $A_6 = \sum I_3$

where $f_W \omega = \omega$, $f_W x = x$.

Define a translation function from expressions to transformations of address-programs and pointers.

$trans = [Extend(A_1, A_2)f_W] \circ h \circ [Extend(A_3, A_4)\tau]$

where h is the identity function from the carrier of A_2 into that of A_3 .

and $\tau i(t_2, n) = (t_2 \cdot (i, n), f_1 i n)$

where $f_1 \omega n = n + 1 - k$, or e if $n < k$ or $n = e$, where $k = \text{arity } \omega$

and $f_1 x n = n + 1$ or e if $n = e$

Obtain an address program t_2 by applying this to the given expression w

$(t_2, n') = trans w (1, 0)$

Translate this to a conventional accumulator program t_3 :

$t_3 = Extend(A_5, A_6) g t_2$

where $g(i, n)$ is the expansion of the macro (i, n) to a sequence of conventional instructions.

Execute the program t_3 on an arbitrary store s_0 , using ϕ the execution homomorphism of the conventional machine to get a final store s' .

$s' = \phi t_3 s_0$

Extract the result (making the obvious simplification of the chain of trivial functions which extracts the value v)

$$v = s'n'$$

REFERENCES

- Arbib, M.A., ed. (1968) *Algebraic Theory of Machine Languages and Semigroups*. New York and London: Academic Press (see especially chapters 1, 3 and 5).
- Burstall, R.M. & Popplestone R.J. (1968) POP-2 reference manual. *Machine Intelligence 2*, pp. 207-46 (eds Dale, E. & Michie, D.). Edinburgh: Oliver & Boyd. Also in *Pop-2 Papers* Edinburgh: Edinburgh University Press.
- Burstall, R. M. (1969) Proving properties of programs by structural induction. *Comp. J.* **12**, 1, 41-8.
- Cohn, P.M. (1965) *Universal Algebra*. New York and London: Harper Row.
- Cooper, D.C. (1969) Program scheme equivalences and second-order logic. *Machine Intelligence 4*, pp. 3-15 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.
- Floyd, R.W. (1966) Assigning meanings to programs. *Mathematical Aspects of Computer Science*, pp. 19-32. Providence, Rhode Island: American Mathematical Society.
- Kaplan, D.M. (1967) Correctness of a compiler for Algol-like programs. *Stanford Artificial Intelligence Memo. No. 48*. Department of Computer Science, Stanford University.
- Landin, P.J. (1966) The next 700 programming languages. *Comm. Ass. Comp. Mach.*, **9**, 3, 157-66.
- McCarthy, J. & Painter, J. A. (1967) Correctness of a compiler for arithmetic expressions. *Mathematical Aspects of Computer Science*. Amer. Math. Soc. Providence, Rhode Island, pp. 33-41.
- Painter, J. A. (1967) Semantic correctness of a compiler for an Algol-like language. *Stanford Artificial Intelligence Memo. No. 44*, Department of Computer Science, Stanford University.



Towards the Unique Decomposition of Graphs

C. R. Snow and
H. I. Scoins

Computing Laboratory
University of Newcastle-upon-Tyne

INTRODUCTION

This paper describes part of a research project which attempts to define an indexing system for linear graphs. Such an indexing system ideally would eliminate the necessity of examining large numbers of permutations of the labels of a graph to find a canonical form for any given graph. In practice it may be permissible to look at a small subset of the set of all permutations, provided this subset is not too large. What is meant by 'too large' is not defined, but presumably this is dependent on the comparative effort involved in reducing the number of permutations and in looking at all permutations.

An indexing system for graphs implies that an ordering can be imposed upon graphs so that if the index of any particular graph G 'is less than' or 'comes before' that of another graph G' , then the statement $G < G'$ is meaningful.

The graphs under consideration may be defined in terms of a vertex set V , together with a set Γ of pairs (a,b) where $a, b \in V$. The possibility of a pair (a,a) (which represents a simple loop) is excluded, and only one pair can consist of the elements a and b in either order, so that only one undirected line may join a pair of nodes.

An attempt is being made to split any graph uniquely into two subgraphs in such a way that if G can be split into g_1 and g_2 , and G' can be split into g'_1 and g'_2 , then an ordering may be defined recursively by saying that $G < G'$ if $g_1 < g'_1$ or ($g_1 = g'_1$ and $g_2 < g'_2$). We also require that $g_1 < G$ and $g_2 < G$.

This definition is not complete without a starting point for the recursion. This can simply be the relation $G_0 \leq G$ for all graphs G where G_0 is the trivial graph with one node and no lines.

Consideration has been given primarily to connected graphs, since in a general graph the connected components may be considered as separate entities.

Scoins (1968) has described a system of indexing trees of various types, and because we have gained some facility in the manipulation of trees, we have decided that g_1 is to be a tree, and in particular a spanning tree of G . A spanning tree of a graph is any partial graph of G which is a tree, and for which the vertex set is the same as the vertex set for G .

INDEXING TREES

The ordering of trees described by Scoins (1968) depends on the height representation of the tree. Ordered rooted trees may be described in terms of a sequence of small integers, provided the order in which the nodes are numbered is canonical in the following sense:

Number the root first and then proceed up the tree taking the leftmost branch whenever possible, labelling each node passed. Having reached the top of a branch, move across to the positive neighbour of the topmost node, number that node, and again continue up the tree and to the left. If at any stage a node has neither an 'above left node' (not already labelled) nor a positive neighbour, then move to the 'below' of that node and try again to move to the positive neighbour. When the root is reached again, the process terminates, as all of the tree has then been covered. The tree being thus labelled, it may be represented by a vector $\mathbf{h}=(h_1, h_2, \dots, h_n)$, where the value of h_i is equal to the distance of the node i from the root of the tree.

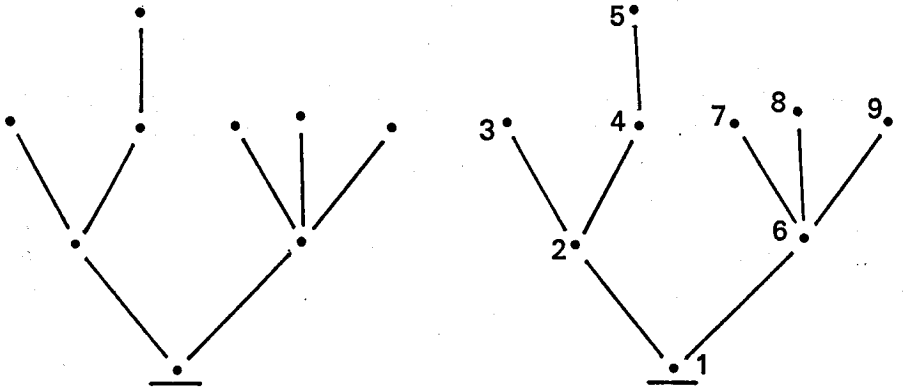


Figure 1

Figure 1 illustrates this method of labelling uniquely an ordered rooted tree, the example having nine nodes and height vector

$$\mathbf{h}=(0 \ 1 \ 2 \ 2 \ 3 \ 1 \ 2 \ 2 \ 2).$$

The action of drawing the tree on paper imposes an ordering on the tree (even if it had none before), and for the sequence to represent an unordered rooted tree, certain restrictions must be placed on the sequence. A canonical form for rooted trees is defined as follows. If a tree T consists of a number of

subtrees T_1, T_2, \dots, T_k all planted at the root of T as shown in figure 2, then T is in canonical form if $T_1 \geq T_2 \geq \dots \geq T_k$. T_1, \dots, T_k are themselves assumed to be in canonical form. In the vector \mathbf{h} , any substring of integers starting with a '1' and going up to but not including the next '1' represents one

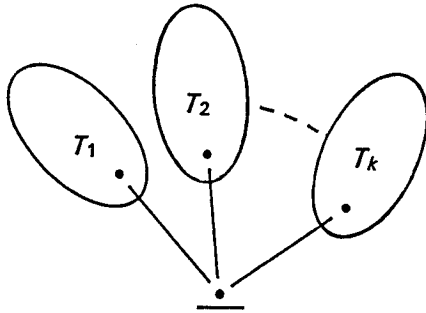


Figure 2.T

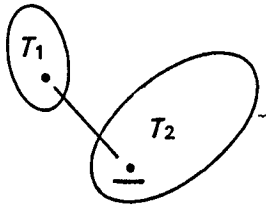


Figure 3

of these planted subtrees. Thus the comparison of subsequences is very straightforward. Now, given two trees T and T' represented by $\mathbf{h}=(h_1, \dots, h_n)$ and $\mathbf{h}'=(h'_1, \dots, h'_n)$, the ordering $T < T'$ can be defined by

$$T < T' \Leftrightarrow h_i < h'_i \quad \text{for some } i,$$

$$\text{and } h_j = h'_j \quad j=1, \dots, i-1.$$

In previous work, a free tree has been made into a rooted tree by forcing a root on to the tree at its centre, if the tree has a centre, or at one end of the bi-centre. This operation is effected unambiguously by positing that if a tree T which has a bi-centre may be drawn as shown in figure 3, then $T_1 \leq T_2$, where T_1 and T_2 are rooted trees in canonical form. Using these rules, any free tree may be transformed uniquely into an ordered rooted tree.

THE CO-TREE

In decomposition of a graph G , the two 'parts' into which the graph is subdivided are both partial graphs over the vertex set of G . The sets of lines of the two parts are disjoint and their union is the set of lines of G .

The decomposition process must terminate since an index will be given to a graph by virtue of the spanning tree and the index of the co-tree. This co-tree

MATHEMATICAL FOUNDATIONS

will in general be a disconnected graph, and some of its components may be isolated points. The co-tree will also have fewer lines than the original graph.

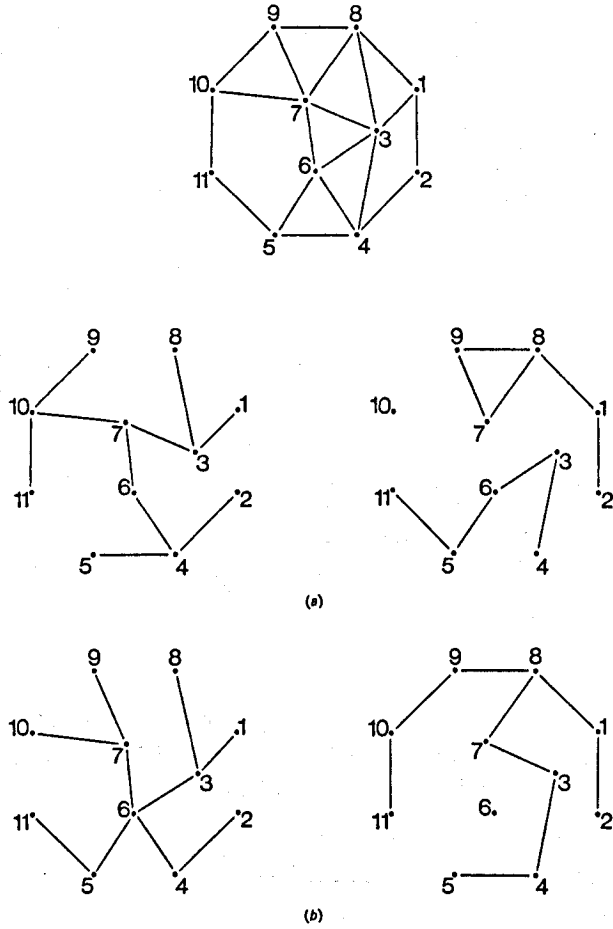


Figure 4

In the comparison of disconnected graphs, we adopt the convention that the 'largest' component is to be inspected first; since the graph G_0 is the earliest graph, all those components of a graph which are isolated points are considered after all other components. In other words, given two disconnected graphs G and G' whose connected components are C_1, \dots, C_k and $C'_1, \dots, C'_{k'}$, respectively, we have

$$\begin{aligned}
 G < G' &\Leftrightarrow C_i < C'_i \text{ for some } i \leq \min(k, k') \\
 \text{and } C_j &= C'_j \text{ for } j = 1, \dots, i-1, \\
 \text{or } C_j &= C'_j \text{ for } j = 1, \dots, \min(k, k') \\
 \text{and } k &< k'.
 \end{aligned}$$

There is, however, a difficulty as regards the largest component of a graph; 'largest' should refer to the index of the component, but to find the largest index all components must be decomposed to find their indices, so that extremely complex recursions may be required. Assuming that this difficulty can be overcome, we may confine our attention to connected graphs.

Figure 4 gives an example of the splitting process. It shows two methods of decomposing the same graph using two different spanning trees. In case (a) the co-tree consists of three components, one of which is a tree, one a graph, and one an isolated point; and in case (b) the co-tree has two components, one a tree, and one the graph G_0 .

CENTRE OF GRAPH

A major problem in carrying out a decomposition process is the question of uniqueness. Some method of determining a single unique point in a graph would be a start to any method for finding a unique spanning tree. From it we might hope to grow a unique tree. Since spanning trees of a general graph are essentially free, a root must be forced on to such a tree, as has been mentioned already. We would like to put the root at the centre (or one end of the bi-centre) of the spanning tree.

The ideal definition of the centre of a graph might therefore be: the centre of a graph is the centre of an 'optimal' spanning tree, that is, a unique spanning tree is defined in some way and has its centre or root at the centre of the graph.

The tree which is to span the graph will have a centre at the mid-point of one of the max-min paths of the graph. A max-min path is the maximum of the shortest paths between any two nodes of the graph. However, there are in general a number of max-min paths, and the set of mid-points contains more than one point. Also, difficulties may arise when the length of the max-min path is odd.

A rather better definition of the centre of a graph G is given by Sabidussi (1966):

If $d(x,y)$ is the shortest distance between the points x and y in a graph, and V is the vertex set of G , then define $s(x) = \max_{y \in V} \{d(x,y)\}$. Now define the centres of the graph to be all those nodes c for which

$$s(c) \leq s(x) \quad x \in V.$$

The set $\{c\}$ is included in the set of mid-points of max-min paths, but in general has less elements, as the reader can see by examining the example of figure 4. The problem of finding a unique point is still not solved.

One method of deciding which member of the set $\mathcal{C} = \{c\}$ is to be the unique point is to look at the number of its first neighbours, second neighbours, etc. For each $c \in \mathcal{C}$, define the p -vector ($p = s(c)$) $v_c = (v_1, v_2, \dots, v_p)$ where v_i is the number of points x for which $d(x,c) = i$. Clearly $\sum_{i=1}^p v_i = n - 1$,

where n is the number of points in the graph. Then a point K is centre of the graph if $v_K < v_c$ for all $c \in \mathcal{C}$ (by $v < v_j$ we mean $v_i < v'_i$ for some $i \leq p$ and $v_j = v'_j$ for $j = 1, \dots, i-1$). Even this does not necessarily yield a unique point.

The graph in figure 4 has a unique centre, if one considers only the values of $s(x)$. $s(6) = 2$, $s(i) \geq 3$ for all $i \neq 6$. The graph in figure 5 has two centres, even though the number of candidates for centre has been reduced by considering the vectors v . $s(x) = 2$ for $x = 2, 3, 4$; $s(x) \geq 3$ for $x = 1, 5, 6, \dots, 10$. $v_2 = (5, 4)$, $v_3 = (4, 5)$ and $v_4 = (5, 4)$. Thus we are unable by this method to differentiate between node 2 and node 4.

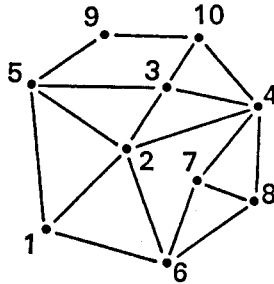


Figure 5

The method just described is similar to the first step in an algorithm due to Read (1966) for the partial classification of the nodes of a graph. This algorithm begins by placing the nodes in equivalence classes according to some rule, such as the degree of each node (the degree of a node is the number of lines that are incident to that node), and then an iteration cycle is carried out, refining the classification of each node, so that the number of equivalence classes is increased and the number of nodes in each class is decreased. The algorithm is terminated when either the number of equivalence classes is the same as the number of nodes in the graph, that is, each class consists of just one node, or when one cycle of the iteration fails to produce any new classes.

The actual refinement is carried out by inspecting the class of each node together with the classes of all its first neighbours. Thus after k iterations, the class of each node depends on the class of all nodes of the graph within a distance of k of that node. Since at each stage the refinement depends on the class of each node, the number of classes cannot decrease, and so one of the stopping criteria must be reached. We then expect the algorithm to finish after at most n iterations, where n is the length of the max-min path, but in practice only about half this number of iterations is required.

A version of this algorithm has been programmed for the KDF9 computer in ALGOL and appears to work satisfactorily.

The initial classification of the nodes is at our disposal, and if the nodes are classified according to their values $s(x)$ a more satisfactory method of finding the centre of a graph seems likely to develop, but no definite conclusions have yet been reached.

This method may or may not produce a unique point, which we may call the centre of the graph, but since all our previous methods for looking for a centre depended in some way on a distance property, as also does the indexing of trees, it would seem disadvantageous to begin to think in terms of an algorithm which uses the degree of each node.

The best way of deciding on the centre of a graph is perhaps to use a mixture of these two methods, that is to form the initial set of centres by the values of the $s(x)$, reduce the size of this set by using the vector v , and finally, using Read's algorithm, to decide which member of the reduced set is to be the centre of the graph. If after all these steps there is still a choice, then spanning trees must be grown from each of the candidate points.

Thus, returning to the example in figure 5, the algorithm of Read gives after 2 refinements:

node	1	2	3	4	5	6	7	8	9	10
class	4	9	7	8	6	5	3	3	1	2

The fact that nodes 7 and 8 are still in the same equivalence class is a reflection of the obvious symmetry of the situation.

The distances $s(2)$ and $s(4)$ were the same, and the vectors v_2 and v_4 also were both (5,4), but the algorithm has given them different classifications, and so we are now able to distinguish between them.

THE SPANNING TREE

In the construction of the spanning tree itself, grown from the centre of the graph, the object has been to construct the 'smallest' spanning tree possible. By 'smallest' tree is meant the earliest tree in the sequence generated by the height representation as described in an earlier section.

There are clearly a number of trees which could have been chosen as optimal, but this use of the height representation seems to be the most convenient method.

There is at least one method of counting the spanning trees of a graph which can be developed into an algorithm for generating all these trees (Percival 1950), so one could simply generate them all, give them all roots, and then pick out the earliest; but this would involve inspecting an impracticably large number of trees. The method of choosing the centre of the graph was designed with the idea of finding the optimal tree by a 'growing' technique.

Obruca (1964) published an algorithm for finding a minimal spanning tree. This algorithm, however, was concerned with spanning a weighted graph, in which each line had a cost or weight associated with it. If some method can be found for using the structure of the graph to impose a 'cost' on each line, the Obruca algorithm automatically finds the spanning tree we need. Later, Obruca (1966) gave a proof that this tree is unique provided the costs on the graph are distinct.

MATHEMATICAL FOUNDATIONS

Obruca (1966) also attempted to find the most efficient method of storing a graph, both in terms of actual computer storage used, and in terms of the work involved in representing a graph in the most convenient form for any particular operation which might be performed on that graph. In the present work, no such attempt has been made; the graph has normally been represented in terms of an $n \times n$ binary matrix $H=(h_{ij})$, called the adjacency matrix (where the graph has n points), such that $h_{ij}=1$ if node i is adjacent to node j and 0 otherwise. From this, by use of a rather primitive shortest distance algorithm, a matrix SD (with typical element d_{ij}) can be constructed, where d_{ij} =length of the shortest path from node i to node j . Since the graphs are all undirected and have no loops, both the matrices H and SD are symmetric with zero diagonal elements.

The i th row (or column) of the matrix SD is a vector r_i which holds the distance of each node from the node i (we may notice here that $s(i)$ =largest element of r_i). In particular we have a vector r_c for the centre, c , of the graph. Since the height representation of a tree gives the distance of each node from the root, it merely remains to re-order the elements of the vector r_c to generate the height representation for some spanning tree.

As an example, consider the graph in figure 4. With the labelling on the graph we have

$$SD = \begin{pmatrix} 0 & 1 & 1 & 2 & 3 & 2 & 2 & 1 & 2 & 3 & 4 \\ 1 & 0 & 2 & 1 & 2 & 2 & 3 & 2 & 3 & 4 & 3 \\ 1 & 2 & 0 & 1 & 2 & 1 & 1 & 1 & 2 & 2 & 3 \\ 2 & 1 & 1 & 0 & 1 & 1 & 2 & 2 & 3 & 3 & 2 \\ 3 & 2 & 2 & 1 & 0 & 1 & 2 & 3 & 3 & 2 & 1 \\ 2 & 2 & 1 & 1 & 1 & 0 & 1 & 2 & 2 & 2 & 2 \\ 2 & 3 & 1 & 2 & 2 & 1 & 0 & 1 & 1 & 1 & 2 \\ 1 & 2 & 1 & 2 & 3 & 2 & 1 & 0 & 1 & 2 & 3 \\ 2 & 3 & 2 & 3 & 3 & 2 & 1 & 1 & 0 & 1 & 2 \\ 3 & 4 & 2 & 3 & 2 & 2 & 1 & 2 & 1 & 0 & 1 \\ 4 & 3 & 3 & 2 & 1 & 2 & 2 & 3 & 2 & 1 & 0 \end{pmatrix}$$

Now if we take the node 6 to be the centre, the r_c vector is

$$2 \quad 2 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2$$

This may be arranged in a number of different orders to form height sequences for spanning trees, e.g.

$$0 \quad 1 \quad 2 \quad 2 \quad 1 \quad 2 \quad 2 \quad 1 \quad 2 \quad 1 \quad 2$$

with permutation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 6 & 3 & 1 & 8 & 7 & 9 & 10 & 4 & 2 & 5 & 11 \end{pmatrix}$$

corresponds to the tree T_1 in figure 6.

Another possible arrangement is

0 1 2 2 2 1 2 1 2 1 2

with permutation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 6 & 7 & 8 & 9 & 10 & 3 & 1 & 4 & 2 & 5 & 11 \end{pmatrix}$$

corresponding to the tree T_2 in figure 7.

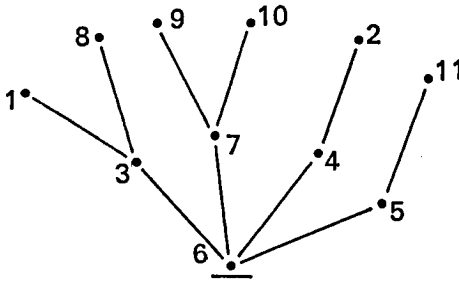


Figure 6.T₁

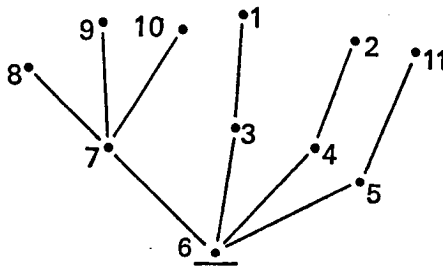


Figure 7.T₂

Of these two examples of spanning trees, T_1 is nearer to the optimum tree since $T_1 < T_2$.

Since the lexicographical ordering of height sequences generates the tree of height 1 first, then the trees of height 2 and so on, one would expect that keeping the numbers in the height sequence as small as possible would help to produce the least spanning tree. Thus one attempts to construct a spanning tree in which the distance from any node to the root is as small as possible, that is, it is the same as the shortest distance between these two nodes in the graph itself. Thus, for each node i ,

$$h(i) = d(i, \text{root}).$$

Spanning trees with this property are called *mushrooming* trees by Obruca (1966).

METHODS OF OBTAINING THE LEAST SPANNING TREE

Two methods of finding the optimal spanning tree are currently being investigated. The first method takes any spanning tree, but preferably a mushrooming tree, and operates on it, trying to improve the tree, that is, iteratively looking for a tree with an earlier height sequence. The second method attempts to find the best tree directly.

A partial specification of an optimal spanning tree is given by the r_c vector obtained from the SD matrix; however, the problem is to convert this into the correct form of the optimal spanning tree. One can construct the least vector at one's disposal by re-ordering the r vector, and thus form a 'target' at which to aim when forming the optimum height sequence.

The vector

2 2 1 1 1 0 1 2 2 2 2

has 1 zero, 4 ones and 6 twos, and thus no sequence can be obtained in canonical form which is better than

0 1 2 2 1 2 2 1 2 1 2.

It is, however, an entirely different matter to form an actual spanning tree which fits this vector. As it turns out, one such tree has been found (see figure 6), but this is by no means always feasible.

The improvement algorithm takes any spanning tree as a starting value. It then tries to make a better spanning tree by considering each line in the co-tree as a candidate for insertion into the tree. Suppose the starting tree were the tree shown in figure 7. Then attempting to place the line (3,8) in the tree would yield an improved tree (figure 6). When a line is placed in the spanning tree, another line has to be removed to retain the tree structure. In the example given, the line removed is the line (1,8), and a number of short-cuts have been devised to prove that this line is the only line which may be removed to maintain the mushrooming property of the spanning tree.

One can show without difficulty that this method will converge, but we are unable to say definitely that the method converges to the optimal tree. In fact, one of the greatest problems in this work is the inability to recognize the optimum when one finds it. The other major difficulty is that of generating all the optimum trees. It is likely that there is more than one optimum tree, in which case one must look at the co-trees to decide between decompositions which give the same tree.

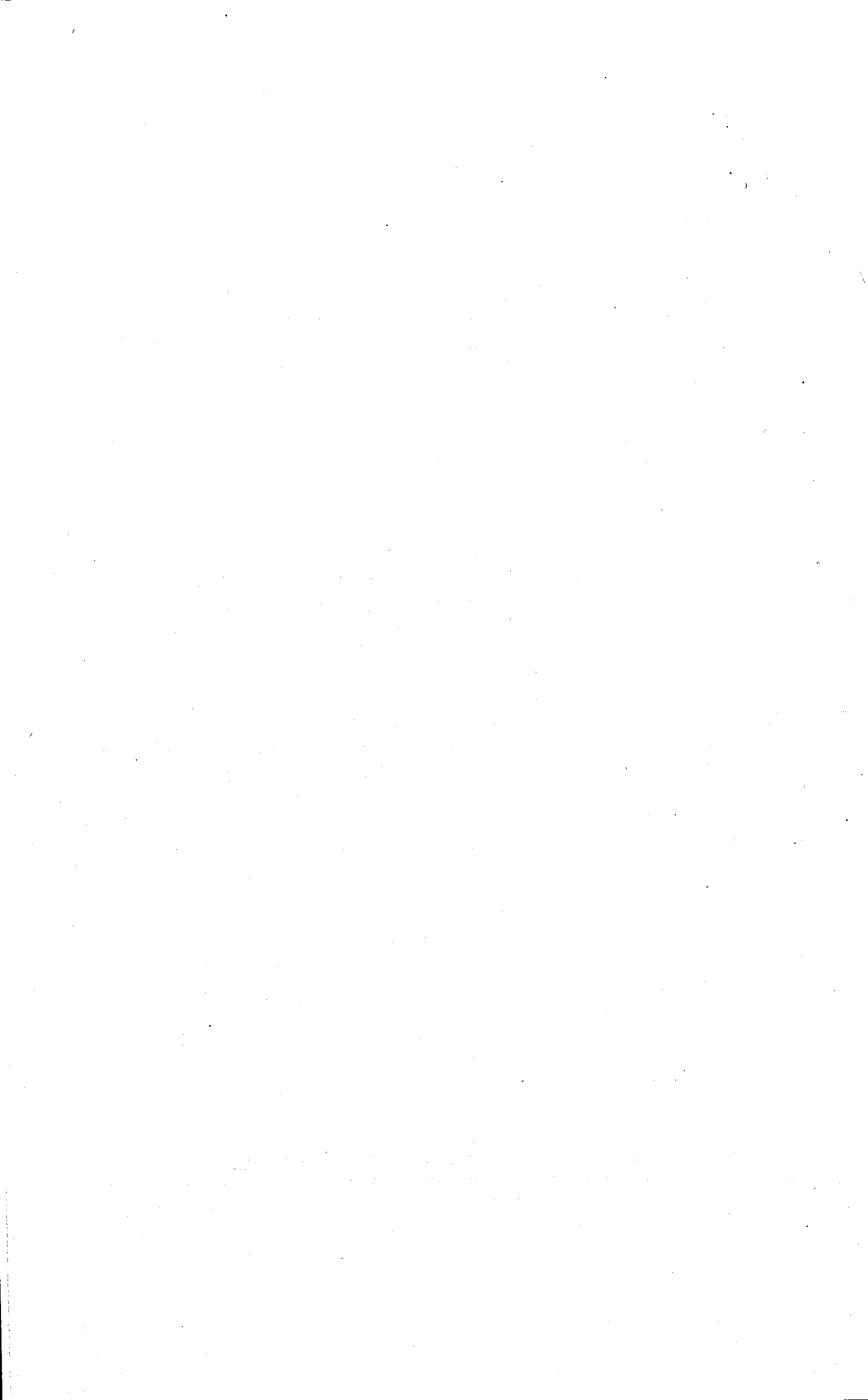
Acknowledgement

This work is supported by a grant from the Science Research Council.

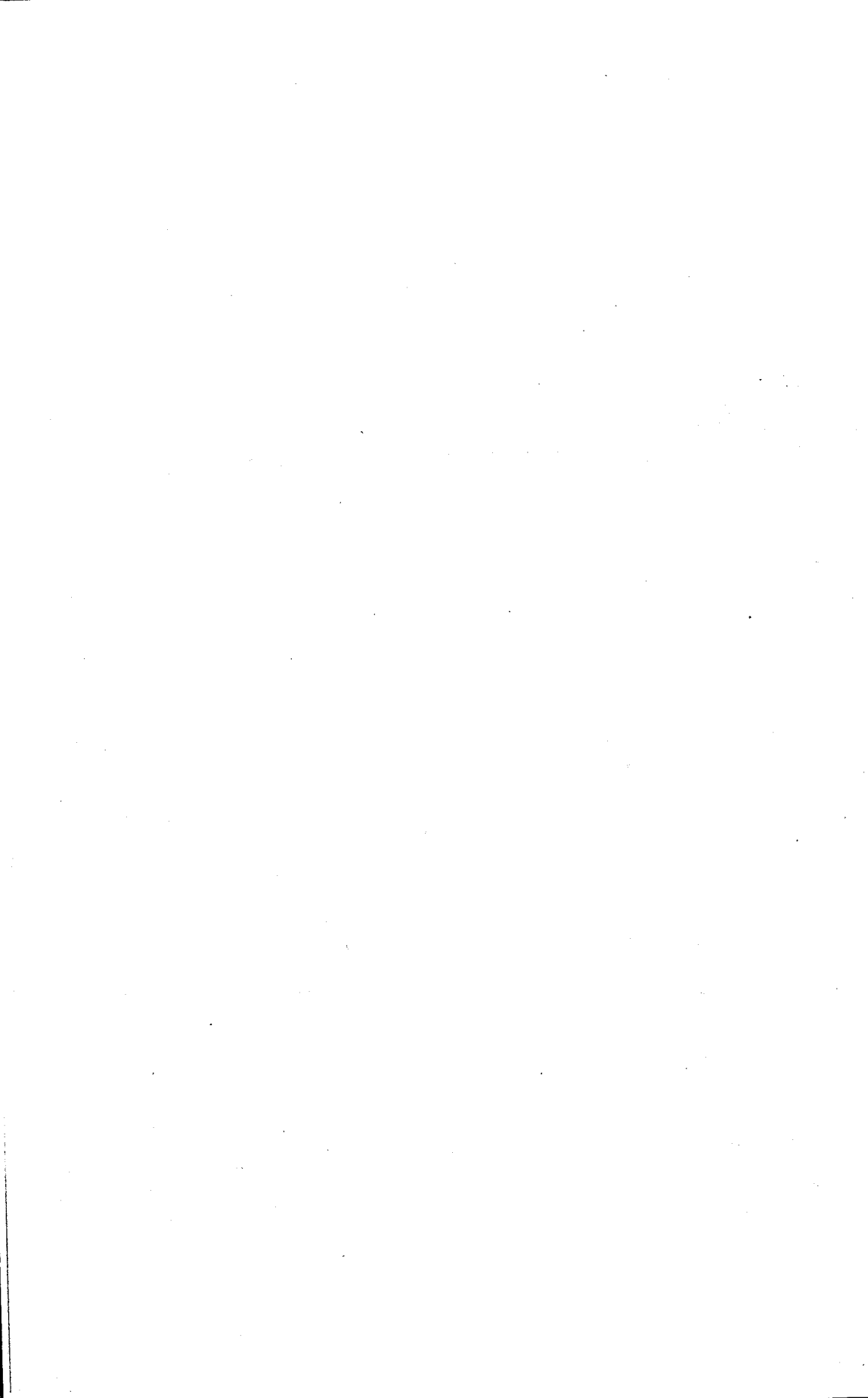
REFERENCES

Obruca, A.K. (1964) Algorithm 1. MINTREE. *Comp. Bul.* 8, 67.
 Obruca, A.K. (1966) The Manipulation of Linear Graphs inside a Computer and some Applications. Ph.D. thesis. University of Newcastle upon Tyne.

- Percival, W.S. (1950) Solution of Passive Networks by means of Mathematical Trees. *Proc. IEE* 100c, 143-50.
- Read, R. C. and Parris R. (1966) Graph Isomorphism and the Coding of Graphs. *UWI/CC3 Research Report*. University of West Indies.
- Sabidussi, G. (1966) The Centrality Index of a Graph. *Theory of Graphs International Symposium*. Rome. pp. 369-72. Dunod, Gordon and Breach.
- Scoins, H.I. (1968) Placing trees in Lexicographic Order. *Machine Intelligence 3*, pp. 43-60 (ed. Michie, D.). Edinburgh: Edinburgh University Press.



THEOREM PROVING



Advances and Problems in Mechanical Proof Procedures

D. Prawitz

University of Stockholm

Abstract

In this paper, I first give a simplified description of the method proposed in my earlier paper 'An improved proof procedure'. This method is then further developed in order to diminish the work of finding a substitution that makes a given formula inconsistent. Finally, the possibility of developing the method in another direction is discussed.

By a proof procedure (or proof method) I understand in this connection an algorithm \mathcal{A} with the property: for any valid formula F in the predicate calculus, $\mathcal{A}(F)$ (i.e., the result of applying \mathcal{A} to F) is a proof of F . Equivalently, one may consider algorithms \mathcal{A} with the property: if F is not satisfiable, then $\mathcal{A}(F)$ is a refutation of F . Following a tradition, I shall use this latter formulation. (It is of course immaterial which formulation one chooses, and when it comes to programming the procedure, all differences disappear completely.)

1. THE PRIMITIVE METHOD

Introduction

To have a convenient reference for the following discussion, I start by giving a description of the original method developed by Skolem (1928) and the main theorem on which this method is based. The necessary logical apparatus can be kept remarkably simple.

We use a formulation of predicate logic containing individual constants and function symbols. To simplify the description of the method, it is convenient to restrict the formulae F to which the method is applicable. Firstly, it is supposed that F is closed and in prenex normal form. Secondly, it is supposed that all existential quantifiers are eliminated. To see how this can

The main part of this paper was also presented in lectures at the University of Stockholm and the Technische Hochschule of Hanover in the spring of 1967.

THEOREM PROVING

be done, consider the formula $\forall x\exists yF(x,y)$. If this formula is true in some model, we may for each value of x choose a particular value of y so that these values satisfy the formula $F(x,y)$; in other words, there exists a so-called Skolem-function that satisfies the formula $\forall xF(x,f(x))$ in the given model. It follows that if $\forall x\exists yF(x,y)$ is satisfiable, then so is $\forall xF(x,f(x))$; the converse of this is, of course, also true.

Given a closed formula F in prenex normal form, I shall say that F^* is a *Skolem-transformation* of F (and that F^* is *Skolem-transformed*), if F^* is obtained from F by eliminating every existential quantifier, and by replacing the corresponding variable in the formula by a term $f^n(x_1, x_2, \dots, x_n)$, where x_1, x_2, \dots, x_n are exactly the variables that are quantified by the universal quantifiers which precede the eliminated existential quantifier in question; for different existential quantifiers, different function symbols not occurring in F are to be chosen. A function symbol with zero arguments is the same as an individual constant and will sometimes be denoted by the letter a (a_0 is to denote a particular individual constant). By the same reasoning as in the example above, it is seen that F is satisfiable if and only if the Skolem-transform F^* is also satisfiable.

Further example. Given a formula $\exists x\forall y\exists z\forall v\exists wF(x,y,z,v,w)$, where $F(x,y,z,v,w)$ is a formula without quantifiers, we replace it by a Skolem-transform $\forall y\forall vF(a,y,f(y),v,g(y,v))$.

Let F be a Skolem-transformed formula. We define the so-called *Herbrand-universe* for F , denoted H_F , as the set of individual terms that can be built up using the individual constants and function symbols of F ; in case there is no individual constant in F , we may use the constant a_0 . In other words, H_F is recursively defined by:

1. All individual constants in F belong to H_F ; if there is no such constant, a_0 belongs to H_F .
2. If t_1, t_2, \dots, t_n belong to H_F and f^n occurs in F , then $f^n(t_1, t_2, \dots, t_n)$ belongs to H_F .

Main theorem

A formula F of the form $\forall x_1\forall x_2\dots\forall x_nM(x_1, x_2, \dots, x_n)$ where $M(x_1, x_2, \dots, x_n)$ contains no quantifier, is satisfiable if and only if every finite subset of the set

$$\{M(t_1, t_2, \dots, t_n): t_i \in H_F, \text{ for every } i \leq n\} \quad (1)$$

is satisfiable.

Proof. Since $\forall x_1\forall x_2\dots\forall x_nM(x_1, x_2, \dots, x_n)$ logically implies $M(t_1, t_2, \dots, t_n)$ one half of the theorem is trivial. The proof of the other half is conveniently divided into two parts, proving

1. If (1) is satisfiable then so is F .
2. If K is a set of quantifier-free formulae and every finite subset of K is satisfiable, then so is K .

The proof of 1 is easy: given a model of the set (1) we eliminate every individual from the domain of the model that is not a value of a term in H_F . The substructure obtained in this way is a model of F .

The proof of 2 is slightly more complex and seems first to have been given by Skolem (1929). In summary, the argument is: let $K = \{F_1, F_2, \dots\}$ be an infinite set of quantifier-free formulae, and let Γ_n be the set of the truth-value assignments to the atomic formulae occurring in F_1, F_2, \dots , or F_n , that satisfy the set $\{F_1, F_2, \dots, F_n\}$. The following facts are easily seen to be true. (i) Γ_n is not empty (by assumption in 2). (ii) Γ_n is finite. (iii) Each member of Γ_{n+1} is an extension of some member of Γ_n . (i)–(iii) imply that there exists an infinite sequence T_1, T_2, \dots such that $T_n \in \Gamma_n$ and T_{n+1} is an extension of T_n (the argument is the same as in König's Unendlichkeitslemma: an infinite tree where each branching is finite contains an infinite branch). The union of all the assignments in this sequence is a truth-value assignment that satisfies K .

The method

Since one can decide the question whether a finite set of quantifier-free lines is satisfiable or, in other words, consistent, the theorem immediately gives rise to a method of the kind sought: searching through the finite subsets of (1) and testing for consistency, one will sooner or later find an inconsistent set if F is not satisfiable; this inconsistent set can be taken as the refutation of F .

Instead of considering all the subsets of (1), one may fix some order M_1, M_2, \dots among the formulae in (1), henceforth called the *instances* of F , and then restrict attention to the sets $M^m = \{M_1, M_2, \dots, M_m\}$. This is sufficient, since if there is an inconsistent subset of (1), then this subset is contained in some M^m , and, hence, M^m is also inconsistent. The algorithm for showing that F is not satisfiable is then:

For successively larger m , form the set $\{M_1, M_2, \dots, M_m\}$ and test for consistency, until an inconsistent set is found.

This method was first stated by Skolem (1928) and is the core of all known methods that do not simply enumerate all proofs. (Many later methods differ in no essential respect from Skolem's.) The first attempts to program Skolem's method for computers were made independently by Gilmore (1959 and 1960), Prawitz, Prawitz, and Voghera (1959 and 1960), and Wang (1960). These three programs are all rather similar, one difference being that the formulae are not required to be in prenex normal form in the programs proposed by Prawitz *et al.*, and by Wang (cf. the remark in section 4 below).

Problematic features in Skolem's method

There are two main problems involved in Skolem's method. The first concerns the question of deciding whether a quantifier-free formula is consistent. This

THEOREM PROVING

may of course be done by considering all the 2^n different assignment of truth-values to the n atomic formulae in the formula. A less tedious method is to develop the formula into disjunctive normal form and see whether every clause contains a contradiction. This method is the one used in all the three programs mentioned above. (Wang uses the technique of simplifying Gentzen-sequents – developed especially by Kanger (1957) – which essentially only accomplishes the task of transforming a formula to conjunctive normal form. Prawitz, Prawitz, and Voghera use the technique of semantic tableaux developed in Beth (1955), which accomplishes the same task but gives a more economical, tree-formed presentation of the required normal form.) However, this method is still very laborious. If one obtains k conjunctive clauses when transforming M_1 to disjunctive normal form (in the usual way by multiplication), one will obtain k^m clauses when similarly transforming $M_1 \& M_2 \& \dots \& M_m$.

The second problem and the main weakness of Skolem's method may be illustrated in the following way. Suppose that F is inconsistent and let M_1, M_2, \dots be some ordering of the instances of F . Then the least inconsistent set of instances is probably much smaller than the least inconsistent set among the sets $\{M_1, M_2, \dots, M_m\}$. The set $\{M_5, M_{61}, M_{100}\}$ may, for example, be inconsistent, but the first inconsistent set among the sets $\{M_1, M_2, \dots, M_m\}$ may well be $\{M_1, M_2, \dots, M_{100}\}$. While it would have been an easy task to refute F if the instances had been generated in the order M_5, M_{61}, M_{100} a machine might be exhausted by trying to decide that $M_1 \& M_2 \& \dots \& M_{100}$ is inconsistent.

The problem thus concerns the order in which the instances of the formula F are generated. One would like to have a method for generating these instances in such an order that a *minimal* inconsistent set of instances is found (i.e., there is to be no smaller inconsistent set of instances).

More efficient procedures for dealing with the first problem were developed by Dunham, Fridsal, and Sward (1959) and by Davis and Putnam (1960). A method for dealing with the second problem was proposed by Prawitz (1960), by which a minimal inconsistent set of instances is always found for every inconsistent formula.¹

2. THE IMPROVED METHOD

Introduction

In this section, I give a somewhat simplified presentation of the method proposed in Prawitz (1960). The main idea is that instead of generating the instances of the given formula $\forall x_1 \forall x_2 \dots \forall x_n M(x_1, x_2, \dots, x_n)$ in some arbitrarily defined order, one should find by calculations the values which substituted for x_1, x_2, \dots, x_n give an inconsistent set of instances. (One may compare the situation with that of solving ordinary equations like $x + 17 = 32$;

¹ For a survey of the development in this area up to 1964, see Cooper (1966).

one may, of course, solve the equation by enumerating the natural numbers, but a more efficient procedure is to use the ordinary subtraction algorithm.) The basic procedure is first to decide whether any substitution for x_1, x_2, \dots, x_n makes $M(x_1, x_2, \dots, x_{2n})$ inconsistent. If this is not the case, one asks whether any substitution for x_1, x_2, \dots, x_{2n} makes $M(x_1, x_2, \dots, x_n) \& M(x_{n+1}, x_{n+2}, \dots, x_{n+n})$ inconsistent; and so on.

One thus requires some method for calculating whether there is any substitution for the variables that makes the formula

$$M(x_1, x_2, \dots, x_n) \& M(x_{n+1}, x_{n+2}, \dots, x_{n+n}) \& \dots \& M(x_{kn+1}, x_{kn+2}, \dots, x_{kn+n}) \quad (k = 0, 1, \dots) \quad (2)$$

inconsistent (other than simply trying the different possibilities).¹

Definitions

Instead of dealing with actual substitutions, I shall consider different *substitution conditions*, which will be built up from equations $t = u$, where t and u are terms (not necessarily in the Herbrand-universe), by using conjunction and disjunction. For instance, $f(x) = f(g(y))$ is an (atomic) substitution condition. A substitution of terms from H_F for the variables in t and u are said to satisfy the atomic substitution condition $t = u$, if t and u are transformed into two occurrences of the same term by the substitution. For instance, the substitution of a for y and $g(a)$ for x satisfies the condition $f(x) = f(g(y))$, but there is no substitution that satisfies the condition $x = f(x)$ or $f(x) = g(y)$. A substitution satisfies a conjunction (disjunction) of substitution conditions, if every (some) condition in the conjunction (disjunction) is satisfied by the substitution. *Example:* the condition $f(x) = f(g(y)) \vee y = f(x)$ is satisfied by the substitution considered above, but the condition $f(x) = f(g(y)) \& y = f(x)$ is not satisfied by any substitution; to satisfy the first equation, we must satisfy $x = g(y)$, and to satisfy both this condition and $y = f(x)$, we must satisfy $x = g(f(x))$, which is impossible.

It is obviously not difficult to set up an algorithm for deciding whether a substitution condition is satisfiable or not.

A pair of atomic formulae $P(t_1, t_2, \dots, t_n)$ and $\sim P(u_1, u_2, \dots, u_n)$ such that the substitution condition

$$t_1 = u_1 \& t_2 = u_2 \& \dots \& t_n = u_n \quad (3)$$

is satisfiable is said to be a *possible contradiction*. The condition (3) is said to be the *corresponding substitution condition* to this possible contradiction.

¹ In simple cases, the question whether (2) is inconsistent may be answered by developing the formula into disjunctive normal form and then 'looking' at the formula. This is essentially the procedure used by Kanger (1959 and 1963), which was developed independently of my method.

THEOREM PROVING

The method

A systematic method for deciding whether any substitution turns a formula of the form (2) into a contradiction is now as follows:

1. Develop the formula (2) into disjunctive normal form.
2. For each clause i in the disjunction obtained by step 1, form the condition $\alpha_1 \vee \alpha_2 \dots \vee \alpha_{j_i}$, called C_i , where $\alpha_1, \alpha_2, \dots, \alpha_{j_i}$ are all the substitution conditions that correspond to possible contradictions among formulae occurring in the clause i .
3. Form the condition $C_1 \& C_2 \& \dots \& C_m$, where C_1, C_2, \dots, C_m are all the conditions obtained in step 2.
4. Decide whether the condition formed in step 3 is satisfiable.

A necessary and sufficient condition for a substitution to turn the formula (2) into a contradiction is obviously that it satisfies the condition formed in step 3. The method for refuting a formula $\forall x_1 \forall x_2 \dots \forall x_n M(x_1, x_2, \dots, x_n)$ is thus to form (2) for successively larger k and go through the steps 1–4 above. The procedure is continued until the question in step 4 is answered by 'yes', which means that the given formula is not satisfiable. To get a set of inconsistent instances of the given formula, one may then take any substitution that satisfies the condition formed in step 3 and carry out this substitution on the last formed formula (2).

3. FURTHER DEVELOPMENTS OF THE IMPROVED METHOD

Introduction

The method described obviously finds a minimal inconsistent set of instances if an inconsistent set of instances exists. As described above, the method uses the technique of developing formulae of propositional logic into disjunctive normal form. Since we now have to deal only with a minimum number of instances of the given formula, this is not a drawback as serious as the one noted in connection with the primitive method, but it is, of course, desirable to develop the method further. One may try, therefore, to combine this method with the methods for deciding whether a formula of propositional logic is inconsistent that were developed by Dunham, Fridsal and Sward and Davis and Putnam, but there is no obvious way of doing that. Attempts to combine the two methods have been made, however, by Davis (1963), and this work has been continued by Chinlund, Davis, Hinman, and McIlroy (1967) and by Loveland (1968). Robinson (1965), whose work has been continued by several authors, has used some of the ideas of the improved method but has developed them in another direction. Davis's method sacrifices some of the strength of the improved method in not finding a minimal inconsistent set of instances (cf. footnote, p. 65). Robinson's method cannot immediately be compared in this respect. It avoids the use of

disjunctive normal form but uses instead another method that seems rather inefficient (cf. footnote, p. 67).

Matrices

The form given to the method as described above was intended to facilitate understanding. When carrying out the procedure in practice, it is, for example, not necessary actually to develop the formula (2) into disjunctive normal form, or to form the whole condition mentioned in step 3. To discuss some questions of this kind I shall now reformulate the method.

Let $M(x_1, x_2, \dots, x_n)$ be written in conjunctive normal form, and let us represent it by the following matrix

$$\begin{array}{l} A_{1,1}, A_{1,2}, \dots, A_{1,n_1} \\ A_{2,1}, A_{2,2}, \dots, A_{2,n_2} \\ \vdots \\ A_{m,1}, A_{m,2}, \dots, A_{m,n_m} \end{array} \quad (4)$$

where we have left out the disjunction signs between the formulae on the same line and the conjunction signs between the different lines. By a *path* in the matrix, is meant a sequence $A_{1,j_1}, A_{2,j_2}, \dots, A_{m,j_m}$; i.e., a path is obtained by taking exactly one literal from each line. The disjunctive normal form of the formula is thus obtained by forming a conjunction of all the literals of a path and then taking the disjunction of all these conjunctions.

We now formulate a necessary and sufficient condition for the existence of a substitution that turns the matrix (4) into a contradiction:

The matrix (4) is made inconsistent by some substitution for the variables iff there exists a set S of atomic substitution conditions with these properties:

1. *Each path of the matrix (4) contains at least one possible contradiction such that the atomic parts of the corresponding substitution condition belong to S .*
2. *The substitution conditions in S are simultaneously satisfiable.*

A set S of substitution conditions that satisfies 1 and 2 will be called a *refutation set* for the matrix (4).

Various methods besides the one described in section 2 may be used to find a set S that satisfies 1 and 2. For instance, one may go through the different paths of the matrix in some order, picking out a possible contradiction for each path and forming the corresponding substitution condition. In this way, one successively builds up a set S of substitution conditions. One has to make sure that each new substitution condition is satisfiable simultaneously with the conditions already in S . If one comes to a path where no such substitution condition can be formed, one has to go back to the last path which has some possible contradiction that has not yet been tried.¹ Erasing the substitution

¹ The method developed by Davis *et al.* differs from the one discussed here in that one does not try all different possible contradictions; some choices of possible contradictions are never reconsidered, and instead one enlarges the matrix.

THEOREM PROVING

conditions that were formed at that and later paths, one then starts anew from that path. If one comes to a path where no substitution condition can be combined with the already formed conditions and there remains no untried possible contradictions in the preceding paths, the procedure ends with a negative result. One has then to form a new matrix by making an alphabetic change of variables and enlarge the old matrix by writing the new one below. The whole procedure can then be repeated.¹

Matrix reduction

The method outlined in the preceding paragraph has some advantages as compared with that described in section 2. The outlined procedure will usually decide whether there is a substitution that makes a given matrix inconsistent in a shorter time. The method is also more economical with respect to space. One need not store the development into disjunctive normal form (the development into conjunctive normal form being a much more straightforward matter); in fact, the information that has to be stored at any one moment is not essentially more extensive than that contained in the minimal refutation itself.

However, the method is still rather time-consuming. When there is a substitution that makes a given matrix inconsistent, one must, with the procedure above, consider all the paths of the matrix in order to find the refutation set. The main problem, it would seem, is to find a faster procedure for building up the refutation set. Such a procedure is also possible to construct; in fact, it is not at all necessary to consider all the paths of the matrix when building up the refutation set (a point already made in Prawitz, 1960, p. 118).

It may be noted that a possible contradiction in a path of a matrix usually belongs to several other paths in the matrix as well. Hence, when we pick out a possible contradiction from one path and introduce the corresponding substitution condition in the set S in order to arrange that this path be in accordance with clause 1 in the definition of refutation set, we have then guaranteed that a number of other paths to which the possible contradiction belongs also conform to clause 1. Therefore, these other paths need not be considered, and one would like to have a procedure that automatically disregards paths that already conform to clause 1. A method of accomplishing this is embodied in the following principle:

Principle of matrix reduction

Let M be a matrix of the form (4) above, let α be the substitution condition that corresponds to a possible contradiction $(A_{i,j}, A_{k,p})$, where $i \neq k$ and none of

¹ The account given above (and some of the improvements in the next paragraphs) are on the whole identical with ideas included in a report written by the author in 1960 and presented to Statens Tekniska Forskningsråd, which supported these researches at that time.

$A_{i,j}$ and $A_{k,p}$ stand alone on their lines (i.e., $n_i, n_k > 1$), and let S be a substitution that satisfies α . Then, S turns M into a contradiction if and only if it turns both M' and M'' into contradictions, where

M' is obtained from M by striking out $A_{k,p}$ and all literals on the line i except $A_{i,j}$, and

M'' is obtained from M by striking out $A_{i,j}$ and all literals on the line k except $A_{k,p}$.

The problem of finding a refutation set for the matrix M is thus reduced to the problem of finding a set containing α that is a refutation set of both M' and M'' . In this way, we cut out not only the paths containing the possible contradiction ($A_{i,j}, A_{k,p}$) but also all paths in the matrix M^* that are obtained from M by striking out $A_{i,j}$ and $A_{k,p}$. To see the validity of the principle, we note that every path in M that does not contain both $A_{i,j}$ and $A_{k,p}$ is also a path in one of the matrices M', M'' and M^* . But if every path in M' and M'' contains a contradiction, then so does every path in M^* . Hence, it is sufficient to consider the paths in M' and M'' . This is equivalent to showing the validity of the equivalence

$$(A \vee B) \& (\sim A \vee C) \& D \equiv (A \& C \& D) \vee (B \& \sim A \& D).$$

Example. Having formed the substitution condition $x = a$, we may replace the matrix M below by the two matrices M' and M'' and try to find a refutation set for them that contains $x = a$:

M	M'	M''
$Qx, f(x), Px$	$Qx, f(x), Px$	$Qx, f(x), P(x)$
$Pa, \sim Qa, f(a), Px$	Pa	$\sim Qa, f(a), Px$
$\sim Px, Qx, y, Qa, f(a)$	$Qx, y, Qa, f(a)$	$\sim Px$
$\sim Pz, \sim Qx, f(a)$	$\sim Pz, \sim Qx, f(a)$	$\sim Pz, \sim Qx, f(x)$

While M contains 36 paths, M' and M'' contain together only 16 paths.

When applying the principle of matrix reduction to a matrix, we shall say that the matrix is *split* into the two simple matrices. We also have the following simpler case of reduction:

Addition to the principle of matrix reductions. Let $M, \alpha, (A_{i,j}, A_{k,p})$, and S be as in the principle above except that either $A_{i,j}$ or $A_{k,p}$ but not both stands alone on its line. Then, S turns M into a contradiction if and only if it turns M' or M'' , respectively, into a contradiction.

Applications of this additional principle are called *simple reductions*.¹

Example continued. If we always use a possible contradiction in the leftmost path when applying the principle of matrix reduction, we see that three simple

¹ This method has some similarities to Robinson's method. However, instead of the principle of matrix reduction, Robinson uses a method that he calls the principle of resolution, which seems less efficient. The principle of resolution is based on the fact that $(A \vee B) \& (\sim A \vee C)$ implies $B \vee C$. By replacing the former formula with the latter one, one loses much information. In contrast, matrix reduction reduces the formula to

THEOREM PROVING

reductions of M' turn M'' into a matrix containing a possible contradiction between literals that both stand alone on their lines. Similarly, one splitting of M'' gives two inconsistent matrices.

Trees of matrices

The principle of matrix reduction may be used thus: given a matrix M , we build up a set S of substitution conditions and a tree structure of matrices in successive steps. The given matrix M is placed at the initial node of the tree. A possible contradiction is located in its leftmost path and the corresponding substitution condition is introduced into S . The principle of matrix reduction is now applied to this possible contradiction. If the matrix is split into two matrices M' and M'' , the tree branches at the node of M , and at the immediately succeeding nodes we place M' and M'' ; if we have a simple reduction, the matrix obtained is placed at the unique node that immediately succeeds the node of M . We then apply the same procedure to the matrix (matrices) obtained by the reduction. If we choose a possible contradiction between two literals that are alone on their lines, the corresponding node is to be an end-node. The procedure terminates with S as a refutation set for M when each branch ends with an end-node.

Before introducing a substitution condition into S , we make sure that the condition is satisfiable simultaneously with S . If at some point we come to a matrix where no such substitution condition compatible with S corresponds to a possible contradiction in the leftmost path, we have to go back to a preceding matrix where an untried possible contradiction remains in the leftmost path, and start anew from this matrix (erasing the matrices at succeeding nodes and the corresponding substitution conditions in S). If there is no such preceding matrix, the matrix at the initial node has to be enlarged as before and the whole process started anew.

4. CONCLUDING REMARKS

Individual variables

The more the variables in a matrix are distinct and the more the individual constants and function symbols are similar, the easier it is to find a substitution that makes the matrix inconsistent. The operation of Skolem-

$(A \& B) \vee (\sim A \& C)$. The two methods have also many other differences, which are not so easy to compare. The above procedure has also some features in common with a procedure proposed by Hans Karlgen.

Simple reductions and matrix splitting bear some resemblance to the rules 1 and 3 respectively, in Davis and Putnam (1960). These rules are applied to formulae that are instances of the given formula. The present reduction principles may be said to be an application of these rules to the situation where the formulae still contain free variables, and, in a way, we have thus combined the ideas in Davis and Putnam (1960) and Prawitz (1960); this combination however is quite different from the one in Davis (1963).

transformation was defined for formulae in prenex normal form but the operation can obviously be extended to formulae in general; the details are left to the reader. One can then often diminish the number of arguments of the function symbols that replaced the existential quantifiers. Furthermore, one can arrange for no variable to have occurrences in two different lines of the matrix (4). (It is also possible, but less important, to replace two different function symbols that have the same number of arguments by the same symbol, if they occur in exactly one line of the matrix (4).) These possibilities are useful especially when applying the procedure to axiomatic theories. They amount essentially to the possibility of treating the axioms separately in the steps preparatory to the main procedure. The number of instances of the matrix that are needed to find an inconsistency can often be diminished in this way.

Alphabetic change of individual variables can also be made when splitting a matrix M into two matrices M' and M'' : such a change can be made for the occurrences of a variable occurring in M'' in lines not affected by the reduction so that they become distinct from the variable occurrences in M' . (This idea was already incorporated in Prawitz (1960) in the form of the so-called interval index.) That this does not invalidate the procedure is seen by realizing the equivalence between the formula

$$\forall x \forall y \forall z \forall v ((Px \vee Qy) \& (\sim Px \vee Rz) \& Sv)$$

and the formula

$$\forall x \forall y \forall z \forall v \forall w ((Px \& Rz \& Sv) \vee (Qy \& \sim Px \& Sw)).$$

Strong minimal procedures

Although the various algorithmic procedures that we have considered in sections 2 and 3 never construct more instances of the initial matrix (4) than are needed for finding an inconsistency, it is possible that some of the matrices constructed are only partially needed. In other words, one may find an inconsistent matrix by choosing different numbers of instances of different lines, a situation which is common when proving theorems in axiomatic theories, where different axioms are not usually invoked the same number of times. Davis (1963) has considered the possibility of guessing the proportion between the number of times different lines of the initial matrix have to be used in order to find an inconsistent matrix.

However, one may also contemplate a procedure that finds an inconsistent matrix which is minimal in the strong sense that no matrix with fewer lines is inconsistent. Starting with a given matrix of the form (4), one may try different possibilities of building up an inconsistent matrix by choosing lines from the given matrix. First, one may consider all matrices containing only two different lines. Only lines that contain literals which together form a possible contradiction need to be considered. If one is to prove theorems in an axiomatic theory and if one is not interested in trying to find an

THEOREM PROVING

inconsistency among the axioms, one may also require that one line comes from a theorem. By applying the principle of matrix reduction, one then investigates whether any one of these matrices can be made inconsistent. If the result is negative, all matrices obtained by different reductions are stored for later use. Instead of forming substitution conditions, however, one may carry out a corresponding substitution. One then considers all possibilities of enlarging the stored matrices by adding a third line containing a literal which forms a possible contradiction together with some literal from the matrix. One then considers all possibilities of reducing these matrices, and so on.

A procedure of this kind requires the storage of much more information, but may have other advantages. By successively enlarging the matrices obtained by different previous applications of the principle of matrix reduction, one utilizes the information from previous attempts to find an inconsistent matrix. Furthermore, two matrices that have been obtained by splitting may now be continued in different ways, that is, by the addition of different lines – and this is an obvious advantage.

It seems difficult to decide which one of these two procedures is the best – the one considered in section 3, or the one now outlined – if this question can be made at all precise. In any case, it seems worthwhile to work out the outlined procedure in more detail and then try both methods.

REFERENCES

- Beth, E. W. (1955) Semantical entailment and formal derivability. *Med. der Kon. Nederl. Akad. Van Wetensch.*, 18, no. 13. Amsterdam.
- Chinlund, T., Davis, M., Hinman, P. & McIlroy (1967) Theorem-proving by matching. *Communications of the ACM*, to appear.
- Cooper, D. C. (1966) Theorem proving in computers. *Advances in programming and non-numerical computation*, pp. 155–82 (ed. Fox, L.). Oxford: Pergamon Press.
- Davis, Martin (1963), Eliminating the irrelevant from mechanical proofs. *Proceedings of Symposia in Appl. Math.*, 15, pp. 15–30.
- Davis, M., & Putnam, H. (1960) A computing procedure for quantification theory. *J. Ass. comput. Mach.* 7, 201–15.
- Dunham, B., Fridsal, R., & Sward, G. (1959) A non-heuristic program for proving elementary logical theorems. *Proceedings of the international conference on information processing*, pp. 282–7. Paris: UNESCO.
- Gilmore, P. C. (1959) A program for the production from axioms of proofs for theorems derivable within the first order predicate calculus. *Proceedings of the international conference on information processing*, pp. 265–73. Paris: UNESCO
- Gilmore, P. C. (1960) A proof method for quantification theory; its justification and realization. *I.B.M. Jl. Res. Devl.*, 4, 28–35.
- Kanger, Stig (1957) *Provability in logic*. Studies in Philosophy, 1. Stockholm: Almqvist and Wiksell.
- Kanger, Stig (1959) *Handbok i logik* (mimeographed). Stockholm.
- Kanger, Stig (1963) A simplified proof method for elementary logic, *Computer programming and formal systems*, pp. 87–94 (ed. Braffort, P. & Hirschberg, D.). Amsterdam.
- Loveland, D. W. (1968) Mechanical theorem proving by model elimination. *J. Ass. comput. Mach.*, 15, 236–51.

- Prawitz, Dag (1960) An improved proof procedure. *Theoria*, **26**, 102-39.
- Prawitz, D., Prawitz, H. & Voghera, N. (1959) Discussion. *Proceedings of the international conference on information processing*, p. 273. Paris: UNESCO.
- Prawitz, D., Prawitz, H., & Voghera, N. (1960) A mechanical proof procedure and its realization in an electronic computer. *J. Ass. comput. Mach.*, **7**, 102-28.
- Robinson, J. A. (1965) A machine oriented logic based on the resolution principle. *J. Ass. comput. Mach.*, **12**, 23-44.
- Skolem, T. (1928) Über die mathematische Logik. *Norsk matematisk Tidsskrift*, **10**, 125-42.
- Skolem, T. (1929) Über einige Grundlagenfragen der Mathematik. *Skrifter utgitt av det Norske Videnskaps-Akademi i Oslo*, I. Mat.-naturv. klasse, no. 4, Oslo.
- Wang, Hao (1960) Towards mechanical mathematics. *IBM JI Res. Dev.*, **4**, 2-22.



Theorem-provers Combining Model Elimination and Resolution

D. W. Loveland

Carnegie-Mellon University

ABSTRACT

A new format for the Model Elimination procedure simpler in structure than the original is presented here. In this form the Model Elimination procedure exhibits a compatibility with the Resolution procedure. Two ways in which this compatibility can be used to design improved theorem-provers are considered, including a strategy designed for problems too complex to be completely solved before memory is filled using either of the procedures mentioned above.

§1. In section 2 we present a new format for the Model Elimination procedure introduced in Loveland (1968). Proofs of the soundness and completeness of the procedure in terms of the new format are given in Loveland (to be published). The structure of the format presented here reveals a kinship between the Model Elimination procedure and the Resolution procedure of J. A. Robinson (see 1965, 1967). In section 3 two classes of strategies that mix resolution and model elimination techniques are discussed. Primary attention is given to a strategy which alternates the use of the Resolution and Model Elimination procedures in an attempt to handle problems too complex to be solved by either procedure in a single 'run'. This paper is not dependent upon knowledge of Loveland (1968), to which we leave the general discussion of the possible advantages of Model Elimination and the notion of 'partially contradictory' set. An appendix to this paper does include some discussion of the operation of the Model Elimination procedure as well as two examples using the format presented here.

By a standard process a closed well-formed formula A of the first-order predicate calculus may be transformed into a well-formed formula (wff) Ω in *Skolem functional form* (for satisfiability) so that Ω is satisfiable if and only if A is satisfiable. Ω has only universal quantifiers, all of which precede the *matrix*, which is assumed to be in conjunctive normal form. In general,

THEOREM PROVING

functions and constant terms will appear in Ω . Herbrand's Theorem states that a wff of form Ω is unsatisfiable if and only if a finite number of clause instances of Ω form a contradictory conjunction. By a clause instance we mean a substitution instance over one of the conjuncts (clauses) of the matrix where the terms of the substitution come from the Herbrand universe U . Here U is formed from the variables of the logic and the constants of Ω according to the standard inductive definition, function symbols of Ω only being employed. If E is an expression, $E\theta$ denotes the expression instance under the substitution θ . In particular, we let $E\xi$, the x -instance of E , denote the simultaneous replacement of all n variables of E by the variables x_1, \dots, x_n . Likewise, $E\eta$ the y -instance of E , denotes the simultaneous replacement of all n variables of E by the variables y_1, \dots, y_n .

Let A_1 and A_2 be two atomic formulae (*atoms*). Following the notation of Robinson (1965, 1967) we say that the set $\{A_1, A_2\}$ is *unifiable* if there is a substitution θ such that $A_1\theta = A_2\theta$. Then θ is said to *unify* $\{A_1, A_2\}$. If $\{A_1, A_2\}$ is unifiable, Robinson (1965) shows that there is a *most general unifier* σ such that $A_1\sigma = A_2\sigma$ and, moreover, for any unifier θ of $\{A_1, A_2\}$ there is a substitution λ such that $A_1\theta = (A_1\sigma)\lambda$ (the Unification Theorem). If L_1 and L_2 are literals we shall say that a *match* exists for L_1 and L_2 with atoms A_1 and A_2 respectively, if there is a most general unifier σ such that $A_1\sigma = A_2\sigma$ and precisely one of L_1 or L_2 has one negation sign preceding the atom. (We assume throughout that no literal contains more than one negation sign.) Two literals are called *complementary* if their atoms are identical and precisely one of the literals has a negation sign preceding the atom.

§2. The basic structure used in the new format is the *chain*, a finite ordered list of literals. Two types of literals may occur in a chain, the *class A* literals (*A-literals*) and *class B* literals (*B-literals*). The most basic chain structure is the *elementary* chain, a list of literals determined by assigning an ordering to the literals of a clause. All literals of an elementary chain are *B-literals*. An important class of elementary chains will be a certain subclass of *matrix chains*; a matrix chain is formed by ordering the set of literals of a clause of the matrix of the given wff Ω . We define the *empty chain* ϕ as a chain having no members. New chains are formed from existing chains by means of three operations to be defined shortly.

We define two useful classes of chains. A chain is *pre-admissible* if the following three conditions are met:

1. if two *B-literals* are complementary they must be separated by an *A-literal*;
2. if a *B-literal* is identical to an *A-literal* the *B-literal* must precede the *A-literal* in the chain;
3. no two *A-literals* have identical atoms.

A pre-admissible chain whose last literal of the chain is a *B-literal* is termed an *admissible* chain.

Each of the three operations has a *parent chain* as input and a *derived chain* as output. (The Extension operation also has a second input, an elementary chain chosen from an *auxiliary set* of chains.) If a literal appears in the derived chain because it is a substitution instance (perhaps under the null substitution) of a literal in the parent clause it is termed a *derived literal* from the corresponding *parent literal* of the parent chain. A sequence of chains K_0, K_1, \dots, K_n is called a *deduction of K_n relative to the wff Ω* if K_0 and the initial auxiliary set (see below) are matrix chains obtained from Ω , and K_i is a derived chain from parent chain K_{i-1} , $1 \leq i \leq n$. We shall usually omit the phrase 'relative to the wff Ω ' for convenience.

In defining the *initial auxiliary set* M_0 of matrix chains, which represents in a certain sense a sufficient set of matrix chains, it is not required that all possible matrix chains that can be created from matrix clauses of Ω be used. Rather, for each literal L of each matrix clause of Ω there is one chain in M_0 whose first literal is L and whose remainder consists of the remaining literals of the matrix clause ordered according to some convenient rule. In the processing of the wff Ω by the procedure certain elementary chains called *lemmas* are developed (which play the identical rôle of the lemmas in Loveland, 1968). These chains may then be used in the same manner as matrix chains for the remainder of the process. We let M_n denote the set consisting of the members of M_{n-1} plus the n th lemma created by the process. We only retain new lemmas which are not substitution instances of previous lemmas or matrix chains. Let M be a variable ranging over the sequence $\{M_n\}$. The set variable M will always be regarded as instantiated and thus regarded as a set; we label M the *auxiliary set* (relative to Ω).

The *scope* is a counter associated with each A -literal of each chain. It is used in the construction of lemmas.

We now define the three operations:

Extension. Given as input are admissible chain K_1 (the parent chain) and elementary chain K_2 from the auxiliary set M . The x -instance $K_1\xi$ and the y -instance $K_2\eta$ are formed. A match is sought between the last literal $L_1\xi$ of $K_1\xi$ and the first literal of $K_2\eta$. If a match does not exist, the Extension operation terminates with no derived chain. If a match exists, let σ represent the most general unifier associated with the match. The derived chain K_3 consists of $K_1\xi\sigma$ with $K_2\eta\sigma$ minus its first literal appended (in original order) after $L_1\xi\sigma$ (so that the last literal of $K_2\eta\sigma$ is the last literal of K_3 unless $K_2\eta\sigma$ is a one-literal chain). $L_1\xi\sigma$ is designated an A -literal with scope 0. Literals of K_3 whose parent literals are A -literals in K_1 are designated A -literals with the same scope as their parent literals. All other literals of K_3 are B -literals.

Reduction. The parent chain is an admissible chain K with a designated A -literal L_1 and a designated B -literal L_2 which occurs later in the ordering of K than L_1 . If a match does not exist for L_1 and L_2 the Reduction operation terminates with no derived chain. If a match exists, let σ be the associated most general unifier. The derived chain K' is $K\sigma$ with the B -literal $L_2\sigma$

THEOREM PROVING

deleted. All literals of K' have their parent's classification, and all literals but possibly $L_1\sigma$ have the parent's scope. Let m be the scope of L_1 in K . If the number n of A -literals (strictly) between L_1 and L_2 is greater than m , then the scope of $L_1\sigma$ in K' is n , otherwise it is m .

Contraction. The parent chain is a pre-admissible chain K . The derived chain K' is an admissible chain formed by deleting all A -literals beyond the last B -literal. Each A -literal L in K' has the same scope as its parent A -literal in K unless the scope exceeds the number n of A -literals in K' beyond the A -literal L . If this occurs the scope is reduced to n . Lemmas are formed during this operation; for this purpose the A -literals are regarded as removed one at a time. As an A -literal L_1 is removed, a lemma is formed consisting of the complement of L_1 and the complement of any preceding A -literal L_2 of K such that the number of A -literals (strictly) between L_2 and L_1 is less than the scope of L_2 . The complement of L_1 is the first literal of the lemma; the remaining literals of the lemma may be ordered by any convenient rule.

Example. Let K be the (non-elementary) chain

$$\underline{P(x_1, f(x_1))} \quad \underline{Q(x_1, x_2)},$$

where underlining denotes an A -literal and ordering is left to right. Let the auxiliary set contain the chain

$$-Q(x_1, f(x_1)) \quad -P(x_1, x_2).$$

We give a deduction of ϕ from K where steps 2, 3, and 4 are obtained by Extension, Reduction and Contraction respectively.

1. $\underline{P(x_1, f(x_1))} \quad \underline{Q(x_1, x_2)}$ given parent chain,
2. $\underline{P(x_1, f(x_1))} \quad \underline{Q(x_1, f(x_1))} \quad -P(x_1, y_1)$ by Extension,
3. $\underline{P((x_1, f(x))} \quad \underline{Q(x_1, f(x_1))}$ by Reduction,
4. ϕ by Contraction.

The lemmas created in passing from step 3 to step 4 are

$$-Q(x_1, f(x_1)) \quad -P(x_1, f(x_1))$$

and $-P(x_1, f(x_1))$.

The first lemma is an instance of the given elementary chain. That the second lemma is not a valid inference from the two chains viewed as clauses reflects the fact that chain K cannot be obtained from the two given chains viewed as elementary chains.

The Appendix contains two further examples of the use of this procedure.

Let $\mathcal{C}_0(\Omega) = M_0$

and $\mathcal{C}_n(\Omega) = \{K \mid \text{there exists a deduction of } K \text{ relative to wff } \Omega \text{ containing at most } n \text{ } A\text{-literals in any chain } K', \text{ where } M(K') \text{ is the auxiliary set employed}\}.$

The set $M(K')$ is that M_m with largest index m such that all members of M_m have been already determined before K' is deduced. The order in which the lemmas are defined (which determines the sequence M_1, M_2, M_3, \dots) need

not be specified here, except that we assume that $M(K')$ includes the lemmas formed in the deduction of K' .

The *soundness* and *completeness* of the procedure outlined is established by the following

Theorem. The wff Ω in Skolem functional form (for satisfiability) is unsatisfiable \leftrightarrow there exists an N such that $\mathcal{C}_N(\Omega)$ contains the empty chain.

A procedure of the above nature designed to develop chains in the order outlined by the sequence $\{\mathcal{C}_n(\Omega)\}$ is called a *Model Elimination* procedure. That this is in behaviour an extension of the Model Elimination procedure in Loveland (1968) is readily seen if one notes that the role of A -literal here is taken by the S -list member in the procedure given in the former paper. The procedure presented in the present paper extends that of the former in that lemmas may be used as matrix clauses for purposes of adding to a chain (a branch in the system given in Loveland, 1968). The proof of the above theorem in the old format appears in Loveland (1968) and in the present format in Loveland (to be published). The definition of the $\mathcal{C}_n(\Omega)$ classes provides the specialized pursuit of the 'partially contradictory sets of clauses' as discussed in Loveland (1968), which is such an important part of the Model Elimination strategy.

§3. In Loveland (to be published) it is established that each lemma of the Model Elimination procedure determines a clause deducible by the Resolution procedure from the same matrix clauses. Given a lemma of Model Elimination the corresponding clause is simply the set of literals comprising the lemma. (A clause is sometimes treated as a set of literals as well rather than as a disjunction of them. We shall use both notions with little danger of confusion.) Likewise, any clause formed by resolution from a given set of initial clauses may, upon assigning an ordering to the literals of the clause thus forming an elementary chain, be added to the auxiliary set without affecting the soundness of the Model Elimination procedure. The theory of Resolution (*see* Robinson 1965, 1967) assures us that adding the clause to the original set gives a set unsatisfiable if and only if the original is unsatisfiable. Thus the soundness of the Model Elimination procedure is unaffected by such additions to the initial auxiliary set. Adding the chain to the auxiliary set at a later point in processing, instead of adding the chain initially, only restricts the chains that can be produced.

These remarks are the basis for two classes of strategies for the interaction of resolution and model elimination. (For convenience we sometimes use the term 'model elimination', no capital letters, to denote any essential part of, or the total, process of the Model Elimination procedure in the same manner as the term 'resolution' serves for the Resolution procedure.) We shall merely outline the classes of strategies. The strategies considered first, use of the Resolution and Model Elimination procedures in iterated sequence, introduce some simple concepts which may be of interest. The second class of strategies involves use of resolution within the model elimination process.

THEOREM PROVING

Although computers are approaching their theoretical speed barrier, the cost-per-operation barrier is nowhere in sight. The physical components that promise to cut computer cost of operation drastically are well into the development stage. The widespread adoption of time-sharing also seems inevitable. The result for mechanical theorem-proving is that theorem-provers will be permitted to run hours rather than minutes, and will work overnight on a theorem for the cost of a mathematician to puzzle over it during the day. The computers and time-sharing equipment to allow this may not be far in the future, especially if our memory requirements are not too demanding. The problem this presents for theorem-proving programs is: how do we design theorem-provers that operate effectively for longer than the capacity cutoff point of their normal procedures? One way to push back this barrier is to run until memory is exhausted, then split the existing data (such as clauses or chains) into segments, probably in such a way that deductions are preserved, retain only one segment and with the newly released storage space continue the deductions a few more levels. The other segments, which have either been stored on tape or are recomputed when wanted, are each extended several levels in turn. This is then repeated with the new results of each calculation; a new segment is recalled or recomputed and then extended. One is confronted with a 'paging' type programming problem with all its inherent problems of data transfer. If recomputation is chosen, the problem of segment description must be faced. The 'segmenting' approach does have the questionable virtue of theoretical completeness.

Another approach is severe trimming with no explicit intent of retrieving the data deleted. The trimming done when capacity is approached differs in quality from the better known trimming heuristics suggested for the Resolution procedure, for example. The latter type heuristic deletes clauses deemed unnecessary for some specific reason. These might be called *procedure heuristics*. For convenience we shall label the other kinds of heuristics *capacity heuristics*. In order to allow the problem to progress at all from the point where memory capacity is reached, the capacity heuristics must cut drastically, aiming more at salvaging the few pieces of data most likely to be useful than discarding only the unnecessary. The question of capacity heuristics preserving completeness is almost irrelevant. Capacity heuristics exist because of a physical upper bound only. Such physical upper bounds convert every complete procedure into an incomplete procedure.

We wish to outline one class of procedure that uses a capacity heuristic. One reason for the interest in this approach as opposed to 'segmenting' is the reduced programming requirement. We believe, however, that many theorems will yield to this approach; one reason for optimism is the observed fact that a set of clauses that is refutable usually has many paths to the establishment of this fact. Many paths may be blocked yet a solution still be found.

Let us consider possible measures of 'interest' for clauses created by the Resolution procedure and for lemmas created by the Model Elimination

procedure. One possible measure of 'interest' of a clause is proportional to the order of the first level at which the clause is generated and inversely proportional to the number of literals in the clause and to the depth of function nesting of its most complex term. For lemmas we first need the concept of deficit of a chain. The *deficit* of a chain K is the non-negative integer $n - m$, where $K \in \mathcal{C}_n(\Omega) - \mathcal{C}_{n-1}(\Omega)$ and m is the number of A -literals in K . As n measures the maximum number of A -literals needed in any chain in the deduction of K , the deficit $D(K)$ measures in a sense the amount of collapse towards the empty chain that K represents relative to the 'longest' chain in the deduction of K . We shall call $D(K)/n$ the *value* of K . The value of the empty chain is 1 and for every non-empty chain K , the value $V(K)$ satisfies $0 \leq V(K) < 1$. The measure of interest for a lemma is the value of the derived chain of the Contraction operation which creates the lemma. If several deductions of the lemma exist, we choose the least of the values associated with the lemma.

A similar notion of value for a clause C is easily defined. The *deficit* $D(C)$ of a clause is the non-negative integer $n - m$, where n is the total number of literal instances that appear in the deduction of C , and m is the number of literals of C . The *value* $V(C)$ of C is given by $D(C)/n$ and satisfies $0 \leq V(C) \leq 1$ with only the empty clause achieving value 1. Another measure of interest for a clause is its value.

We now outline a particular theorem-prover using capacity heuristics, which utilizes both Resolution and Model Elimination. Let S denote the set of given clauses. First, the Resolution procedure (with appropriate heuristics) is run over S with no restricted set of support. If the empty clause is not realized, a few sufficiently scattered clauses of highest interest are added to the set S . Call this set S_1 . By a 'sufficiently scattered' set we mean that no clause in the set can be derived from another clause with only a few resolutions. We do not envisage an exhaustive test for this property, but as clauses of high interest generate like clauses as immediate resolvents, the wasteful retention of two clauses, one easily derivable from the other, should be avoided when convenient.

As a second step, the clauses of S_1 are converted to chains and the Model Elimination process is run to capacity. Then a few lemmas (rather, their associated clauses) of highest interest and sufficiently scattered are added to S_1 to define set S_2 . The Resolution procedure is applied to S_2 . We limit the set of support (see Wos *et al.* 1965) to the new clauses. This leads to S_3 , to which Model Elimination is applied. This pattern is continued until successful or manually halted.

There are obviously many possible modifications; we mention one point. It seems desirable to keep the number of chains produced at any given depth as small as possible (see the discussion below). Some non-matrix chains might be restricted so that they are not free for arbitrary use by the Extension operator. A way of handling the two classes of chains is discussed later in this section.

THEOREM PROVING

Why may such a mixture of procedures be preferable to iterations of either procedure alone? The reasons stem from the tendency of the two procedures to complement each other in certain aspects; each tends to offset relative weaknesses of the other. We consider two such complementary properties.

Resolution combines clauses at a rapid rate. It is particularly effective when a fair number of these recombine several times or more to yield the empty clause. (This does not imply the Model Elimination procedure is *necessarily* inferior for such problems; see example 2 of the Appendix.) Moreover, just as matrix clauses are used more than once in general, so are certain derived clauses. If these are available for direct use instead of having to be implicitly rederived, an often substantial saving is realized. The repeated use of certain clauses (actually chains) is exhibited in the Model Elimination proof of example 2 of the Appendix. Remember that any direct use of a lemma in deriving the empty chain is a (successful) second use. The first use occurs in the chain leading to the definition of the lemma. The Resolution procedure is used to generate such useful clauses (hopefully). For example, clauses of high interest which are combinations of the axioms of a theorem may often be useful 'lemmas' (in the general sense) such as useful reductions of several applications of associativity in group theory. Certain 'promising' clauses of this type should be unrestricted for the Extension operator in the subsequent Model Elimination run.

Some solutions do not make much use of the resolved clauses created 'in advance'. Their proofs by resolution show almost all clauses with one parent clause a matrix clause. For such situations the Model Elimination procedure is often quite desirable if there are clauses containing more than two literals. (If no clause contains more than two literals, the length of the longest chain in the derivation of the empty chain may often be close to n , where n is the number of clauses in the underlying contradictory set. The Resolution procedure should find the empty clause at a considerably lower level.) If the auxiliary set is kept small (remember that only the matrix chains are required for completeness), the growth rate of the $\mathcal{C}_n(\Omega)$ is relatively slow, and a fair depth should be reached before cutoff. This, plus the need to locate 'partially contradictory sets' (see Loveland 1968) only within any one chain, gives Model Elimination considerable ability to handle problems requiring extensive combining of elementary parts. In this setting some of the 'elementary' parts may be clauses derived after considerable processing. A crude summary of this polarization is 'Resolution constructs the building blocks, Model Elimination manipulates them.'

Resolution and Model Elimination also complement each other in the way each deals with its working unit, the clause and chain, respectively. The distinction is well exhibited in the different measure of length for clause and chain. Clause length is simply the number of literals in the clause. Chain length is the number of A -literals in the chain. A matrix clause with many literals is an undesirable parent clause because the resolvent is almost certainly

a long clause. In contrast, any matrix chain adds only a unit length to any chain to which it is attached. Moreover, a contradictory set of clauses with many long clauses is more easily found by model elimination than a contradictory set of the same size but with shorter clauses, for longer clauses lead to the creation of more chains but of shorter average length. As the main parameter of computation time is the length of the longest chain, an overall benefit is realized from the longer clauses. Example 1 of the Appendix illustrates how long matrix clauses result in numerous short chains with subsequent benefit to ease of processing.

We plan to evaluate the proposed theorem-prover using a program being written for the CDC 6600 to implement the Model Elimination procedure. We anticipate relatively little programming trouble to implement the Resolution procedure once the first program is complete. (The addition of Model Elimination given a Resolution program should also be easy.) Because the operating system used with the CDC 6600 (located at New York University) runs seven programs 'simultaneously', we expect to do most of our experimentation with about one-seventh of the machine capacity available to us. This served as motivation for considering techniques for extending the 'machine capacity' in some artificial way. We expect to obtain some solutions that cannot be obtained (with our program) within a single 'run' to capacity. However, we also expect any such solution to come within a very low number of iterations of the cycle described above. Hard problems (perhaps by definition) undoubtedly have long clauses (or chains) in the initial stages of processing which do not begin to reduce in length until a level well beyond that obtainable with reasonable memory capacity. The proposed measures of interest clearly will not retain the appropriate clause or chain for further development in this case. Problem-oriented measures of interest will have to be developed to tackle such situations or the 'blunderbuss' approach of segmentation outlined earlier may have to be employed.

We consider now the second class of strategies for mixing resolution and model elimination techniques. This concerns the use of resolution within the framework of model elimination. At the beginning of this section it was remarked that the lemmas of model elimination may be treated as clauses. In particular they may be resolved with other lemmas or with matrix clauses. The question of when this is useful is not easily answered except when two unit clauses can be resolved to obtain the empty clause. The set of matrix chains forms a sufficient auxiliary set to guarantee completeness; adding chains to the auxiliary set should be done only when concrete heuristic justification can be given. The reason is clear: any addition to the auxiliary set yields another elementary chain which the Extension operator can use to extend chains. The size of $\mathcal{C}_n(\Omega)$ grows substantially (for fixed n) with each addition even for small n .

This argument concerning addition of chains to the auxiliary set affects the addition of lemmas also, of course. Limited experience of several

THEOREM PROVING

hand-computed examples shows the lemmas to be useful chiefly when the resulting chain can subsequently be reduced to a pre-admissible chain. It seems desirable to introduce the heuristic of a *split auxiliary set* which contains two classes of chains. The first class of chains can be used freely for extension of any chain. Matrix chains generally belong to this class. The second class of chains may only be used for extension when this leads to contraction (possibly after suitable reductions). Lemmas will in general belong to this class. This limitation, incidentally, is 'built in' in the format of Loveland (1968).

Chains entered in the second class of the auxiliary set do not expand the growth rate of the sets $\mathcal{C}_n(\Omega)$. The criteria for adding chains to this class are clearly less stringent. Here one might enter chains from clauses which are resolvents of two clauses from two existing lemmas, for example.

One other observation should be made. Every chain corresponds to a clause formed from the set of B -literals of the chain. It is not at all clear that this fact yields a useful modification of the Model Elimination procedure. It might be useful if in the middle of a solution using the Model Elimination procedure the machine decides to 'throw in the towel' and convert entirely to a Resolution procedure.

APPENDIX

This appendix gives two examples of the use of the Model Elimination procedure in the present format. Both examples appear in the literature, where they illustrate specific heuristics added to Resolution. Each illustrates a characteristic of the Model Elimination procedure that is of some interest.

As a compromise between ease of reading and ease of reporting, we adopt the following conventions in listing the deduction of the empty chain in the following examples. The matrix clauses are numbered and the literals of each clause are regarded as consecutively lettered by the first letters of the alphabet. A particular matrix chain is indicated by the clause number and letter of the first literal. (The order of the remaining literals of the chain is apparent from context.) The consecutive chains listed in an example represent progressive 'snapshots' of the deduction, determined as follows. Successive extensions are entered on the same line until a substitution in a literal already recorded occurs, or a reduction or contraction is required. A new chain is used to record any of the situations just listed. (A new chain is sometimes also written to break up an otherwise lengthy chain.) The A -literals are underlined. Negations are indicated by minus signs. The notation $\dots n \dots$, where n is an integer indicates that the first n literals of the previous chain appear unchanged as the first n literals of the present chain.

Example 1. This example appears as Example 2 in Wos *et al.* (1964) and as Example 4 in Luckham (1968). It represents the negation of the following theorem: in an associative system with an identity element, if the square of every element is the identity, the system is commutative. The predicate $P(x,y,z)$ is to be interpreted as ' $x \cdot y = z$ '. The wff Ω to be refuted is the

universal closure of the conjunction of the following clauses. (The disjunction symbol between literals is suppressed.)

1. $P(x,e,x)$
2. $P(e,x,x)$
3. $-P(x,y,u) - P(y,z,v) - P(u,z,w)P(x,v,w)$
4. $-P(x,y,u) - P(y,z,v) - P(x,v,w)P(u,z,w)$
5. $P(x,x,e)$
6. $P(a,b,c)$
7. $-P(b,a,c)$

The last two clauses come from the negation of the conclusion of the theorem.

The derivation given below has depth 2, i.e., has the empty chain in $\mathcal{C}_2(\Omega)$. One starts with matrix chain 3a, which is one-half the statement of the associative law. Starting with chain 7, part of the negation of the conclusion, the empty chain ϕ is derived at depth 4. (As these examples are hand-computed the author cannot be sure the optimal result is achieved, but here it seems likely.)

This situation is interesting for the following reason. The Model Elimination procedure satisfies a 'set of support' strategy (see Wos *et al.* 1964 or 1965) which here may be stated: if a matrix clause has an instance in a contradictory set then there is a deduction of ϕ having a matrix chain from that clause as the first chain of the derivation. A clause from the negation of the conclusion is quite a safe bet as an initial clause of a deduction of ϕ . It would seem reasonable to limit the $\mathcal{C}_n(\Omega)$ to chains with derivations from chains 6 or 7 only as the size of $\mathcal{C}_n(\Omega)$ is reduced with no loss of completeness. Here the judgement is costly, however, for ϕ is achieved earlier with another initial chain.

There are reasons to believe this phenomenon may occur frequently, particularly in considerably more complex examples. One is that multiliteral clauses which should occur frequently in various instances in a derivation (such as associativity in group theory) should have an instance located so as best to balance the lengths of the longer chains of the derivation, often avoiding the occurrence of one long chain with many shorter chains. As the length of the longest chain almost entirely determines the difficulty of deriving ϕ , a 'balancing' of longer chains is clearly desirable. This situation is best understood by use of the notion of proof tree as given in Loveland (1968), with which the effects of balancing chain length are easily pictured. We leave to the interested reader as an exercise the further pursuit of this point.

It is interesting to note that the empty clause was realized at level 5 using Resolution as exhibited by the two proofs given in Wos *et al.* (1964) (and levels 6 and 10 in Luckham 1968). Thus, even with the good strategies of Wos *et al.* and Luckham keeping the number of clauses generated by level 5

THEOREM PROVING

(6 resp.) to a small number, the Model Elimination procedure with no further heuristics should produce fewer chains in deriving ϕ at depth 2.

<i>chain</i>	<i>elementary chains used</i>
1. $-P(x,y,u) - P(y,z,v) - P(u,z,w)P(x,v,w)$	3a
2. ... 3 ... $\underline{P(x,v,w) - P(v,s,t) - P(x,t,r)P(w,s,r)}$	4a
3. ... 2 ... $\underline{-P(u,z,b)P(x,v,b) - P(v,a,t) - P(x,t,c)P(b,a,c)}$	7
4. $-P(c,y,u) - P(y,z,v) - P(u,z,b)\underline{P(c,v,b) - P(v,a,e) - P(c,e,c)}$	1
5. $-P(c,y,u) - P(y,z,a) - P(u,z,b)\underline{P(c,a,b) - P(a,a,e)}$ new lemma formed: $-P(c,a,b)$	5
6. $-P(c,y,e) - P(y,b,a) - \underline{P(e,b,b)}$	2
7. $-P(c,y,e) - \underline{P(y,b,a) - P(x,s,y) - P(s,b,v) - P(x,v,a)}$	4d
8. ... 2 ... $\underline{-P(a,s,y) - P(s,b,e) - P(a,e,a)}$	1
9. ... 2 ... $\underline{-P(a,b,y) - P(b,b,e)}$	5
10. $-P(c,c,e) - \underline{P(c,b,a) - P(a,b,c)}$ new lemma formed: $P(c,b,a)$	6
11. $\underline{-P(c,c,e)}$	5
12. ϕ .	

Example 2. This example appears as Example 7 of Luckham (1968). It represents the negation of the following theorem: any number greater than 1 has a prime divisor. The predicates $D(x,y)$, $L(x,y)$ and $P(x)$ are interpreted as 'x divides y', 'x < y' and 'x is a prime' respectively. The value $g(x)$ represents a non-trivial divisor of x, if x is composite and a is the least counter-example to the theorem. (The value $f(x)$ represents the prime divisor of every x such that $1 < x < a$.) The wff Ω to be refuted is the conjunction of the following clauses.

1. $D(x,x)$
2. $-D(x,y) - D(y,z)D(x,z)$
3. $P(x)D(g(x),x)$
4. $P(x)L(1,g(x))$
5. $P(x)L(g(x),x)$
6. $L(1,a)$
7. $-P(x) - D(x,a)$
8. $-L(1,x) - L(x,a)P(f(x))$
9. $-L(1,x) - L(x,a)D(f(x),x)$

We include for comparison the shorter of two Resolution proofs given in Luckham (1968). The empty clause is achieved at level 5. We present a derivation of the empty chain ϕ from Ω which has depth 5.

This example illustrates the use of lemmas, which are used here only when they are immediately followed by contractions. It also shows that even on examples where Resolution performs quite efficiently the Model Elimination procedure can perform competitively. This is not a claim that Model Elimination always is at least as efficient as Resolution; it often is not. Resolution is often more efficient when no matrix clause contains more than two literals, or when the deduction of the empty clause contains a number of resolutions where both parent clauses were formed themselves just a few levels earlier (such as in this example). The latter situation cannot in general be ascertained by inspection of the set of matrix clauses. Even if either of the above cases hold, it does not seem predictable when Resolution will be more efficient than Model Elimination. Model Elimination is in general more efficient than Resolution when at least one matrix clause contains more than two literals and the deduction of the empty clause *via* Resolution uses a matrix clause in (almost) every resolution. Again, the latter property cannot generally be determined by inspection of the matrix clauses. Of course, the heuristics that can be brought to bear on each procedure will greatly influence their relative efficiencies.

<i>chain</i>	~	<i>elementary chains used</i>
1. $-L(1,g(a)) \underline{D(f(g(a)),g(a))} - \underline{L(g(a),a)} P(a)$		9a, 5b
2. $\dots 3 \dots \underline{P(a)} - \underline{D(a,a)}$ new lemmas formed: $-P(a)$ and $L(g(a),a)$		7a, 1
3. $-L(1,g(a)) \underline{D(f(g(a)),g(a))} - \underline{D(g(a),a)} \underline{D(f(g(a)),a)}$ $\quad - \underline{P(f(g(a)))} - \underline{L(1,g(a))} - \underline{L(g(a),a)}$		2a, 7b 8c, lemma
4. $\dots 5 \dots - \underline{L(1,g(a))} P(a)$ new lemmas formed: $L(1,g(a),P(f(g(a))))$ and $-D(f(g(a)),a)$		4b, lemma
5. $-L(1,g(a)) \underline{D(f(g(a)),g(a))} - \underline{D(g(a),a)} P(a)$ new lemmas formed: $D(g(a),a)$ and $-D(f(g(a)),g(a))$		3b, lemma
6. $- \underline{L(1,g(a))}$		lemma
7. ϕ .		

Note that the initial chain 9a has the last two literals reversed from the 'natural' ordering given in the listing of clause 9. Though perfectly permissible it can be argued with reason that this interchange requires insight. Without this interchange, the deduction of ϕ requires depth 6 for the lemma $L(g(a),a)$ then is not available as in the given proof. If Model Elimination is more efficient than Resolution in this example, it is because about the same depth is required as for Resolution and only matrix chains need to be used to extend chains (that do not immediately contract). Resolution apparently cannot realize a level 5 deduction with use only of resolutions involving

THEOREM PROVING

matrix clauses for one parent. In any case, it is conservative to estimate that Model Elimination is as efficient here as Resolution in a case where the deduction by Resolution given below suggests Resolution has been reasonably efficient.

We present the proof by Resolution given in Luckham (1968).

<i>clause</i>	<i>source</i>
10. $P(a)D(f(g(a)),g(a)) - L(1,g(a))$	from 9, 5
11. $P(a)P(f(g(a))) - L(1,g(a))$	from 8, 5
12. $D(f(g(a)),g(a))P(a)$	from 4, 10
13. $D(z,x) - D(z,g(x))P(x)$	from 3, 2
14. $P(f(g(a)))P(a)$	from 4, 11
15. $D(f(g(a)),a)P(a)$	from 13, 12
16. $P(a) - D(f(g(a)),a)$	from 7, 14
17. $P(a)$	from 15, 16
18. $-P(a)$	from 7, 1
19. Empty clause	from 17, 18

Acknowledgement

The author was supported by NSF Grant GP-7064 during the writing of this paper.

REFERENCES

- Loveland, D. W. (1968) Mechanical theorem proving by model elimination. *J. Ass. comput. Mach.*, **15**, 236-51.
- Loveland, D. W. A simplified format for the model elimination theorem-proving procedure (to be published).
- Luckham, D. (1968) Some tree-parsing strategies for theorem proving. *Machine Intelligence 3*, pp. 95-112 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
- Robinson, J. A. (1965) A Machine-oriented logic based on the resolution principle. *J. Ass. comput. Mach.*, **12**, 23-41.
- Robinson J. A. (1967) A review of automatic theorem-proving. *Annual symposia in applied mathematics. XIX*. Providence, Rhode Island: American Mathematical Society.
- Wos, L. T., Carson, D. F. & Robinson, G. A. (1964) The Unit Preference Strategy in Theorem Proving. *AFIPS*, **26**, 615-21, Fall, J. C. C. Washington, D.C.: Spartan Books.
- Wos, L. T., Carson, D. F. & Robinson, G. A. (1965) Efficiency and completeness of the set-of-support strategy in theorem-proving. *J. Ass. comput. Mach.*, **12**, 536-41.

Semantic Trees in Automatic Theorem-Proving

R. Kowalski

and

P. J. Hayes

Metamathematics Unit
University of Edinburgh

INTRODUCTION

We investigate in this paper the application of a modified version of semantic trees (Robinson 1968) to the problem of finding efficient rules of proof for mechanical theorem-proving. It is not our purpose to develop the general theory of these trees. We concentrate instead on those cases of semantic tree construction where we have found improvements of existing proof strategies. The paper is virtually self-contained and to the extent that it is not, Robinson's review paper (1967) contains a clear exposition of the necessary preliminaries.

After dealing with notational matters we define a notion of semantic tree for the predicate calculus without equality. A version of Herbrand's theorem is then proved. The completeness of clash resolution (Robinson 1967) is proved and it is shown that restrictions may be placed upon the generation of all factors when resolving a latent clash. The completeness of binary resolution is proved by specializing the notion of clash, and an ordering principle is shown to be complete when used in conjunction with it. Slagle's AM-clashes (1967) are shown to be complete by another specialization, and some clarification is presented of the rôle of Slagle's model M at the general level. A further specialization of AM-clashes is then made to the case of hyper-resolution (Robinson 1965a) and renaming (Meltzer 1966). It is shown in this case how the restrictions on generating factors and Slagle's A-ordering can be combined to give a highly efficient refutation procedure. Moreover, additional restrictions on the generation of factors are obtained for all cases of AM-clashes by employing throughout a modified notion of A-ordering. In the last section we report on attempts to apply the methods of semantic trees to the construction of inference systems for the predicate calculus with equality.

THEOREM PROVING

PRELIMINARIES

Familiarity is assumed with the reduction of sentences to clausal form. Atomic formulae are sometimes referred to simply as *atoms*. *Literals* are atoms or their negations; *clauses* are disjunctions of literals. Disjunctions and conjunctions will often be identified with the sets of their disjuncts and conjuncts respectively. Thus one may speak of a literal L occurring in a clause C and write $L \in C$. The *null disjunct* \square is always false and therefore identical to the truth value *false*.

The result of applying a substitution σ to an expression E is denoted by $E\sigma$. If $E\sigma = F$ for some σ , then F is said to be an *instance* of E . In case F contains no variables F is a *ground expression* and a *ground instance* of E . If F is an instance of E and E of F , then E and F are *variants*. If expressions E and F have a common instance G , then E and F are *unifiable* and there is a most general common instance $E\sigma = F\sigma$, where σ is the *most general unifier (m.g.u.)* of E and F . The m.g.u. σ of E and F is such that if μ is any unifier of E and F then there is a λ such that $\mu = \sigma\lambda$.

Constants are functions of zero arguments. The *Herbrand universe* \mathbf{H} of a set S of clauses is the set of all terms constructible from the function letters appearing in S (augmented by a single constant symbol if S contains no constant symbols). The *Herbrand base* $\hat{\mathbf{H}}$ is the set of all ground instances over \mathbf{H} of atoms occurring in S . If K is a set of ground atoms, then by a *complete assignment* to the set K we mean a set \mathcal{A} such that for every atom $A \in K$ exactly one of the literals A or \bar{A} occurs in \mathcal{A} and \mathcal{A} contains no other members. If \mathcal{A} is a complete assignment to some subset $K' \subseteq K$, then \mathcal{A} is called a *partial assignment* to K . Given a set S and its Herbrand base $\hat{\mathbf{H}}$ any complete assignment \mathcal{A} to $\hat{\mathbf{H}}$ can be considered as a possible interpretation of S (i.e., the universe of the interpretation is \mathbf{H} ; the definition of the functions over \mathbf{H} is incorporated in the definition of \mathbf{H} ; and an n -place predicate P holds for (t_1, \dots, t_n) , $t_i \in \mathbf{H}$ if and only if $P(t_1, \dots, t_n) \in \mathcal{A}$).

Every tree is a partially ordered set T whose elements are its nodes. We shall use \leq to refer to the partial ordering of the nodes. The unique node $N \in T$ such that $N \geq N'$ for every node N' is the *root* of the tree. Trees will be considered as growing downward. Thus the root of a tree is the highest node in the tree, and if there are at most finitely many nodes immediately below any node then the tree is finitely branching. A *tip* of a tree T is a node N which is above no other node. A *branch* of T is a sequence of nodes beginning with the root and such that each other node in the sequence lies immediately below the preceding node in that sequence. A branch of T is *complete* if either it is infinite or else it is finite and ends in a tip.

SEMANTIC TREES FOR THE PREDICATE CALCULUS WITHOUT EQUALITY

Definitions

Let K be a set of atoms. A finitely-branching tree T is a *semantic tree* for K

when finite sets of atoms or negations of atoms from K are attached to the nodes of T in such a way that

- (i) the empty set is attached to the root and to no other node;
- (ii) if nodes N_1, \dots, N_n lie immediately below some node N and the sets of literals \mathcal{B}_i are attached to the nodes N_i , then $\hat{\mathcal{B}}_1 \vee \dots \vee \hat{\mathcal{B}}_n$ is a tautology, where $\hat{\mathcal{B}}_i$ is the conjunction of the literals in \mathcal{B}_i ;
- (iii) the union of the sets of literals attached to the nodes of a complete branch of T is a complete assignment to K .

Given a set S of clauses and a semantic tree T for $\hat{\mathbf{H}}$ (the Herbrand base of S), then the union of all the sets attached to any complete branch of T is a complete assignment to $\hat{\mathbf{H}}$ and therefore a possible interpretation of S . Indeed it can be easily shown from condition (ii) of the definition that every complete assignment \mathcal{A} to $\hat{\mathbf{H}}$ can be obtained in this way.

The partial assignment which is the union of all the sets of literals attached to the nodes of a branch ending in a node N is written \mathcal{A}_N and is termed the *assignment at N* . In this notation the set \mathcal{B}_i attached to N_i , referred to in (ii) above, is just $\mathcal{A}_{N_i} - \mathcal{A}_N$.

The only case of an infinite semantic tree that we shall consider in this paper is that of a *simple binary tree*, which is used in the proof of the version of Herbrand's theorem necessary for our applications. In this tree if N_1 and N_2 lie immediately below the node N , then \mathcal{B}_1 and \mathcal{B}_2 are just $\{A\}$ and $\{\bar{A}\}$ respectively for some ground atom A in K . Every other semantic tree considered will be a finite *clash tree*. If T is a clash tree, $N \in T$ and N_1, \dots, N_k, N_{k+1} lie immediately below N , then the set \mathcal{B}_i attached to N_i for $1 \leq i \leq k$ is just $\{L_i\}$ and the set \mathcal{B}_{k+1} attached to N_{k+1} is $\{\bar{L}_1, \dots, \bar{L}_k\}$, where $\{L_1, \dots, L_k\}$ is a partial assignment to K disjoint from the partial assignment \mathcal{A}_N . The nodes N_1, \dots, N_k are termed *satellite nodes* and the node N_{k+1} , a *nucleus node*.

Failure

If S is a set of clauses and T a semantic tree for $\hat{\mathbf{H}}$, then T is in some sense an exhaustive survey of all possible interpretations of S . If S is in addition unsatisfiable, then S fails to hold in each of these interpretations. These considerations motivate the definitions given below.

Let T be a semantic tree and C a clause. We say that C *fails at a node* $N \in T$ when C has a ground instance $C\sigma$ such that \mathcal{A}_N logically implies $\neg(C\sigma)$. (We also write $\mathcal{A}_N \vDash \neg(C\sigma)$, using the symbol \vDash to denote *logical implication*). Note that if C fails at N then $\mathcal{A}_N \vDash \neg C$. The converse, however, is not in general true. For if $\mathcal{A}_N = \{P(a), P(f(f(a)))\}$ and $C = \bar{P}(x) \vee P(f(x))$ then $\mathcal{A}_N \vDash \neg C$, but C does not fail at N .

Let T be a semantic tree and S a set of clauses. A node $N \in T$ is a *failure point for S* when some clause $C \in S$ fails at N but no clause in S fails at any node $M > N$. If N is a failure point for S and $M > N$, then M is a *free node* for

THEOREM PROVING

S . Note that if N is free for S then any node $M > N$ is also free for S and both M and N are free for any subset of S . Also if N is a failure point for S , then no node $M < N$ is free for S and both M and N are not free for any superset of S .

A semantic tree every branch of which contains a failure point for S is said to be *closed* (for S).

Herbrand's Theorem

The following is easily shown to be equivalent to Herbrand's Theorem.

Theorem 1. If S is an unsatisfiable set of clauses then there is a finite subset $K \subseteq \hat{H}$ such that every semantic tree T for K is closed for S .

Proof. Let (A_1, \dots, A_n, \dots) be an enumeration of the Herbrand base of S and let T' be a simple binary tree for \hat{H} constructed as follows: the empty set ϕ is attached to the root of T' ; the sets $\{A_1\}$ and $\{\bar{A}_1\}$ are attached to the two nodes immediately below the root; and if either $\{A_n\}$ or $\{\bar{A}_n\}$ is attached to the node N , then the sets $\{A_{n+1}\}$ and $\{\bar{A}_{n+1}\}$ are attached to the nodes immediately below N . Any complete branch through T' represents a complete assignment \mathcal{A} to \hat{H} and therefore is a possible interpretation of S . Since S is unsatisfiable, \mathcal{A} fails to be a model of S and some clause $C \in S$ must be false in \mathcal{A} . It follows that some ground instance $C\sigma$ of C must be false in \mathcal{A} . But for this to happen the complement of each literal in C must occur in \mathcal{A} , and since there are only finitely many such literals they must occur already in some partial assignment \mathcal{A}_N with $C\sigma$ false in \mathcal{A}_N . Thus, some $M \geq N$ is a failure point for S and T' is closed for S .

The number of nodes of T' free for S is finite, for otherwise, by König's lemma we could find an infinite branch of free nodes containing no failure point. Let k be the length of the longest branch of T' which ends in a failure point and let $K = \{A_1, \dots, A_k\}$. Then every branch of T' corresponding to a complete assignment to K already contains a failure point for S . Now if T is any semantic tree for K then every complete branch corresponds to a complete assignment to K and must also contain a failure point for S . Therefore T is closed for S . Q.E.D.

Note that Robinson (1967) uses essentially the same tree T' in his proof of Herbrand's theorem. The semantic trees of this paper differ, however, from those of Robinson (1968). Robinson defines failure of a clause at a node of a semantic tree for ground clauses and establishes his main results for ground clauses first. These results are then 'lifted' to the general level by applying Herbrand's Theorem. By generalizing the definition of failure and by applying Herbrand's Theorem in the form above, we establish our results for the general level directly. A principal advantage of this modification is that it becomes clear how to restrict the generation of factors of clauses.

Inference node

The concept of inference node makes it possible to transfer from the semantics of semantic trees to the syntax of inference systems. A node N of a semantic

tree T is an *inference node* for a set of clauses S if N is free for S and the nodes immediately below N are failure points for S . Note that if T is closed for S and $\square \notin S$, then T contains an inference node. For if $\square \in S$, then \square fails at the root of T , and T contains neither free nodes nor inference nodes; otherwise, if T contains no inference node, then it contains free nodes and since every free node lies above another free node, we can construct a complete branch all of whose nodes are free for S , contradicting the assumption that T is closed for S .

If \mathcal{R} denotes a system of valid inference rules for clauses, then by $\mathcal{R}(S)$ we denote the union of the set S with the set of all clauses which can be obtained from S by one application of one of the inference rules in \mathcal{R} to clauses in the set S . Setting $\mathcal{R}^0(S) = S$ we define $\mathcal{R}^{n+1}(S) = \mathcal{R}(\mathcal{R}^n(S))$.

The following theorem provides the foundation for our use of semantic trees in automatic theorem-proving.

Theorem 2. Let \mathcal{R} be a system of valid inference rules and let there be given a particular way of associating with every unsatisfiable set of clauses S a finite semantic tree T for S such that

- (*) there is an inference node $N \in T$, and for some subset $S' \subseteq S$ of the set of clauses which fail immediately below N there is a clause $C \in \mathcal{R}(S')$ such that C fails at N .

Then $\square \in \mathcal{R}^n(S)$ for some $n \geq 0$, and consequently \mathcal{R} is a complete system of refutation.

Proof: Let S be unsatisfiable, T the semantic tree associated with S . Let n be the number of nodes of T free for S (n is finite since T is finite). If $\square \in S$, then $\square \in \mathcal{R}^0(S)$. Otherwise, by (*), there is an inference node $N \in T$ and a clause $C \in \mathcal{R}(S)$ such that C fails at N . Therefore the number of nodes of T free for $\mathcal{R}(S)$ is less than or equal to $n - 1$. Similarly, since T is a closed semantic tree for $\mathcal{R}^{m-1}(S)$, $m > 1$, (*) applies to $\mathcal{R}^{m-1}(S)$; and consequently the number of nodes of T free for $\mathcal{R}^m(S)$ is less than or equal to $n - m$. No node of T is free for $\mathcal{R}^n(S)$, and therefore the root of T is a failure point for $\mathcal{R}^n(S)$. But then $\square \in \mathcal{R}^n(S)$, for no other clause fails at the root of a semantic tree.
Q.E.D.

Theorem 1 has been used implicitly in the statement and proof of Theorem 2, because $\mathcal{R}^i(S)$ unsatisfiable implies, by Theorem 1, that T is closed, and thus T has an inference node for each $\mathcal{R}^i(S)$, $i \geq 0$.

Deletion strategies

A clause is a *tautology* if it contains complementary literals. A clause C *subsumes* a clause D if it has an instance $C\sigma$ which is a subclause of D (i.e. $C\sigma \subseteq D$). If \mathcal{R} is a system of inference whose completeness can be justified by Theorem 2, then \mathcal{R} remains a complete inference system when we allow in \mathcal{R} the deletion of tautologies and of subsumed clauses.

If C is a tautology then $C\sigma$ contains complementary literals for every σ . But no \mathcal{A}_N contains complementary literals and C cannot fail on any semantic

THEOREM PROVING

tree. If $C\sigma \subseteq D$ and D fails at some node N of a semantic tree, then some ground instance $D\xi$ of D fails at N , but then $C\sigma\xi$ also fails at N . Thus tautologies and subsumed clauses need never occur in a proof of \square in the system \mathcal{R} , for in the proof of Theorem 2 it is clear that only clauses which fail at nodes of the semantic tree T associated with the original unsatisfiable set S need ever occur in such a proof. (If S is any unsatisfiable set of clauses, then certainly S remains unsatisfiable after deleting tautologies and subsumed clauses. However, such a demonstration does not provide a proof of the compatibility of these strategies with a system of inference.)

CLASH TREES

The Latent Clash Rule

All our applications of Theorem 2 will be to inference systems \mathcal{R} which consist of just one rule of inference that is in each case a specialization of Robinson's (1967) latent clash resolution rule. The corresponding tree T associated with an unsatisfiable set S will similarly be a specialization of a clash tree.

If clauses B_1, \dots, B_k fail at the satellite nodes immediately below some inference node N , then we term them *satellite clauses*. If A fails at the corresponding nucleus node, then A is a *nucleus clause*.

The following theorem and its proof provide the general setting for subsequent specializations.

Theorem 3. Let a finite clash tree T be associated with every unsatisfiable set of clauses S (where T depends on S) and let \mathcal{R} consist of the single rule of inference (latent clash resolution):

- (**) From the 'nucleus clause' $A = A_0 \vee D_1 \vee \dots \vee D_m$, and the 'satellite clauses' $B_i = B_{0i} \vee E_i$, $1 \leq i \leq m$, where the complements of the literals in E_i are unifiable with the literals in D_i , and ξ is the most general simultaneous unifier of these sets of literals for all $1 \leq i \leq m$ (the variables occurring in the clauses A, B_1, \dots, B_m being standardized apart),
infer the 'resolvent' $C = A_0\xi \vee B_{01}\xi \vee \dots \vee B_{0m}\xi$.

(Moreover we may insist that the *clash condition* be satisfied, namely that no $E_i\xi$ or complement of $E_i\xi$ occurs in any of the clauses $A\xi, B_1\xi, \dots, B_m\xi$ except in $B_i\xi$ itself and in $A\xi$ as $D_i\xi$).

Then, if any clauses A, B_1, \dots, B_k fail immediately below an inference node N in T , A has the form of the nucleus clause in (**) and corresponding to A we have satellite clauses B_1, \dots, B_m , $m \leq k$, having the form of the satellite clauses in (**) such that the resolvent C in (**) fails at N .

Remarks. (a) Theorems 2 and 3 combine to yield *the completeness of latent clash resolution*; for the conclusion of Theorem 3 satisfies the hypothesis of Theorem 2 and therefore the conclusion of Theorem 2 holds, namely that (**) is complete.

(b) The rule (**) is stated without reference to unifiable partitions and lends itself naturally to a statement in terms of factors. In either case the number

of unifiable partitions or of factors which need be generated is in general less than the total number possible. We shall return to this point after the proof of Theorem 3.

(c) Later we shall specialize in various ways the form of the clash tree T associated with an unsatisfiable set S . The corresponding specializations of (***) and of the proof of Theorem 3 will provide proofs of completeness for these inference systems when combined with Theorem 2.

Proof of Theorem 3. Let T be a clash tree, $N \in T$ an inference node, N_1, \dots, N_k the satellite nodes immediately below N , and N_{k+1} the corresponding nucleus node. Let \mathcal{A}_N be the partial assignment at N , the singleton $\{L_i\}$, where L_i is a ground literal, the set attached to the satellite node N_i , $1 \leq i \leq k$, and $\{L_1, \dots, L_k\}$ the set attached to N_{k+1} . Suppose A fails at N_{k+1} and that B_i fails at N_i , $1 \leq i \leq k$.

First we show that each B_i has the form of a satellite clause in (**). Since B_i fails at N_i but not at N there is a ground instance $B_i\sigma_i$ which is false at N_i but not at N . Thus the complements of the literals in $B_i\sigma_i$ all occur in $\mathcal{A}_N \cup \{L_i\}$ but not in \mathcal{A}_N . So $B_i\sigma_i = B_0\sigma_i \vee L_i$, where $B_i = B_0 \vee E_i$, $E_i\sigma_i = L_i$, and $B_0\sigma_i$ is false in \mathcal{A}_N .

Now, to show that A has the form of a nucleus clause in (***) we note that similarly, as above, A fails at N_{k+1} but not at N . Therefore some ground instance $A\sigma$ is false at N_{k+1} but not at N , and consequently the complements of the literals in $A\sigma$ all occur in $\mathcal{A}_N \cup \{L_1, \dots, L_k\}$ but not in \mathcal{A}_N . Thus $A\sigma = A_0\sigma \vee L_1 \vee \dots \vee L_m$, where for simplicity the nodes N_1, \dots, N_k have been reordered if necessary so that the literals L_i , $1 \leq i \leq m$, which occur in $A\sigma$, will be an initial segment of L_1, \dots, L_k . Thus $A = A_0 \vee D_1 \vee \dots \vee D_m$, where $A_0\sigma$ is false in \mathcal{A}_N and $D_i\sigma = L_i$.

It only remains now to show that the inferred clause C fails at N and that the clash condition may be imposed upon the clash rule. We have already shown that the clause $C' = A_0\sigma \vee B_0\sigma_1 \vee \dots \vee B_0\sigma_m$ fails at N since each of $A_0\sigma$ and $B_0\sigma_i$ are false in \mathcal{A}_N . We shall show that C fails at N by showing that C' is an instance of C . But because ξ is the most general unifier which transforms all of the literals 'resolved upon' in the inference into single literals, and because A, B_1, \dots, B_m have been standardized apart, there is a substitution λ such that $C' = C\lambda$, and therefore C fails at N .

Suppose that the clash condition is violated and that some $E_i\xi$ or complement of $E_i\xi$ occurs in C . Then $E_i\xi\lambda = L_i$ or $\overline{E_i\xi\lambda} = \overline{L_i}$ occurs in C' which fails at \mathcal{A}_N . But then $\overline{L_i}$ or L_i would have to occur already in \mathcal{A}_N , and this is impossible. The only other possibility is for $E_i\xi$ or its complement to be identical to $E_j\xi$ or $D_j\xi$ for $j \neq i$. But then L_i would be identical to L_j or $\overline{L_i}$ to $\overline{L_j}$ for $j \neq i$, which is likewise impossible. Q.E.D.

Factoring

When applying the latent clash rule (***) or some specialization of it to prove a set of clauses unsatisfiable, a single clause will normally occur as a premiss

THEOREM PROVING

of an application of the rule many times. To avoid the duplication involved in repeatedly unifying the same groups of literals within a clause we may employ the device of factoring. The single most general simultaneous unifier (m.g.s.u.) ξ of (**), can be decomposed into a sequence of components $\xi_1, \dots, \xi_m, \xi_{m+1}$, and ξ' such that ξ_i is the m.g.u. of E_i , $1 \leq i \leq m$, ξ_{m+1} the m.g.s.u. of D_1, \dots, D_m , and ξ' the m.g.s.u. of $\overline{E_i \xi_i}$ with $\overline{D_i \xi_{m+1}}$, $1 \leq i \leq m$. Then in (**) the resolvent $C = A_0 \xi \vee B_{01} \xi \vee \dots \vee B_{0m} \xi$ equals $(A_0 \xi_{m+1} \vee B_{01} \xi_1 \vee \dots \vee B_{0m} \xi_m) \xi'$, where the unifiers ξ_i , $1 \leq i \leq m+1$, perform the necessary unifications within a single clause and ξ' only mates simultaneously single literals in the satellite clauses with the corresponding single literals in the nucleus clause. The unifier ξ' must be constructed separately for each application of the inference rule; but the unifiers ξ_i need only be constructed once when a clause is first produced, and this same substitution may be associated with its clause whenever the literals E_i , in case $1 \leq i \leq m$, or the literals D_1, \dots, D_m , in case $i = m+1$, are the literals unified and resolved upon in an inference.

These considerations motivate replacing (**) by two independent operations, factoring and the resolution of factored clauses. A *factor* of a clause C is a clause $C\theta$, where θ is the m.g.u. of a single subset of literals in C in case $C\theta$ is to be used as a satellite clause, or a clause $C\theta$ where θ is the m.g.s.u. of subsets D_1, \dots, D_m of literals in C in case $C\theta$ is to be used as a nucleus clause. In addition we require that the literals in $C\theta$ which have been deliberately unified by θ be somehow *distinguished* from those which have not (this may be accomplished for programming purposes, for instance, by storing $C\theta$ with its distinguished literal or literals occurring first in $C\theta$ and separated in some way from those literals which follow and are not distinguished). The operation of factoring then consists of replacing each clause C which is not a factor by the set of all its factors. The clash rule for factored clauses then amounts to resolving a clash on its distinguished literals.

The notion of factoring defined above is an improvement on the notion one obtains by straightforward translation of unifiable partitions into terms of factors. Firstly, only the literals which are to be resolved upon are deliberately unified in a factor. Conversely, only literals deliberately unified need be resolved upon. For the case of binary resolution this version of factoring is equivalent to Robinson's (1965a) notion of 'key triple'. The remarks about factoring above are however completely general and apply to any inference rule which is a specialization of latent clash resolution.

Binary resolution and A-ordering

Given a set of clauses S we define an *A-ordering* for S to be a total ordering \leq defined on some subset of the set of literals $\{L\sigma \mid L \in C \text{ for some clause } C \in S\}$ such that

- (i) if $L_1 < L_2$ then $L_1\sigma < L_2\sigma$ for all σ ;
- (ii) if L_1 and L_2 are alphabetic variants or complements then $L_1 \leq L_2$ and $L_2 \leq L_1$.

This definition is similar to Slagle's (1967) but has the advantage of allowing a finer discrimination between literals.

Now let S be an unsatisfiable set of clauses and $K \subseteq \bar{H}$ a finite subset such that any semantic tree for K is closed. Let \leq be an A-ordering for S and $\{A_1, \dots, A_n\} = K$ be an enumeration of K compatible with \leq ; i.e., if $A_i < A_j$ then $i < j$. We associate with S the simple binary tree T for K obtained by attaching ϕ to the root of T , the sets $\{A_1\}$ and $\{\bar{A}_1\}$ immediately below the root and the sets $\{A_{i+1}\}$ and $\{\bar{A}_{i+1}\}$ immediately below any node to which $\{A_i\}$ or $\{\bar{A}_i\}$ has been attached.

Referring to the proof of Theorem 3 and using the notion of factor, we see that if a clause C fails at a failure point $N \in T$, then the distinguished literal L_1 of some factor $C\theta$ fails properly at N , while the remaining literals in $C\theta$ fail at nodes above N . Since the enumeration $\{A_1, \dots, A_n\}$ is compatible with \leq it follows that $L_1 < L_2$ for no $L_2 \in C\sigma$, where $L_1 \neq L_2$. For otherwise, if $L_1 < L_2$ and $(C\theta)\sigma$ is the ground instance of $C\theta$ which is false at N , then $L_1\sigma < L_2\sigma$ by (i) and yet, by the construction of T , $L_2\sigma$ is A_i or \bar{A}_i , and $L_1\sigma$ is A_j or \bar{A}_j , where $i < j$; so by (ii) $L_1\sigma < L_2\sigma$ cannot occur.

Taking into consideration the remarks above, specializing Theorem 3 appropriately and applying Theorem 2 we obtain completeness of the following version of binary resolution:

Given a set of clauses S and an A-ordering \leq for S , infer from the factors $L_1 \vee A_0$ and $L'_1 \vee B_0$ with distinguished literals L_1 and L'_1 respectively, the clause $(A_0 \vee B_0)\xi$ where ξ is the m.g.u. of L_1 and L'_1 , and where neither $L_1 < L_2$ nor $L'_1 < L_2$ for any literal L_2 in either A_0 or B_0 .

As an example of the use of an A-ordering in conjunction with this rule, let the A-ordering \leq be determined for some set of clauses by the conditions $P(f(x))\sigma < P(g(y))\sigma$ and $P(x)\sigma < Q(y)\sigma$ for all σ . Then the unfactored clause $C = P(g(a)) \vee P(f(x)) \vee Q(b)$ has only one factor $C\theta$, where θ is the identity substitution and $P(f(x))$ is the distinguished literal. In this case the A-ordering \leq has eliminated the need to consider two of the three possible factors. If $C = Q(f(a)) \vee P(x) \vee P(f(a))$ then there are three factors of C compatible with \leq and only one of the four possible factors need not be generated.

The following example shows that the rule above is compatible with neither set of support (Wos, Carson and Robinson 1965) nor P_1 -deduction (Robinson 1965b): let S be the set $\{L_1 \vee L_2, L_1 \vee \bar{L}_2, \bar{L}_1 \vee L_2, \bar{L}_1 \vee \bar{L}_2\}$, and let \leq be determined by $L_1 < L_2$. Then, although S is unsatisfiable, \square can be deduced with neither P_1 -deduction nor with set of support if we take $\{L_1 \vee L_2\}$ as the set of support.

In the next sections we shall see that a weaker version of the A-ordering restriction applies to M-clashes, and that in the particular case of P_p -deduction (P_1 -deduction with renaming) a more restrictive ordering principle based on A-ordering is complete.

M-clashes

We have been unable to construct binary semantic trees to justify either the set of support strategy or P_1 -deduction. However, the M-clash trees which we introduce below can be used to prove completeness of M-clashes, and by suitably choosing the interpretation M and by decomposing the corresponding M-clash rules we obtain, following Slagle (1967), the completeness of these inference systems.

Let S be a set of clauses. Define a *Herbrand interpretation* of S to be any complete assignment to \hat{H} , the Herbrand base of S (we have already seen how any complete assignment to \hat{H} can be regarded as a possible interpretation of S). Assume for the moment that M is a Herbrand interpretation of S . Let S be unsatisfiable and $K \subseteq \hat{H}$ a finite subset such that any semantic tree for K is closed. Let $M = \{A'_1, \dots, A'_n\}$ be M restricted to K , where $A_i \in \hat{H}$ and $A'_i = A_i$ if $A_i \in M$ and $A'_i = \bar{A}_i$ if $\bar{A}_i \in M$, so that $M \subseteq M$ and M is a complete assignment to K . We associate with S the *M-clash tree* T defined as follows:

- (i) ϕ is attached to the root of T ;
- (ii) the root of T has $n + 1$ immediate descendants with $\{A'_i\}$ assigned to the i th satellite node, $1 \leq i \leq n$, and $\{\bar{A}'_1, \dots, \bar{A}'_n\}$ to the nucleus node;
- (iii) let $N \in T$, \mathcal{A}_N not a complete assignment to K , $\mathcal{A}_N \subseteq M$ and $M - \mathcal{A}_N = \{A'_{j_1}, \dots, A'_{j_k}\}$; then N has $k + 1$ immediate descendants, with the singletons $\{A'_{j_1}\}, \dots, \{A'_{j_k}\}$ attached to the k satellite nodes and the set $\{\bar{A}'_{j_1}, \dots, \bar{A}'_{j_k}\}$ to the nucleus node.

Note that the assignment at any nucleus node is a complete assignment to K and therefore every nucleus node of T is a tip of T . Note, too, that the assignment at any satellite node is always a subset of M , and that for any such assignment containing $m \leq n$ literals there is a total of exactly $m!$ satellite nodes with the same assignment.

Suppose the clause B fails at a satellite node which is a failure point. Then some ground instance $B\sigma$ of B is false in M and therefore in M . Thus B itself is false in the Herbrand interpretation M .

Suppose A fails at a failure point N which is a nucleus node. Some ground instance $A\sigma$ of A must fail at N and some literals $A'_{j_1}, \dots, A'_{j_m}$, for some $m \leq k$, must fail properly at N . But since these literals belong to $M \subseteq M$ and therefore are true in M it follows that $A\sigma$ is true in M . Thus A has an instance which is true in M .

Note that, since the resolvent C of a clash fails at a satellite node, C must be false in M . Note also that a nucleus clause is never a resolvent and therefore must belong to the original set of clauses S .

Specializing Theorem 3 to the M-clash tree, keeping in mind the considerations above and applying Theorem 2, we obtain the completeness of M-clash resolution:

Given a set of clauses S and a Herbrand interpretation M of the Herbrand base H of S , from a factored nucleus clause $A = L'_1 \vee \dots \vee L'_m \vee A_0$ and factored satellites $B_i = \bar{L}_i \vee B_{0i}$, $1 \leq i \leq m$, where ξ is the most general unifier such that $L_i \xi = L'_i \xi$ simultaneously for all $1 \leq i \leq m$,

infer the resolvent $C = (A_0 \vee B_{01} \vee \dots \vee B_{0m})$,

when the clash conditions are satisfied; here each satellite B_i is false in M , C is false in M , and A has an instance true in M .

Remarks. (a) Slagle (1967) has remarked that if $T \subseteq S$ and $T - S$ is satisfiable then it is satisfied by a Herbrand interpretation M . It follows that no clause in $T - S$ can be a satellite of an M -clash. Decomposing the resulting M -clashes into sequences of binary resolutions we see that no two clauses are resolved which both come from $T - S$. But this is just the defining condition of the set of support strategy.

(b) Complications arise if we wish to use an interpretation M explicitly when in an application of the M -clash rule we need to decide the truth or falsity of clauses and their instances. Firstly, if M is not a Herbrand interpretation then we must extend M to an interpretation in which all the Skolem functions which actually occur in S are defined in some way. To M extended in this way there will then correspond a Herbrand interpretation which will justify the use of this extended M . Otherwise questions of truth or falsity for clauses whose vocabularies are not fully interpreted in M are literally meaningless.

A much more serious restriction on the explicit use of an interpretation M is that it actually admit of an algorithm for deciding truth and falsity of clauses and their instances. Otherwise there is no way in which its use can be mechanized for a computer. Interpretations containing only a finite number of elements are effective in this sense. But unless they possess other special properties the exhaustive instantiation of each clause over the domain of the interpretation for the purpose of testing it for false or true instances is likely to be prohibitive. These same considerations apply to model partitions (Luckham 1968), which can be justified as a special case of M -clashes.

(c) Slagle has also shown that hyper-resolution may be regarded as the special case of M -clashes where all instances of positive literals are regarded as false in M and all instances of negative literals as true. Then all satellites and resolvents contain only positive literals, and all the negative literals of the nucleus clause are resolved upon in the clash. The resulting clash is maximal in the sense that no subclash need ever be generated. This highly desirable property can be extended by the device of renaming. We shall show in a later section how advantage can be taken of maximality to yield a particularly efficient version of P_1 -deduction.

AM-clashes

Let \leq be an A-ordering and M a Herbrand interpretation for a set of clauses – then the following ordering principle may be imposed upon the M -clash rule:

H

THEOREM PROVING

Given the factored nucleus $A = L'_1 \vee \dots \vee L'_m \vee A_0$ and the factored satellites $B_i = L_i \vee B_{0i}$, $1 \leq i \leq m$, with resolvent $C = (A_0 \vee B_{01} \vee \dots \vee B_{0m})\xi$ satisfying the M-clash conditions, we may insist that for no literal $L_i \in B_{0i}$ do we have $L_i < L'$ for any $1 \leq i \leq m$.

This restriction improves upon Slagle's ordering principle, for the notion of A-ordering defined above is generally more restrictive. The proof that follows of the compatibility of the ordering principle above with the M-clash rule is essentially an adaptation of Slagle's argument.

Given a closed M-clash tree T and an A-ordering \leq for the unsatisfiable set S , we shall show that there is in T an inference node N such that the factored clauses A and B_i , $1 \leq i \leq m$, which fail below the inference node and their resolvent C , satisfy the AM-clash conditions. It will then follow by Theorem 2 that the AM-clash rule is complete. We note that it is in fact only necessary to show that the satellites B_i satisfy the ordering principle above since we have already seen that the M-clash conditions are satisfied.

As before let $M \subseteq M$ be defined as the set $\{A'_1, \dots, A'_n\}$, where the ordering of the A'_i is compatible with the A-ordering \leq , i.e. if $A'_i < A'_j$, then $i < j$. We construct a subset M' of M , as follows:

- (i) $M'_0 = \phi$.
- (ii) If some factored clause $B_{i+1} = L_{i+1} \vee B_{0\ i+1}$ of some clause in S has a ground instance $B_{i+1}\sigma$ false in $\{A'_{i+1}\} \cup M'_i$ but no clause in S fails in M'_i , then $M'_{i+1} = M'_i$; otherwise $M'_{i+1} = \{A'_{i+1}\} \cup M'_i$. In the former case we may choose the factor B_{i+1} so that $L_{i+1}\sigma = A'_{i+1}$, and we say that B_{i+1} is associated with A'_{i+1} .
- (iii) $M' = M'_n$.

M' is a partial assignment to K and there is some node N such that $\mathcal{A}_N = M'$. We claim that N is an inference node. Note that the factor B_i associated with A'_i fails at the satellite node N_i immediately below N . Some nucleus clause fails at the nucleus node and no clause in S fails at N , by the construction of M' . A resolvent C from the nucleus clause and from some subset of the satellite clauses fails at N .

Suppose now that the A-ordering restriction is violated and that therefore for some i the distinguished literal L_i in the factor $B_i = L_i \vee B_{0i}$ associated with A'_i is less than some literal $L'_i \in B_{0i}$, i.e., $L_i < L'_i$. Then, if $B_i\sigma$ is the ground instance of B_i which fails at N_i , we have $L_i\sigma < L'_i\sigma$. But $L_i\sigma = A'_i$ and $L'_i\sigma = A'_j$ for some $j < i$, and since it is not true that $A'_i < A'_j$ by the compatibility of the enumeration of the A'_i with \leq , it follows that $L_i\sigma < L'_i\sigma$ does not hold; consequently it is not true that $L_i < L'_i$.

Hyper-resolution and P₁-deduction

Until other more efficient and mechanizable applications are found for AM-clash resolution, the two most likely candidates for an efficient proof strategy

seem to be set of support and hyper-resolution both supplemented by factoring and A-ordering. Meltzer (1968) has shown that renaming and P_1 -deduction can often be used to sharpen a set of support strategy. Luckham (1968) has given examples of proofs obtained by what is essentially P_1 -deduction with renaming, and these proofs are no less efficient than those obtained with set of support. The version of P_1 -deduction stated below seems to us a distinct improvement over the existing strategy. Preliminary results obtained by programming this strategy in ATLAS-AUTOCODE on the KDF9 support this view.

Hyper-resolution has the advantage over P_1 -deduction that it avoids generating the $(2^n)!$ resolvents that are produced by resolving in all possible ways among n -factored satellite clauses and a given factored nucleus clause (where each distinguished literal in the nucleus is associated with only one of the satellites). It has the disadvantage of not saving the partial hyper-resolvents that are generated on the way to producing the maximal hyper-resolvent. These partial hyper-resolvents need to be recomputed each time any of them is completed in a distinct way. In addition, the problem of searching for hyper-resolvents is more complicated than the corresponding problem for P_1 -deduction. The following version of P_1 -deduction incorporates the advantages of hyper-resolution over P_1 -deduction without suffering from its disadvantages.

Given a set of clauses S and an A-ordering \leq and after a renaming (if desired),

- (i) replace each non-positive clause in S by the set of its factors (a non-positive clause is factored as a nucleus clause). Choose any total ordering of the distinguished literals in such a factor A (the ordering may be chosen independently for each A). Let $A = L_1 \vee \dots \vee L_n \vee A_0$, where we agree to write negative literals L_i in order and before positive literals and where A_0 is the positive subclause of A .
- (ii) replace each positive clause in S and, later, each positive resolvent by the set of its factors (positive clauses are factored as satellite clauses). We may insist that the A-ordering restriction is satisfied for each such factor.
- (iii) resolve positive factors on their distinguished literal against the *first* negative literal of non-positive factors. If the resolvent is negative it is not factored, but a new ordering of its distinguished literals may be chosen if desired. If a resolvent is positive it is factored as in (ii) above.

It is easily seen that every hyper-resolvent is obtainable exactly once by a sequence of resolutions satisfying restriction (iii). It follows that this inference system is complete and that it satisfies the properties claimed above.

THEOREM PROVING

Condition (i) may be improved and replaced by

(i') replace each non-positive clause A by its factor $A\theta = A$; where θ is the identity substitution. Choose any total ordering of the distinguished literals . . . , etc.

Condition (i') states in effect that non-positive clauses are not factored at all. The proof that completeness is preserved when (i) is replaced by (i') is somewhat complicated and does not lie within the scope of this paper.

Darlington (1969) shows how to exploit renaming, A-ordering, and the ordering of negative literals in non-positive clauses to avoid performing most of the resolutions excluded by set of support. He does this for the case of applications of theorem-proving to large-scale information retrieval systems where a set of support strategy seems to be highly desirable.

Other applications of semantic trees

The notion of semantic trees employed in this paper can easily be extended to the predicate calculus with equality. Indeed, Robinson's (1968) original formulation of the semantic tree construction was for this logic. None the less, we have been unable to find any binary semantic trees which yield reasonably mechanizable inference systems. It is easy to show that assignment trees (Sibert 1967) can be constructed as semantic clash trees. In this case, by exploiting the generalized notion of failure, it has been possible to impose additional restrictions on the generation of unifiable partitions. However, the basic system of three inference rules corresponding to inference nodes remains essentially that of Sibert's thesis. We are pessimistic about the possibilities of finding other semantic tree constructions which yield efficient inference systems for the predicate calculus with equality.

Hayes (1969) has applied the semantic tree method to obtain a simple mechanizable inference system for J. McCarthy's three-valued predicate calculus.

Acknowledgements

This work was supported by a grant to Dr Meltzer from the Science Research Council (Hayes) and by an I.B.M. fellowship and grant from Imperial College (Kowalski). The authors wish to thank Miss Isobel Smith for helpful discussion and Dr Meltzer for unflagging encouragement and guidance.

REFERENCES

- Darlington, J. (1969) Theorem proving and information retrieval. *Machine Intelligence 4*, pp. 173-81 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.
- Hayes, P. J. (1969) A machine-oriented formulation of the Extended Functional Calculus. *A.I. Memo*. Stanford University: Artificial Intelligence Project. (To appear.)
- Luckham, D. C. (1968) Some tree-parsing strategies for theorem-proving. *Machine Intelligence 3*, pp. 95-112 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

- Meltzer, B. (1966) Theorem-proving for computers: some results on resolution and renaming. *Comput. J.*, **8**, 341-3.
- Meltzer, B. (1968) Some notes on resolution strategies. *Machine Intelligence 3*, pp. 71-5 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
- Robinson, J. A. (1965a) A machine-oriented logic based on the resolution principle. *J. Ass. comput. Mach.*, **12**, 23-41.
- Robinson, J. A. (1965b) Automatic deduction with hyper-resolution. *Int. J. comput. Math.*, **1**, 227-34.
- Robinson, J. A. (1967) A review of automatic theorem-proving. *Annual symposia in applied mathematics XIX*. Providence, Rhode Island: American Mathematical Society.
- Robinson, J. A. (1968) The generalized resolution principle. *Machine Intelligence 3*, pp. 77-93 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
- Sibert, E. E. (1967) A machine-oriented logic incorporating the equality relation. Ph.D. thesis. Rice University.
- Slagle, J. R. (1967) Automatic theorem-proving with renamable and semantic resolution. *J. Ass. comput. Mach.*, **14**, 687-97.
- Wos, L. T., Carson, D. F. & Robinson, G. A. (1965) Efficiency and completeness of the set of support strategy in theorem-proving. *J. Ass. comput. Mach.*, **12**, 536-41.



A Machine-Oriented Logic incorporating the Equality Relation

E. E. Sibert, Jnr.

Rice University

1. INTRODUCTION

A major source of inefficiency in complete proof procedures for first-order logic has been the lack of a satisfactory way of handling the equality relation (Wang 1965, Robinson 1967). In almost all methods previously studied it has been necessary to treat equality as simply another binary relation, and to introduce a number of axioms so that this relation has the desired properties. Such an approach not only lengthens the statement of the problem, but also tends to cause the generation of numerous redundant inferences.

Here we present a system of first-order logic which is machine-oriented, in the sense that single inferences usually require a substantial amount of information processing, and which is specifically intended to provide a theoretical basis for proof procedures capable of handling equality in an efficient manner.

As has commonly been the case with machine-oriented formalisms, the system is organized around the concepts of unsatisfiability and refutation, instead of validity and proof; and the sentence to be refuted is taken to be in prenex form, with no existential quantifiers in the prefix, and the matrix in conjunctive normal form.

The plan of the paper is as follows: section 2 is devoted to an account of the syntax and semantics used, and to a discussion of Herbrand's theorem in this formalism. In the third and fourth sections the three rules of inference are described, and the main completeness theorem is established. Section 3 discusses the case in which no variables are present, and section 4 treats the general case.

Section 5 develops an improved form of one of the rules of inference, making use of a recent result on resolution given by J. R. Slagle. Section 6 treats some devices for increasing the efficiency of refutation procedures, concluding with a discussion of the subsumption principle, a technique for detecting superfluous clauses, and a thorough analysis of its

THEOREM PROVING

relation to completeness. An example is presented which illustrates the application of these results.

2. FORMAL PRELIMINARIES

The syntax used here is an extension of that described in Robinson (1965a), with some additional notation for equality. In order to keep the presentation reasonably self-contained, we give here a brief description of that syntax, borrowing freely from Robinson's account.

We assume an infinite supply of *variables*, *function* symbols of each degree, and *predicate* symbols of each degree. The usual conventions are used in the choice of letters for these symbols. We use one logical symbol, that of *negation*: \sim . An ordering, called the *alphabetical order*, is supposed to well-order the collection of symbols, with the negation symbol preceded by all the others.

Terms. A variable is a term, and a string of symbols consisting of a function symbol of degree $n \geq 0$ followed by n terms is a term.

Atomic formulae. A string of symbols consisting of a predicate letter of degree $n \geq 0$ followed by n terms is an atomic formula (*atom*).

Literals. An atomic formula is a literal; and if A is an atomic formula then $\sim A$ is a literal.

Clauses. A finite set (possibly empty) of literals is termed a clause. The empty clause is denoted by: \square .

Ground literals. A literal which contains no variable is termed a ground literal.

Ground clauses. A clause containing only ground literals is a ground clause. In particular, \square is a ground clause.

The set of all terms and literals is well-ordered in the *lexical order* by the rule that A precedes B iff A is shorter than B or, if A and B have the same length, then A has the alphabetically earlier symbol in the first position at which A and B differ.

When writing out terms and literals we shall use parentheses in the customary fashion, so it will not be necessary to show explicitly the degrees of the function and predicate symbols.

The following definitions concerning instantiation, while not actually part of the syntax of our logical system, may conveniently be given here.

A *substitution component* is any expression of the form T/V , where V is a variable and T is any term different from V . V is called the *variable of the component* T/V , and T is called the *term of* T/V . A *substitution* is a finite set (possibly empty) of substitution components, no two of which have the same variable. The *empty substitution* is denoted by ϵ .

If E is a finite string of symbols and θ is a substitution, then the *instance* of E by θ (denoted by $E\theta$) is the string obtained by replacing each occurrence in E of a variable which is the variable of a component in θ by an occurrence of the term of that component. If C is a set of strings, we denote by $C\theta$ the set $\{E\theta : E \in C\}$.

The *composition* of two substitutions, $\theta = \{T_1/V_1, \dots, T_k/V_k\}$ and λ , is the substitution $\theta' \cup \lambda'$, where λ' is the set of all components of λ whose variables are not among V_1, \dots, V_k , and $\theta' = \{T_i\lambda/V_i: i = 1, \dots, k\}$. It is shown in Robinson (1965a) that, for any substitutions $\lambda, \theta, \mu, \varepsilon \theta = \theta\varepsilon = \theta; (\lambda\theta)\mu = \lambda(\theta\mu)$; and, if E is any string, then $E(\lambda\theta) = (E\lambda)\theta$.

We now return to the development of the syntax. As special forms for equality we allow two additional kinds of literals. First, we allow *equations* having the form

$$= \{\alpha, \beta\} \quad (2.1)$$

$$\text{or} \quad = \{\alpha\}, \quad (2.2)$$

where α and β are any terms. Second, we allow *inequalities* having the form

$$\neq \{\alpha_1, \dots, \alpha_n\}, \quad n \geq 1, \quad (2.3)$$

where $\alpha_1, \dots, \alpha_n$ are any terms. The negation symbol is never used with either of these forms.

Expressions of the form (2.1) will be interpreted as the equation

$$\alpha = \beta, \quad (2.4)$$

and we shall commonly use this more usual-form to represent (2.1). The 'labelled set' notation is introduced because it emphasizes that the order in which α and β are written in (2.1) is immaterial. (2.2) is equivalent to the trivial equation $\alpha = \alpha$.

The inequality (2.3) will be interpreted as ' $\alpha_1, \dots, \alpha_n$ are not all equal'. As for equations, when $n = 2$ we shall commonly write $\alpha_1 \neq \alpha_2$ instead of (2.3). The case $n = 1$ represents the trivial inequality $\alpha_1 \neq \alpha_1$.

We shall not use literals of the form $= \{\alpha_1, \dots, \alpha_n\}$ even though it would seem a natural extension of the notation already adopted. The reason for this will appear later.

The special expressions for equality do not fall within the Polish syntax described in Robinson (1965a), and we shall not consider that either $=$ or \neq is a predicate letter. To distinguish the two sorts of literals in our syntax, we shall refer to literals formed in the usual way with predicate letters as *type I literals*, equations and inequalities will be termed *type II literals*.

We shall denote the set appearing in a type II literal L by L^s . If, for example, L is the literal $= \{\alpha, \beta\}$ then L^s is $\{\alpha, \beta\}$, and if M is $\neq \{\alpha, \beta, \gamma\}$ then M^s is $\{\alpha, \beta, \gamma\}$.

Let S be a finite set of clauses, and let T be a set of ground terms such that

- (a) If $T \in T$ and T' is a subterm of T then $T' \in T$.
- (b) If $T_1, \dots, T_n \in T$ ($n \geq 0$) and F is any n -ary function symbol appearing in S then $F(T_1, \dots, T_n) \in T$.
- (c) T contains at least one constant (function symbol of degree 0).

We shall be particularly interested in the case that T is the Herbrand universe

THEOREM PROVING

of S , i.e., the set of all terms that can be formed by using the function letters appearing in S (and the constant a if S has no constants).

Let U denote the set of all ground atoms which are instances over T of type 1 atoms occurring in S , and let E denote the set of all (ground) equations and inequalities which can be formed with the terms in T .

Note that if $A, B \in T \cup U$ and A is a proper subexpression of B , then A has fewer letters than B , so that A precedes B in the lexical ordering. Henceforth, if we refer to an ordering of T or $T \cup U$ the lexical ordering is always understood.

The set of all clauses which are instances of clauses in S by terms of T will be denoted by $T(S)$. It is obvious that $T(S) \subset U \cup \sim U \cup E$.

Ordinarily, an interpretation of a sentence S is just an arbitrary assignment of truth-values to the ground atoms arising from S . In the present system something more elaborate is required. An interpretation should specify the equality relation on the terms and assign truth-values to the atoms in a manner consistent with the equality relation it specifies. Accordingly, we make the following definition:

By an *interpretation* (over T) of S we mean a function

$$V: T \cup U \cup E \rightarrow T \cup \{True, False\}$$

with the following properties:

(0) $V(T) \subset T$, $V(U \cup E) \subset \{True, False\}$.

(i) If $T_j \in T$ and $V(T_j) = T_k$, then T_k does not succeed T_j (in the lexical ordering).

(ii) If X is an n -ary predicate or function letter, $\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_n$ are terms in T , and

$$V(\alpha_i) = V(\beta_i), \quad i = 1, \dots, n,$$

then

$$V(X(\alpha_1, \dots, \alpha_n)) = V(X(\beta_1, \dots, \beta_n)),$$

provided that $X(\alpha_1, \dots, \alpha_n)$ and $X(\beta_1, \dots, \beta_n)$ occur in $T \cup U$.

(iii) If $\alpha, \beta \in T$,

$$V(= \{\alpha, \beta\}) = \begin{cases} True & \text{if } V(\alpha) = V(\beta), \\ False & \text{otherwise.} \end{cases}$$

If $\alpha_1, \dots, \alpha_n \in T$, $n \geq 1$,

$$V(\neq \{\alpha_1, \dots, \alpha_n\}) = \begin{cases} True & \text{if } V(\alpha_j) \neq V(\alpha_k), \\ & \text{some } j, k, \\ False & \text{otherwise.} \end{cases}$$

The function V will also be referred to as an *assignment (function)*, and $V(x)$ will be termed the *value* of x . We define V for literals of the form $\sim L$, $L \in U$, by

(iv) $V(\sim L) = \sim V(L)$.

The semantics here is obvious. The function V specifies which terms are 'equal' by giving the same value to equal terms. Property (ii) is simply the

universal substitution property of equality, and (iii) ensures that statements about equality have their normal meaning. In particular, trivial equations are always true, and trivial inequalities are always false. (iv), of course, is just the usual meaning of negation.

Note that the definition of V on T completely determines the values of V on E , independently of the values of V on U .

We shall sometimes represent a list of terms $\alpha_1, \dots, \alpha_n$ by the symbol $\vec{\alpha}$, and write $X(\vec{\alpha})$ for $X(\alpha_1, \dots, \alpha_n)$, where it is always understood that if X is an n -ary symbol, then $\vec{\alpha}$ has n entries. In this context ' $V(\vec{\alpha}) = V(\vec{\beta})$ ' means that $\vec{\alpha}$ and $\vec{\beta}$ have the same number of entries, say n , and that

$$V(\alpha_i) = V(\beta_i), \quad i = 1, \dots, n.$$

We shall also use this notation to denote sets of literals, e.g. $\{\vec{\alpha} \neq \vec{\beta}\}$ denotes

$$\{\alpha_1 \neq \beta_1, \dots, \alpha_n \neq \beta_n\}.$$

An interpretation V is said to *satisfy* a ground clause $C \subset U \cup \sim U \cup E$ iff at least one literal of C has the value *True* under V , otherwise V is said to *falsify* C . V is said to satisfy the system S iff V satisfies every ground clause which is an instance of a clause in S by terms in T . If there is no choice of the set T and interpretation V which satisfies S , then we say that S is *unsatisfiable*; if there is, we say S is *satisfiable*.

If $L \in U \cup \sim U$ then, for any interpretation V , exactly one of $L, \sim L$ is true under V . Thus any clause which contains such a 'complementary pair' of literals is satisfied by any interpretation (i.e., every ground instance of it is satisfied by any interpretation).

Suppose that L is a ground equation and M a ground inequality such that $L^s \subset M^s$. If, for some interpretation V , $V(L) = \text{False}$ then L has the form $\alpha = \beta$, where $V(\alpha) \neq V(\beta)$. But then M has the form $\neq \{\alpha, \beta, \dots, \gamma\}$, so $V(M) = \text{True}$. Thus at least one of L, M is true in any interpretation. This fact motivates the following definition.

Two literals of type II, L and M , are said to be *complementary* iff one of them, say L , is an equation, the other (M) is an inequality, and $L^s \subset M^s$.

Complementarity for type II literals as thus defined is a weaker notion than for type I literals, but, what is most important, we still have the property that any clause which contains a complementary pair of literals is satisfied by every interpretation.

We now state Herbrand's Theorem for the logical system just described. Let T_H be the Herbrand Universe of S , and $T_H(S)$ denote the set of all ground instances of clauses in S by terms in T_H . *Herbrand's Theorem*: If S is unsatisfiable then there is a finite subset of $T_H(S)$ which is unsatisfiable, and conversely.

The argument given in Robinson (1967) is readily adapted to the present case.

3. GROUND REFUTATIONS

We shall now formulate the ground forms of the rules of inference and establish the completeness of these rules; but first we need to define a restricted type of interpretation suitable for use with ground systems and derive its relation to the general interpretations defined earlier.

Let S be a finite set of ground clauses. Let

$$T = \{T_1, \dots, T_n\}$$

be all of the terms appearing in S (including those appearing only as sub-expressions), ordered as before. Let E denote the set of all equations and inequalities involving only terms in T , and denote by U the collection of all type 1 atoms which appear (or whose negations appear) in S . The sets T, E, U are, of course, all finite.

By a ground interpretation of S we mean a function V defined just as before, but on the restricted sets T, E, U , and satisfying the requirements for general interpretations.

Now let S' be some set of clauses (with variables), and let T' be a set of terms satisfying properties (a), (b), (c) for S' . Let U', E' be the corresponding sets of literals. Suppose that the ground sentence S consists of clauses in $T'(S')$. Clearly $T \subset T', U \subset U', E \subset E'$. Any ground interpretation of S is the restriction of an interpretation of S' to $T \cup U \cup E$. The restriction of an arbitrary interpretation V' of S' to $T \cup U \cup E$ will not, in general, be a ground interpretation of S , for it need not be the case that $V'(T) \subset T$. The following lemma shows, however, that there is a ground interpretation of S which is not essentially different from V' .

Lemma. In the above notation, if V' is any interpretation over T' of S' , then there is a ground interpretation of S , V which coincides with V' on $U \cup E$. Conversely, if V is a ground interpretation of S , there is an interpretation V' over T' which coincides with V on $T \cup U \cup E$.

Proof. Let V' be an interpretation over T' . Define a ground interpretation V of S as follows: For $X \in U \cup E$, $V(X) = V'(X)$. For $T \in T$, let T_1 be the earliest term in T such that $V'(T_1) = V'(T)$, and put $V(T) = T_1$. Clearly, for $T_1 \in T, T_2 \in T$ we have

$$V(T_1) = V(T_2) \text{ iff } V'(T_1) = V'(T_2)$$

and it follows trivially that V is the required interpretation.

The converse follows easily; a straightforward induction argument can be used to extend V to the required interpretation.

From this lemma it follows at once that there is a ground interpretation which satisfies S iff there is a (general) interpretation of S' which satisfies S . Thus, in considering the satisfiability or unsatisfiability of ground systems we may restrict our attention to ground interpretations. In view of this fact, we shall henceforth simply say 'interpretation' instead of 'ground interpretation'.

In the subsequent discussion it will frequently be necessary to consider assignments which are defined on only a part of $T \cup U \cup E$, and it is convenient to adopt some conventions specifying just what subsets may be the domain of such a partial assignment. If an assignment is given for all of T (and hence all of E), it induces a natural partition of the set U , corresponding to the equivalence relation $P(\vec{\alpha}) \cong P(\vec{\beta})$ iff $V(\vec{\alpha}) = V(\vec{\beta})$; and, because of property (ii), any extension of the assignment must give the same value to all members of a class of the partition. Any partial assignment considered here will either be defined on (a) an initial segment of T , together with those literals in E to which it can be extended by (iii), or else on (b) $T \cup E$ and the union of some of the classes of the partition induced on U .

A partial assignment V is said to falsify a clause $C \in S$ iff V is defined and has the value *False* for every literal in C . V is said to satisfy C iff V is defined and has the value *True* for at least one literal of C .

If V is a partial assignment and T is a term for which V is not defined, then V is said to *force* the value of T if, for some function letter F , $T = F(\vec{\alpha})$, $V(\vec{\alpha}) = V(\vec{\beta})$ (V being defined on $\vec{\alpha}$, $\vec{\beta}$), and there is a term $T' = F(\vec{\beta})$ for which V is defined. The significance of this concept is, of course, that if V^* is an extension of V to a set which includes T , then $V^*(T) = V(T')$.

If a partial assignment V defined on $\{T_1, \dots, T_k\}$ forces the value of a later term T_m , it may be the case that there is a clause C which is falsified by every extension of V which is defined at T_m , because of the forced assignment of T_m . One could then say that V already falsified C in such circumstances. However, the definitions we have made specifically exclude any 'look-ahead' of this sort, and we shall not say that such a partial assignment falsifies C . The approach used instead will greatly simplify the ensuing arguments.

Given a finite set S of clauses, form a set S' by (a) deleting from S any clause which contains a trivial equation or which contains a complementary pair of literals, and (b) deleting from the remaining clauses any trivial inequalities. Certainly S is satisfiable iff S' is satisfiable, for every clause deleted in (a) is true in every interpretation, and each literal deleted in (b) is false in every interpretation. Note that in (b) we may obtain the empty clause, in which case S is clearly unsatisfiable.

Henceforth we shall suppose that these reduction processes have been applied to any clauses we consider. A clause is said to be *trivially satisfied* iff it can be eliminated by (a) above.

Since any instance of a trivial equation or inequality is also trivial, any set of clauses (not necessarily ground clauses) may be reduced in this way without affecting satisfiability.

A term α is said to be *maximal* in a clause C iff α is not a proper sub-expression of any term appearing in C . This definition also is not restricted to ground clauses. Note that we do not require that α appear in C .

We are now in a position to state three rules of inference (for ground systems) and establish their completeness. In the following A and B denote

THEOREM PROVING

sets of literals, $\{A \vee B\} = A \cup B$, and, if L is a literal, $\{A \vee L\} = A \cup \{L\}$.

Rule 1. Let P be a predicate letter. Then from

$$\{A \vee P(\vec{\alpha}_1) \vee \dots \vee P(\vec{\alpha}_j)\} \quad (3.1)$$

and $\{B \vee \sim P(\vec{\beta}_1) \vee \dots \vee \sim P(\vec{\beta}_k)\} \quad (3.2)$

infer $\{A \vee B \vee \neq \{\vec{\alpha}_1, \dots, \vec{\alpha}_j, \vec{\beta}_1, \dots, \vec{\beta}_k\}\}. \quad (3.3)$

A and B may also contain literals with predicate letter P . We require that (3.1) and (3.2) be different clauses.

Rules 2 and 3 are to be applied only to clauses consisting entirely of equations and inequalities.

Rule 2. Let

$$A = \{A \vee \alpha = \beta_1 \vee \dots \vee \alpha = \beta_r\}$$

and B_1, \dots, B_r be clauses such that

1. α appears in, and is maximal in, each of the clauses A, B_1, \dots, B_r ;
2. α does not appear in A .

Then from these infer

$$\{A \vee B_1[\beta_1/\alpha] \vee \dots \vee B_r[\beta_r/\alpha]\}. \quad (3.4)$$

The notation $B_i[\beta_i/\alpha]$ means 'replace each occurrence of α in B_i by β_i '. Note that, because of 1 above, (3.4) will contain no term which does not already appear in A, B_1, \dots, B_r . The clauses B_1, \dots, B_r need not all be distinct.

The deduction in this rule is simply that if all of the literals in A are false, then at least one equation $\alpha = \beta_i$ is true, hence $B_i[\beta_i/\alpha]$ must be true. Note that there are (at least) $r!$ ways to apply this rule to A, B_1, \dots, B_r .

Rule 3. Let A be a clause and $F(\vec{\alpha})$ be a term which occurs in A and is maximal in A . (F represents any function letter of positive degree.) Let $F(\vec{\gamma})$ be some other term such that $F(\vec{\alpha})$ is not a subexpression of $F(\vec{\gamma})$. Then infer

$$\{\vec{\alpha} \neq \vec{\gamma} \vee A[F(\vec{\gamma})/F(\vec{\alpha})]\}. \quad (3.5)$$

In applying this rule to the clauses of a ground system S the term $F(\vec{\gamma})$ is to be a term appearing in S .

The maximality restraint imposed in rules 2 and 3 is chosen because it carries over to the case in which variables appear. The results we are about to establish for ground systems could be strengthened slightly by requiring 'lexically largest' terms instead of maximal terms, but this condition could not be retained in the general case.

The soundness of these rules follows readily.

For rule 1: Let V be an interpretation which satisfies (3.1) and (3.2). If V satisfies any literal in $A \cup B$, then V satisfies (3.3). If not, then a literal $P(\vec{\alpha}_j)$ in (3.1) is true in V , and likewise a literal $\sim P(\vec{\beta}_s)$ in (3.2), whence $V(\vec{\alpha}_j) \neq V(\vec{\beta}_s)$. Thus V satisfies $\neq \{\vec{\alpha}_1, \dots, \vec{\beta}_k\}$.

For rule 2: Suppose A, B_1, \dots, B_r are clauses satisfying the requirements of the rule. Let V be an interpretation which satisfies all of these clauses. If every literal in A is false under V , then, for some $i, 1 \leq i \leq r$, we have that $V(\alpha) = V(\beta_i)$, hence $B_i[\beta_i/\alpha]$ is also satisfied by V and V satisfies (3.4).

For rule 3: Let $A, F(\vec{\alpha}), F(\vec{\gamma})$ be a clause and terms as required by the rule, and let V be an interpretation which satisfies A . If $V(\vec{\alpha}) \neq V(\vec{\gamma})$ then (3.5) holds. If not, then by property (ii) for interpretations, $V(F(\vec{\alpha})) = V(F(\vec{\gamma}))$ and V satisfies $A[F(\vec{\gamma})/(\alpha)]$, whence, in any case, V satisfies (3.5).

The following theorem is the crucial part of the completeness argument.

Expansion theorem. Let S be a finite, unsatisfiable set of ground clauses (reduced as above) which does not include \square . Then at least one of the rules 1, 2, 3, when applied to suitable clauses of S , will produce a clause which is not in S and is not trivially satisfied.

Proof. Let the sets T, U, E be defined as before.

Case 1: There is a partial assignment V defined on all of T (and all of E) which does not falsify any clause in S .

Let $\{U_1, \dots, U_R\}$ be the partition of U induced by V . Extend V as follows:

Having defined V on $U_1 \cup \dots \cup U_k$ so that no clause of S is falsified, define $V(X)$ for $X \in U_{k+1}$ to be

True, if no clause of S is thereby falsified; otherwise

False, if no clause of S is falsified by that choice; otherwise do not assign V on U_{k+1} and stop the extension process.

As S was assumed to be unsatisfiable, this extension procedure must halt with V defined on $T \cup E \cup U_1 \cup \dots \cup U_k$, where V does not falsify any clause of S , but any extension of V to U_{k+1} does falsify some clause in S .

Since the value *False* for the atoms in U_{k+1} falsifies some clause, S must contain a clause

$$\{A \vee P(\vec{\alpha}_1) \vee \dots \vee P(\vec{\alpha}_j)\}, \quad (3.6)$$

where V already falsifies all of the literals in A , and $P(\vec{\alpha}_i) \in U_{k+1}, i = 1, \dots, j$.

Likewise, the choice *True* for U_{k+1} can only falsify a clause of the form

$$\{B \vee \sim P(\vec{\beta}_1) \vee \dots \vee \sim P(\vec{\beta}_k)\}, \quad (3.7)$$

where B is also falsified by V and $P(\vec{\beta}_i) \in U_{k+1}$.

Consider the clause

$$\{A \vee B \vee \neq \{\vec{\alpha}_1, \dots, \vec{\alpha}_j, \vec{\beta}_1, \dots, \vec{\beta}_k\}\} \quad (3.8)$$

obtained by applying rule 1 to (3.6) and (3.7). Now all of the literals in $A \cup B$ are false in V , and since the atoms $P(\vec{\alpha}_i)$ and $P(\vec{\beta}_i)$ belong to U_{k+1} , we have

$$V(\vec{\alpha}_1) = V(\vec{\alpha}_2) = \dots = V(\vec{\alpha}) = V_j(\vec{\beta}_1) = \dots = V(\vec{\beta}_k),$$

THEOREM PROVING

so the clause (3.8) is false under V . Since V does not falsify any clause in S , we conclude that (3.8) is not in S , and clearly (3.8) is not trivially satisfied. Case 2: There is no partial assignment defined on all of T which does not falsify some clause in S . Since we have assumed that S has no trivial literals the (only possible) assignment $V(T_1) = T_1$ does not falsify any clauses in S . Consequently there is a non-empty partial assignment V defined, say, on T_1, \dots, T_{n-1} (and by extension for those equations in E involving only terms among T_1, \dots, T_{n-1}) such that no clause in S is falsified by V , but any extension of V to T_n does falsify some clause in S . We distinguish two sub-cases:

Subcase (a): V does not force the value of T_n .

Setting $V(T_n) = T_n$ falsifies only literals $T_n = T_k, k < n$, so S must contain a clause A of the form

$$\{A \vee T_n = \beta_1 \vee \dots \vee T_n = \beta_r\},$$

where A is already falsified by V , and β_1, \dots, β_r are terms preceding T_n . Note that T_n is maximal in A , being the latest term which appears in A , and that, as V falsifies A , T_n does not appear in A .

Now setting $V(T_n) = V(\beta_i)$, for $i = 1, \dots, r$, also falsifies a clause in S ; call it B_i . T_n therefore appears in B_i , and is maximal, for B_i can involve only T_n and earlier terms. Thus we can apply rule 2 to A, B_1, \dots, B_r (with T_n as the key term) to obtain

$$\{A \vee B_1[\beta_1/T_n] \vee \dots \vee B_r[\beta_r/T_n]\}. \quad (3.9)$$

But V must falsify (3.9) since V falsifies A , and the clause B_i is falsified by putting $V(T_n) = V(\beta_i)$. Thus (3.9) is not in S and is not trivially satisfied.

Subcase (b): V forces the value of T_n .

T_n has the form $F(\vec{\alpha})$ and there is an earlier term $F(\vec{\gamma})$ with $V(\vec{\alpha}) = V(\vec{\gamma})$. The only allowable choice $V(T_n) = V(F(\vec{\gamma}))$ falsifies some clause A in S . Since T_n is the latest term that can occur in A , T_n is maximal in A . Also, as $F(\vec{\gamma})$ precedes $F(\vec{\alpha})$, $F(\vec{\alpha})$ is not a subexpression of $F(\vec{\gamma})$. Thus we may apply rule 3 to obtain

$$\{\vec{\alpha} \neq \vec{\gamma} \vee A[F(\vec{\gamma})/F(\vec{\alpha})]\}. \quad (3.10)$$

Now by assumption, $V(\vec{\alpha}) = V(\vec{\gamma})$, and the choice $V(F(\vec{\alpha})) = V(F(\vec{\gamma}))$ falsifies A , hence V falsifies $A[F(\vec{\gamma})/F(\vec{\alpha})]$, hence V falsifies (3.10) so that (3.10) is not in S and is not trivially satisfied. Q.E.D.

Notice that in each case of the proof of the expansion theorem we produced a clause C and a partial interpretation V which falsified C but did not falsify any clause in S . This point will be of considerable importance in section 6.

The motivation for the less natural developments in the earlier discussion appear in the argument just given. The 'large' inequalities were introduced to provide a natural and compact notation for rule 1. Correspondingly large equations were precluded to avoid complications in rule 2. And the restricted

viewpoint on partial interpretations (especially the avoidance of 'look-ahead' for forced terms) was adopted in order to obtain a clear-cut distinction between subcase 2(a) and subcase 2(b).

We are now in a position to define a procedure for determining whether a finite set of ground clauses is unsatisfiable. If S is such a set, let $E(S)$ denote the set S together with all of the clauses that can be deduced from S by application of rules 1-3, with trivial clauses and literals eliminated as usual. $E(S)$ is a finite set of ground clauses, so we may (inductively) define $E^n(S) = E(E^{n-1}(S))$ for all $n > 0$, where we define $E^0(S) = S$. Observe that $E^{n-1}(S) \subset E^n(S)$.

Such a procedure is suggested by the following:

Theorem. Let S be a finite set of ground clauses not containing \square . Then either

1. for some $n > 0$, $\square \in E^n(S)$, in which case S is unsatisfiable, or else
2. none of the sets $E_j(S)$ contains \square , but for some $n > 0$, $E^n(S) = E^{n-1}(S)$, in which case S is satisfiable.

Note that if $E^n(S) = E^{n-1}(S)$ and $j \geq 0$, then $E^{n+j}(S) = E^{n-1}(S)$.

Proof. If 1 holds, certainly S is unsatisfiable, since any interpretation which satisfies S satisfies $E^n(S)$, but no interpretation satisfies \square .

Suppose then that 1 does not hold. Now there are only finitely many literals which may appear in clauses of $E^j(S)$, namely

- (i) type I literals which already appear in S ,
- (ii) equations involving only terms already appearing in S , and
- (iii) inequalities, again, involving only terms already present in S .

From this finite supply of literals only a finite number of clauses can be generated, hence the sequence

$$S, E(S), \dots, E^n(S), \dots \quad (3.11)$$

cannot increase indefinitely, i.e., for some n , $E^n(S) = E^{n-1}(S)$. But $E^{n-1}(S)$ cannot be unsatisfiable, for it does not contain \square , so the expansion theorem implies that if $E^{n-1}(S)$ were unsatisfiable then $E^n(S)$ would contain clauses not already in $E^{n-1}(S)$. Thus $E^{n-1}(S)$ is satisfiable, and, as $S \subset E^{n-1}(S)$, S is satisfiable. Q.E.D.

Thus, if, for a given set S , we compute the sequence (3.11), then in a finite number of steps we either deduce the empty clause or the sequence breaks off, and in any case the question of the satisfiability of S is settled.

4. GENERAL COMPLETENESS THEORY

We now wish to extend the results of the preceding section to the general case, i.e., to systems containing variables. To this end we require some preliminary discussion of unification and the replacement of maximal terms.

THEOREM PROVING

By a formula we mean a term or a literal of type 1 (not one of the equality literals). Let $A = \{A_1, \dots, A_n\}$ be a collection of finite, non-empty sets of formulae. If there is a substitution θ such that $A_1\theta, \dots, A_n\theta$ are all singletons, then θ is said to *unify* A , and A is said to be *unifiable*. The unification algorithm, given in Robinson (1967), can be applied to any such collection. The algorithm determines, first, whether the input collection is unifiable, and, if it is, produces a substitution, the most general unifier (m.g.u.) of the input set. This substitution has the property that if, say, σ is the (m.g.u.) of a unifiable collection A and θ is any substitution which unifies A then there is a substitution λ such that $\theta = \sigma\lambda$.

If A is a disjoint collection and θ unifies A , we say that θ *contracts* A iff there are $A \in A, B \in A$ such that $A \neq B$ and $A\theta = B\theta$.

Lemma (4.1). If G is a clause, θ a substitution, and T is any term such that $T\theta$ is maximal in $G\theta$, then T is maximal in G . (Briefly: the ancestors of a maximal term are maximal.)

Proof. Suppose, to the contrary, that there is a term U in G such that $U = XTY$, where X and Y are some strings of letters, not both empty. Then $U\theta = (X\theta)(T\theta)(Y\theta)$, which contradicts the hypothesis that $T\theta$ is maximal in $G\theta$.

A term T is said to be a *primary* term of a clause C iff T appears in C as an argument of a predicate letter or as a member of an equation or inequality.

As an example, in the clause

$$C = \{P(x, f(y)) \vee \neq \{x, y, f(z)\}\}$$

the terms $x, f(y), f(z)$ are primary and maximal, y is a primary term but not maximal, z appears but is neither maximal nor primary, and x_1 is maximal over C but not a primary term of C .

We also need to establish some elementary lemmas relating ordinary substitutions and the replacement of maximal terms.

Lemma. Let A and B be sets of terms with $A\theta = B$. Let $\alpha\theta = \beta$ where β is maximal over B , and suppose that A contains no term other than α which is carried into β by θ . Then, if γ is any term, $A[\gamma/\alpha]\theta = B[\gamma\theta/\beta]$.

Note. The hypotheses of the lemma do not require that $\alpha \in A$ and $\beta \in B$.

Proof. Since A contains no ancestor of β other than α , we have that $\alpha \in A$ iff $\beta \in B$. If $\alpha \notin A$ and $\beta \notin B$, then $A[\gamma/\alpha] = A$ and $B[\gamma\theta/\beta] = B$, and the conclusion follows at once.

If, on the other hand, $\alpha \in A$ and $\beta \in B$, we have that $T^* \in A[\gamma/\alpha]\theta \Rightarrow$ there exists $T' \in A[\gamma/\beta]$ such that $T'\theta = T^*$. Now either $T' = \gamma$ or $T' \neq \gamma$, but

$$T' = \gamma \Rightarrow T^* = \gamma\theta \Rightarrow T^* \in B[\gamma\theta/\beta], \text{ since } \beta \in B;$$

and $T' \neq \gamma \Rightarrow T' \in A \ \& \ T' \neq \alpha \Rightarrow T^* \in B \ \& \ T^* \neq \beta \Rightarrow T^* \in B[\gamma\theta/\beta]$.

Thus $A[\gamma/\alpha]\theta \subset B[\gamma\theta/\beta]$.

Also, let $T^* \in B[\gamma\theta/\beta]$. Either $T^* = \gamma\theta$ or $T^* \neq \gamma\theta$, and $\alpha \in A$, so $\gamma \in A[\gamma/\alpha]$ whence $\gamma\theta \in A[\gamma/\alpha]\theta$. But

$$T^* \neq \gamma\theta \Rightarrow T^* \in B \ \& \ T^* \neq \beta \Rightarrow \text{there exists } T' \in A \text{ such that}$$

$$T'\theta = T^* \ \& \ T' \neq \alpha \Rightarrow T' \in A[\gamma/\alpha] \Rightarrow T^* \in A[\gamma/\alpha]\theta.$$

Thus $B[\gamma\theta/\beta] \subset A[\gamma/\alpha]\theta$, so $A[\gamma/\alpha]\theta = B[\gamma\theta/\beta]$.

Corollary. A and B in the preceding lemma may be taken to be both equations (or both inequalities). For an equation (inequality) is nothing more than a set with a label, and the labels are not affected by the substitutions.

Corollary. The preceding lemma holds if A and B are taken to be type 1 literals, rather than sets of terms.

Proof. Let T be the i th argument of A and T' be the i th argument of B . By the lemma, $\{T\}[\gamma/\alpha]\theta = \{T'\}[\gamma\theta/\beta]$, and the conclusion follows.

Lemma (4.2). Let A and B be clauses such that $A\theta = B$. Let $\alpha\theta = \beta$ where β is maximal over B , and suppose that A contains no term other than α which is carried into β by θ . Then, if γ is any term, $A[\gamma/\alpha]\theta = B[\gamma\theta/\beta]$.

Proof. $L^* \in A[\gamma/\alpha]\theta \Rightarrow$ there exists $L \in A$ such that

$$L[\gamma/\alpha]\theta = L^* \Rightarrow (L\theta)[\gamma\theta/\beta] \in B[\gamma\theta/\beta].$$

But, from the preceding corollaries,

$$(L\theta)[\gamma\theta/\beta] = L[\gamma/\alpha]\theta = L^*,$$

so $A[\gamma/\alpha]\theta \subset B[\gamma\theta/\beta]$.

$L^* \in B[\gamma\theta/\beta] \Rightarrow$ there exists $L \in A$ such that

$$(L\theta)[\gamma\theta/\beta] = L^* \Rightarrow L[\gamma/\alpha]\theta \in A[\gamma/\alpha]\theta.$$

But $L[\gamma/\alpha]\theta = L\theta[\gamma\theta/\beta] = L^*$,

so $A[\gamma/\alpha]\theta \supset B[\gamma\theta/\beta]$.

Thus $A[\gamma/\alpha]\theta = B[\gamma\theta/\beta]$.

We shall now carry out the 'lifting' of rules 1–3 for ground clauses so as to state more general rules which apply also to clauses with variables. For each ground rule we shall formulate a corresponding general rule (called by the same name) and justify the general rule by establishing that *if, say, C' is deduced by the application of the ground rule to reduced instances of clauses C_1, \dots, C_r , then C' is a reduced instance of a clause C which may be deduced by applying the general rule to C_1, \dots, C_r* . The correspondence between the general rule and the ground rule is shown diagrammatically in figure 1, where the dashed arrows indicate instantiation followed by reduction, and the solid arrows indicate application of the rule.

The soundness of the rules about to be presented follows at once, for each rule consists of applying the inference made in the ground rule to suitable instances of clauses already present.

Rules 1 and 2 are stated to be applicable only to input clauses which have no variables in common. In general this restriction is met by standardizing the input clauses apart.

THEOREM PROVING

Again, we shall suppose that none of the input clauses is trivially satisfied and that trivial inequalities have been removed. We denote by $\mathcal{R}(C)$ the clause C with trivial inequalities deleted.

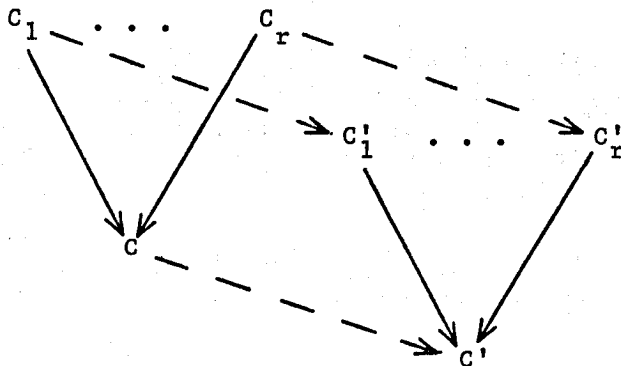


Figure 1. Correspondence of the general rule to the ground rule

Note that if σ, ρ are substitutions such that no variable of a component of σ is the variable of a component of ρ , then $\sigma \cup \rho$ is also a substitution. Henceforth, whenever we write an expression $\sigma \cup \rho$ we imply that σ and ρ are just such substitutions.

We shall frequently make use of the trivial fact that if A and B are sets of formulae such that no variable of A is the variable of a component of ρ , and no variable of B is the variable of a component of σ , then

$$(A \cup B)(\sigma \cup \rho) = (A\sigma \cup B\rho).$$

Whenever, in the course of a construction, we call for a substitution θ such that, say, $C\theta$ has a certain property, it is understood that every variable of a component of θ actually appears in C . Clearly, if there is any substitution at all which fulfils the requirement, then there is such a 'non-redundant' substitution.

Rule 1 (general form). Let P be a predicate letter (not one of the special forms for equality) and let

$$\{A \vee P(\vec{\alpha}_1) \vee \dots \vee P(\vec{\alpha}_j)\} \tag{4.3}$$

and $\{B \vee \sim P(\vec{\beta}_1) \vee \dots \vee \sim P(\vec{\beta}_k)\}$ (4.4)

be clauses with no variables in common. Then infer

$$\{A \vee B \vee \neq \{\vec{\alpha}_1, \dots, \vec{\alpha}_j, \vec{\beta}_1, \dots, \vec{\beta}_k\}\}. \tag{4.5}$$

As with ground clauses, A and B may also have literals with predicate letter P , $\sim P$ respectively. Note that ground rule 1 is a special case of this one.

The clause (4.4) may be a variant of (4.3), although in section 5 we shall establish that this case can be eliminated.

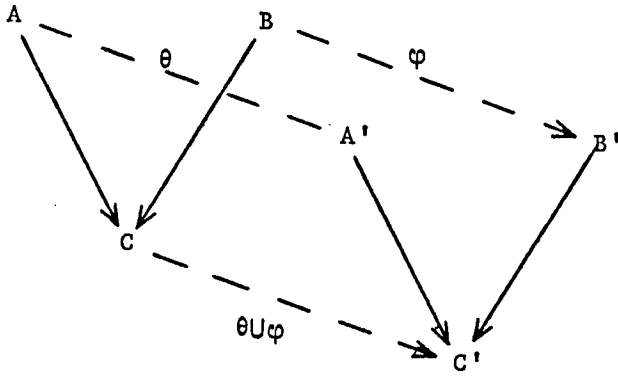


Figure 2. Diagram for the justification of rule 1

Justification of rule 1 (see figure 2). Let

$$A' = \{A' \vee P(\vec{\alpha}'_1) \vee \dots \vee P(\vec{\alpha}'_r)\}$$

and $B' = \{B' \vee \sim P(\vec{\beta}'_1) \vee \dots \vee \sim P(\vec{\beta}'_s)\}$

be ground clauses which are reduced instances respectively of

$$A = \{A^* \vee A \vee P(\vec{\alpha}_1) \vee \dots \vee P(\vec{\alpha}_j)\}$$

and $B = \{B^* \vee B \vee \sim P(\vec{\beta}_1) \vee \dots \vee \sim P(\vec{\beta}_k)\}$

by the substitutions θ and ϕ . A^* represents those inequalities in A which are unified by θ , $P(\vec{\alpha}_1), \dots, P(\vec{\alpha}_j)$ are all of the literals in A which are carried into $P(\vec{\alpha}'_1), \dots, P(\vec{\alpha}'_r)$ by θ , and A represents the remaining literals. Similarly for B .

Now, applying ground rule 1 to A' and B' gives

$$C' = \{A' \vee B' \vee \neq \{\vec{\alpha}'_1, \dots, \vec{\beta}'_s\}\}.$$

Applying general rule 1 to A, B gives

$$C = \{A^* \vee B^* \vee A \vee B \vee \neq \{\vec{\alpha}_1, \dots, \vec{\beta}_k\}\},$$

but $C(\theta \cup \phi) = \{A^* \theta \vee B^* \phi \vee A' \vee B' \vee \neq \{\vec{\alpha}'_1, \dots, \vec{\beta}'_s\}\},$

which reduces to C' , since $A^* \theta$ and $B^* \phi$ consist of trivial inequalities.

THEOREM PROVING

Rule 2 (general form). Let A, B_1, \dots, B_r be clauses with no variables in common, and such that $B_i \neq A, i=1, \dots, r$. Let M be the set of all terms which are primary terms of any of the clauses A, B_1, \dots, B_r . Let \mathcal{P} be a unifiable partition of M with m.g.u. σ , where σ does not contract \mathcal{P} , and put

$$A^* = \mathcal{R}(A\sigma),$$

$$B_i^* = \mathcal{R}(B_i\sigma), i=1, \dots, r.$$

Require that A^* have the form

$$\{A^* \vee \alpha^* = \beta_1^* \vee \dots \vee \alpha^* \beta_r^*\},$$

where

1. α^* appears in, and is maximal in, each of A^*, B_1^*, \dots, B_r^* .
2. α^* does not appear in A^* .
3. none of the clauses A^*, B_1^*, \dots, B_r^* is trivially satisfied.

Then infer

$$\{A^* \vee B_1^*[\beta_1^*/\alpha^*] \vee \dots \vee B_r^*[\beta_r^*/\alpha^*]\}.$$

Justification of rule 2 (see figure 3). Let

$$A' = \{A' \vee \alpha' = \beta_1' \vee \dots \vee \alpha' = \beta_r'\}$$

and B_1', \dots, B_r' be ground clauses as required for the application of ground rule 2 (with α' denoting the key term). Suppose that

$$A' = \mathcal{R}(A\theta),$$

$$B_i' = \mathcal{R}(B_i\theta_i), i=1, \dots, r,$$

where we assume that A, B_1, \dots, B_r have no variables in common. Put $\Theta = \theta \cup \theta_1 \cup \dots \cup \theta_r$.

Let M be the set of all primary terms of A, B_1, \dots, B_r and let \mathcal{P} be the partition of M induced by Θ (i.e., the partition of M determined by the equivalence relation $X \cong Y$ iff $X\Theta = Y\Theta$). \mathcal{P} is unifiable and is not contracted by its m.g.u. σ . Thus put

$$A^* = \mathcal{R}(A\sigma),$$

$$B_i^* = \mathcal{R}(B_i\sigma), i=1, \dots, r.$$

There is a substitution λ such that $\Theta = \sigma\lambda$. Moreover, λ does not unify any pair of primary terms occurring in A^*, B_1^*, \dots, B_r^* , since σ already unifies the partition \mathcal{P} induced on M by Θ ; so

$$A' = A^*\lambda,$$

$$B_i^* = B_i^*\lambda \quad i=1, \dots, r.$$

Let α^* be the (unique) term in A^* such that $\alpha^*\lambda = \alpha'$. Since α' appears and is maximal in each clause A', B_1', \dots, B_r' , it follows by lemma (4.1) that α^* appears and is maximal in each clause A^*, B_1^*, \dots, B_r^* .

Let $\beta_1^*, \dots, \beta_r^*$ be the terms in A^* which are the ancestors of $\beta_1', \dots, \beta_r'$ respectively by λ . Then A^* has the form

$$\{A^* \vee \alpha^* = \beta_1^* \vee \dots \vee \alpha^* = \beta_r^*\},$$

where α^* does not appear in A^* , since α' does not appear in A' .

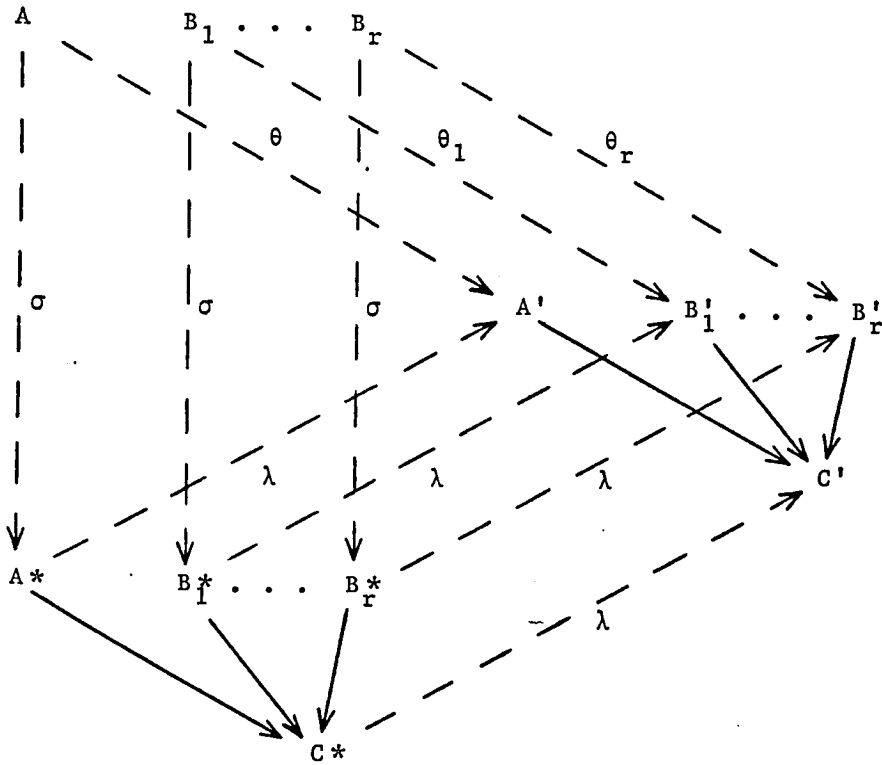


Figure 3. Diagram for the justification of rule 2

Also, since none of A, B_1, \dots, B_r is trivially satisfied, it must be the case that none of A^*, B_1^*, \dots, B_r^* is trivially satisfied. Hence general rule 2 applies and by it we deduce

$$C^* = \{A^* \vee B_1^*[\beta_1^*/\alpha^*] \vee \dots \vee B_r^*[\beta_r^*/\alpha^*]\}.$$

Ground rule 2 applied to A', B_1, \dots, B_r yields

$$C' = \{A' \vee B_1[\beta_1/\alpha'] \vee \dots \vee B_r[\beta_r/\alpha']\}.$$

But now

$$\begin{aligned} C^*\lambda &= \{A^*\lambda \vee B_1^*[\beta_1^*/\alpha^*]\lambda \vee \dots \vee B_r^*[\beta_r^*/\alpha^*]\lambda\} \\ &= \{A' \vee B_1[\beta_1/\alpha'] \vee \dots \vee B_r[\beta_r/\alpha']\} \\ &= C' \end{aligned}$$

for, by lemma (4.2), $B_i^*[\beta_i^*/\alpha^*]\lambda = B_i[\beta_i/\alpha']$.

Rule 3 (general form). Let A be a clause and let M be the set consisting of all the primary terms of A . Let \mathcal{P} be a unifiable partition of M which is not contracted by its m.g.u. σ . Set

$$A^* = \mathcal{R}(A\sigma),$$

where we require that A^* not be trivially satisfied. Let α^* be a maximal term of A^* , appearing in A^* , which is not a constant. We distinguish two cases:

THEOREM PROVING

Case 1. α^* is a variable. Let n be the maximum of the degrees of the function letters appearing in the problem at hand, and let $X_1, \dots, X_n, Y_1, \dots, Y_n$ be variables not appearing in A^* . Then infer

$$\{\vec{X} \neq \vec{Y} \vee A^*\},$$

where $\vec{X} = X_1, \dots, X_n$ and $\vec{Y} = Y_1, \dots, Y_n$.

N.B. We shall show in section 6 that no deduction whatever is required in this case. The justification of the rule, however, is vastly simplified if we retain this rather trivial inference for the present.

Case 2: α^* begins with a function letter of degree $n > 0$. Let X_1, \dots, X_n be variables which do not appear in A^* , and set $\vec{X} = X_1, \dots, X_n$. Let F denote the initial letter of α^* and write $\alpha^* = F(\vec{\beta}^*)$. Infer

$$\{\vec{X} \neq \vec{\beta}^* \vee A^*[F(\vec{X})/\alpha^*]\}.$$

Justification of rule 3. Let A' be a ground clause and let α' be a maximal term appearing in A' , $\alpha' = F(\vec{\beta})$, where F is a function letter of positive degree. Let $F(\vec{\gamma})$ be some other ground term which does not contain α' . Applying the ground form of rule 3 we obtain

$$C' = \{\vec{\gamma} \neq \vec{\beta} \vee A'[F(\vec{\gamma})/\alpha']\}.$$

Now suppose that $A' = \mathcal{R}(A\theta)$. Let M be the set of all primary terms of A , and let \mathcal{P} be the partition of M induced by θ . As for rule 2, \mathcal{P} is unifiable and is not contracted by its m.g.u., σ .

Put $A^* = \mathcal{R}(A\sigma)$ and let λ be a substitution such that $\sigma\lambda = \theta$. λ does not unify any pair of primary terms occurring in A^* , since σ unifies \mathcal{P} and \mathcal{P} is the partition induced by θ . Thus

$$A' = A^*\lambda,$$

and there is a unique term α^* appearing in A^* such that $\alpha^*\lambda = \alpha'$. α^* is maximal in A^* and, since A' is not trivially satisfied, A^* is not trivially satisfied. Moreover, α^* is not a constant, since $\alpha^*\lambda = F(\vec{\beta})$ and F has positive degree. Consider each of the cases in the rule.

Case 1. α^* is a variable. Let n, \vec{X}, \vec{Y} be as in the statement of the rule and deduce

$$C^* = \{\vec{X} \neq \vec{Y} \vee A^*\}.$$

Let k be the degree of F , so $k \leq n$, and set

$$\phi = \{\beta_1/X_1, \gamma_1/Y_1, \dots, \beta_k/X_k, \gamma_k/Y_k, X_{k+1}/Y_{k+1}, \dots, X_n/Y_n, F(\vec{\gamma})/\alpha^*\}.$$

Then

$$C^*\phi = \{\vec{\beta} \neq \vec{\gamma} \vee X_{k+1} \neq X_{k+1} \vee \dots \vee X_n \neq X_n \vee A^*[F(\vec{\gamma})/\alpha^*]\},$$

which, after partial reduction, becomes

$$D = \{\vec{\beta} \neq \vec{\gamma} \vee A^*[F(\vec{\gamma})/\alpha^*]\},$$

but $D\lambda = \{\vec{\beta} \neq \vec{\gamma} \vee A'[F(\vec{\gamma})/\alpha']\}$

so $\mathcal{R}(C^*\phi\lambda) = C'$.

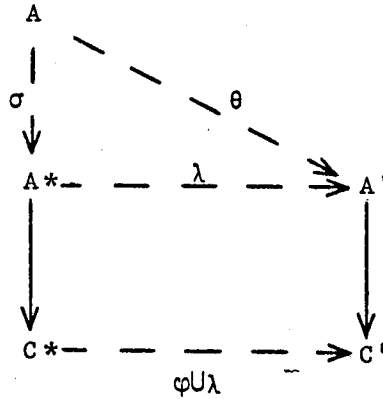


Figure 4. Diagram for the justification of rule 3 (case 2)

Case 2. α^* begins with a function letter of positive degree (see figure 4). That letter must be F , since $\alpha^*\lambda = F(\vec{\beta})$. Let n be the degree of F and apply the general rule to deduce

$$C^* = \{\vec{X} \neq \vec{\beta}^* \vee A^*[F(\vec{X})/\alpha^*]\},$$

where $\vec{X} = X_1, \dots, X_n$ consists of variables not appearing in A^* and $\alpha^* = F(\vec{\beta}^*)$. Let

$$\phi = \{\gamma_1/X_1, \dots, \gamma_n/X_n\}.$$

Then

$$\begin{aligned} C^*(\phi \cup \lambda) &= \{\vec{\gamma} \neq \vec{\beta} \vee A^*\lambda[F(\vec{X})\phi/\alpha^*\lambda]\} \\ &= \{\vec{\gamma} \neq \vec{\beta} \vee A'[F(\vec{\gamma})/\alpha']\}. \end{aligned}$$

General rules 1–3 are to be applied to the clauses of a set S as follows.

Rule 1. For each pair of clauses $A, B \in S$ (not necessarily distinct) form variants A', B' which have no variables in common and apply the rule stated above in all possible ways to A', B' .

Rule 2. For each clause $A \in S$ containing k equations (and no type I literals) and for each sequence B_1, \dots, B_j of clauses in $S, j \leq k, (B_j$ contains only type II

THEOREM PROVING

literals), form variants A', B'_1, \dots, B'_j having no variables in common and apply the rule in all possible ways to A', B'_1, \dots, B'_j .

Rule 3. Apply the rule in all possible ways to each clause of S which consists entirely of type II literals. In case 1 of the rule, n is to be the maximum of the degrees of the function letters appearing in S .

Clearly, we can deduce only a finite number of clauses by applying rules 1–3 to the clauses of a finite set S .

The main completeness theorem now follows. For any finite set of clauses S let $E_*(S)$ denote the set consisting of S together with all of the clauses that can be deduced by applying general rules 1–3 to clauses of S , reduced as always. $E_*(S)$ will also be a finite set of clauses, so we may recursively define $E_*^n(S) = E_*(E_*^{n-1}(S))$, $E_*^0(S) = S$.

Now any instance of a trivial equation is also a trivial equation, and likewise for trivial inequalities, hence, in view of the justifications given, we have established the following.

Theorem. Let S be any set of clauses and let S' be any set of reduced ground instances of S , then all of the clauses in $E^n(S')$ are reduced instances of clauses in $E_*^n(S)$.

By a *terminal* clause we mean a clause which has an instance which reduces to \square . It is clear that a clause C is terminal iff $C = \square$ or $C = \{L_1 \vee \dots \vee L_n\}$, where the literals L_i are all inequalities and $\{L_1^s, \dots, L_n^s\}$ is unifiable.

Now suppose that S is unsatisfiable. By Herbrand's Theorem there is a finite set of (reduced) instances of clauses of S over the Herbrand universe of S which is unsatisfiable. Call this set of ground clauses S' . From section 2 we see that for some integer n , $\square \in E^n(S')$, hence we have proved the following:

Completeness theorem. Let S be an unsatisfiable set of clauses, then for some integer n , $E_*^n(S)$ contains a terminal clause.

5. APPLICATION OF A RESULT ON RESOLUTION

Rule 1 is essentially resolution (Robinson 1965a) applied to equivalence classes of literals, instead of identical instances. As such, it would seem likely that some of the stronger results about resolution could be adapted to the present situation (Wos *et al.* 1965, Robinson 1965b, Slagle 1967). This is indeed the case.

Let us consider again the expansion theorem of section 3, and particularly case 1 of the proof of that theorem. Given an unsatisfiable set S of ground clauses we suppose that V is a partial assignment defined on all of T (hence all of E) which does not falsify any clause of S , and take $\mathcal{P} = \{U_1, \dots, U_R\}$ to be the partition of U induced by V .

Writing $\sim U_i$ for the set $\{X: \sim X \in U_i\}$, we define the *representation* of a clause C to be the set

$$C' = \{W: (W \in \mathcal{P} \text{ or } \sim W \in \mathcal{P}) \ \& \ W \cap C \neq \phi\}.$$

If we now regard the sets U_1, \dots, U_R as ground atoms, then the representations become clauses, and if we denote by S' the set consisting of the representations of the clauses in S which are not satisfied by V , then the hypothesis that S is unsatisfiable implies that S' is unsatisfiable. The argument in case 1 of the expansion theorem consisted essentially of the application of the proof of the ground resolution theorem of Robinson (1965a) to S' .

Perhaps the strongest result known at present which can be used here is the the unresolved AM-clash theorem proved by J. R. Slagle (1967, theorem 6). Let A be a linear ordering of the (finite) set \mathcal{P} , and let M be a model of \mathcal{P} .

An AM-clash is a finite collection of clauses $\{E_1, \dots, E_q, C\}$ satisfying the following conditions:

- (i) The *nucleus* C contains at least q literals L_1, \dots, L_q , where $q \geq 1$.
- (ii) For each i , $1 \leq i \leq q$, the clause E_i contains the complement $\sim L_i$ of the literal L_i , but does not contain the complement of any literal which occurs in any E_j , $1 \leq j \leq q$, nor the complement of any other literal in C .
- (iii) Each of the clauses E_1, \dots, E_q is false in M , thus C is true in M .
- (iv) $|L_i|$ is the largest atom appearing in E_i (in the ordering A). $|L_i|$ denotes L with the negation symbol (if present) removed.

If, in addition, L_1, \dots, L_q are the only literals in C which are true in M , then $\{E_1, \dots, E_q, C\}$ is a *maximal AM-clash*.

The *resolvent* of $\{E_1, \dots, E_q, C\}$ is the clause

$$\{(E_1 - \{\sim L_1\}) \vee \dots \vee (E_q - \{\sim L_q\}) \vee (C - \{L_1, \dots, L_q\})\},$$

which is implied by E_1, \dots, E_q, C . These concepts are discussed in Robinson (1967) and Slagle (1967).

The AM-clash theorem states that if S' is a finite, unsatisfiable collection of non-empty clauses, A is an ordering of the atoms appearing in S' , and M is an interpretation, then S' contains a maximal AM-clash whose resolvent is not in S' .

In constructing a rule based on this theorem we cannot, of course, refer directly to the classes U_i . To avoid this difficulty we shall use orderings and interpretations defined solely in terms of the predicate letters, which will be applicable to any partition \mathcal{P} .

We define an *equable set* of literals to be a non-empty set of type 1 literals which have the same sign and predicate letter. Let A be an ordering of the predicate letters appearing in a ground system S , and let M be an interpretation defined for the type 1 literals of S which always assigns the same truth-value to atoms which have the same predicate letter. An equable set is said to be true in M if all of its literals are true in M , and false in M otherwise.

An AME-clash is a set of clauses

$$\{E_1, \dots, E_q, C\} \quad (5.1)$$

THEOREM PROVING

satisfying the following requirements:

- (i) The nucleus C contains at least $q \geq 1$ disjoint, equable sets of literals $\mathcal{L}_1, \dots, \mathcal{L}_q$.
- (ii) For each $i, 1 \leq i \leq q$, the clause E_i contains an equable set \mathcal{L}'_i whose literals have the same predicate letter as those in \mathcal{L}_i , but with opposite sign. Moreover, $E_i - \mathcal{L}'_i$ does not contain the complement of any type I literal in $C - \mathcal{L}_i$, and E_i does not contain the complement of any type I literal in any clause $E_j, 1 \leq j \leq q$.
- (iii) All of the type I literals of E_1, \dots, E_q are false in M .
- (iv) The predicate letter appearing in \mathcal{L}'_i is maximal (in the ordering A) among the predicate letters appearing in E_i .

The clauses are, of course, distinct. Strictly speaking, the clash consists of the set (5.1) together with a specification of the sets $\mathcal{L}_1, \dots, \mathcal{L}_q, \mathcal{L}'_1, \dots, \mathcal{L}'_q$, since there will, in general, be many ways to choose these.

The clash (5.1) is said to be maximal iff $C - \mathcal{L}_1 - \dots - \mathcal{L}_q$ contains no type I literals which are true in M .

If X is an equable set of atoms, say $\{P(\vec{\alpha}_1), \dots, P(\vec{\alpha}_n)\}$, then write $\neq \{X\}$ for the set of inequalities $\neq \{\vec{\alpha}_1, \dots, \vec{\alpha}_n\}$. We define the E -resolvent of the clash (5.1) to be the clause

$$\{(E_1 - \mathcal{L}'_1) \vee \dots \vee (E_q - \mathcal{L}'_q) \vee (C - \mathcal{L}_1 - \dots - \mathcal{L}_q) \\ \vee \neq \{|\mathcal{L}'_1 \cup \mathcal{L}_1|\} \vee \dots \vee \neq \{|\mathcal{L}'_q \cup \mathcal{L}_q|\}\}. \quad (5.2)$$

The E -resolvent of an E -clash follows from the clauses in the clash; for suppose that V is an interpretation which satisfies E_1, \dots, E_q, C but falsifies all of the literals in the resolvent (5.2) except perhaps those in $C - \mathcal{L}_1 - \dots - \mathcal{L}_q$. Then the inequalities indicated explicitly in (5.2) are all false, so, for each i , the literals in \mathcal{L}'_i are all true, since $E_i - \mathcal{L}'_i$ is false. Hence the literals of \mathcal{L}_i are all false under V , so at least one literal in $C - \mathcal{L}_1 - \dots - \mathcal{L}_q$ is true.

We shall now show that ground rule 1 of section 3 can be replaced by the following rule, rule 1', without changing the results obtained in that section.

Rule 1'. Infer the E -resolvent of any maximal AME-clash.

We need only re-argue case 1 of the expansion theorem. Let S' be the unsatisfiable set of representations defined earlier. The model M is already defined for the classes of \mathcal{P} , since these are equable sets, and we may choose a linear ordering A^* which contains the given order A .

The unresolved AM-clash theorem assures us that S' contains a maximal A^*M -clash, say

$$\{E'_1, \dots, E'_q, C'\} \quad (5.3)$$

whose resolvent is not contained in S' . Let L_1, \dots, L_q be literals (i.e., classes)

of C' as in the definition, so that the resolvent of (5.3) is

$$\{(E'_1 - \{\sim L_1\}) \vee \dots \vee (E'_q - \{\sim L_q\}) \vee (C' - \{L_1, \dots, L_q\})\}. \quad (5.4)$$

Let E_1, \dots, E_q, C be clauses in S whose representations are E'_1, \dots, E'_q, C' respectively. These clauses must be distinct since their representations are distinct. For each $i, 1 \leq i \leq q$, define

$$\begin{aligned} \mathcal{L}_i &= L_i \cap C, \\ \mathcal{L}'_i &= \sim L_i \cap E_i. \end{aligned}$$

Clearly \mathcal{L}_i and \mathcal{L}'_i are equable sets with the same predicate letter but opposite sign, and the remaining requirements of conditions (ii)–(iv) in the definition of an AME-clash follow at once from the corresponding requirements for the AM-clash (5.3). Thus

$$\{E_1, \dots, E_q, C\} \quad (5.5)$$

with the sets $\mathcal{L}_1, \dots, \mathcal{L}_q, \mathcal{L}'_1, \dots, \mathcal{L}'_q$ is an AME-clash, and, in fact, is maximal, since, as (5.3) is maximal, $\mathcal{L}_1 \cup \dots \cup \mathcal{L}_q$ contains all the literals in C which are true in M .

The E -resolvent of (5.5) is (5.2), which cannot already appear in S , for its representation is just (5.4), which is not in S' .

It is a straightforward matter now to extend rule 1' to the general case. We define an AME-clash for clauses containing variables precisely as for ground clauses, except for the additional proviso that the clauses E_1, \dots, E_q, C have no variables in common. It is allowed that some clause E_i is a variant of another clause E_j , but note that requirement (iii) precludes the possibility that any clause E_i is a variant of the nucleus. The E -resolvent of a general AME-clash is defined exactly as before, and the extended rule 1' is the same as the ground rule.

We omit the details of the justification of the extension, since the argument is not essentially different from that given for rule 1.

6. SEARCH PRINCIPLES

In the foregoing sections we have developed a special system of logic which provides a theoretical basis for the design of theorem-proving programs with the equality relation 'built in'. This alone, however, is not enough for the development of an efficient procedure, and we shall now treat three techniques for increasing the efficiency of refutation procedures. Two of these are rather elementary devices based on the properties of equality. The third, however, the subsumption principle, is a very powerful technique, both computationally and theoretically, and we shall discuss it at some length.

Let C be a ground clause containing the inequalities L and M , where $L^s \cap M^s \neq \phi$. Then C is equivalent to the clause

$$C' = \{(C - \{L, M\}) \vee \neq (L^s \cup M^s)\}.$$

THEOREM PROVING

One can show by a straightforward argument that any interpretation which satisfies C also satisfies C' , and conversely. The expansion result of section 3 is not affected by combining inequalities in the clause produced there, so we may combine intersecting inequalities without losing completeness.

There is a way in which a clause can be trivially satisfied which has not previously been discussed. Suppose a ground clause A contains type 1 literals $P(\vec{\alpha})$, $\sim P(\vec{\beta})$, where P represents some predicate letter of degree $n > 0$, and $\vec{\alpha} = \{\alpha_1, \dots, \alpha_n\}$, $\vec{\beta} = \{\beta_1, \dots, \beta_n\}$. Suppose further that for each i , $1 \leq i \leq n$, either $\alpha_i = \beta_i$, or there is an inequality $M_i \in A$ such that $\{\alpha_i, \beta_i\} \subset M_i^s$. Then A is satisfied by every interpretation V ; for suppose that V falsifies every literal in A except perhaps for $P(\vec{\alpha})$ and $\sim P(\vec{\beta})$. Then $V(\vec{\alpha}) = V(\vec{\beta})$, since for each i , $\alpha_i = \beta_i$ or $\{\alpha_i, \beta_i\} \subset M_i^s$, and $V(M_i) = \text{False}$. But then $V(P(\vec{\alpha})) = \sim V(\sim P(\vec{\beta}))$ and V satisfies A .

We may drop the restriction to ground clauses, for if any clause A contains type 1 literals and inequalities as above, then so does any ground instance of A . Thus we may include such clauses in the class of trivially satisfied clauses defined earlier without affecting any of the preceding results.

We come now to the subsumption principle. A clause C is said to *subsume* the clause D ($\neq C$) iff there is a substitution θ and a function t satisfying the following conditions:

1. $t: \mathcal{R}(C\theta) \rightarrow D$. (As before, $\mathcal{R}(C\theta)$ denotes $C\theta$ with any trivial inequalities deleted.)
2. If $L \in \mathcal{R}(C\theta)$ and L is not an inequality, then $t(L) = L$.
3. If $L \in \mathcal{R}(C\theta)$ and L is an inequality, then $t(L)$ is also an inequality and $L^s \subset (t(L))^s$.

This definition is a natural extension of that given by Robinson (1965a). The following is also quite similar to a theorem proved by Robinson.

Subsumption theorem. If C subsumes D then C implies D .

Proof. Suppose that θ and t are respectively the substitution and function in the definition (see figure 5). Let $D' = \mathcal{R}(D\sigma)$ be any ground instance of D and put $C^* = \mathcal{R}(C\theta)$, $C' = \mathcal{R}(C^*\sigma)$. Define a function $t': C' \rightarrow D'$ as follows: For each literal $L' \in C'$ choose one literal $L^* \in C^*$ such that $L^*\sigma = L'$, and define $t'(L') = t(L^*)\sigma$. If $L' \in C'$ is not an inequality then

$$t'(L') = t(L^*)\sigma = L^*\sigma = L'.$$

If L' is an inequality we have that

$$(t'(L'))^s = (t(L^*))^s\sigma \supset (L^*)^s\sigma = (L')^s.$$

Now C' is a ground clause, since every term in C^* appears in D and D' is a ground clause. If V is an interpretation which satisfies C then V also satisfies C' , so there is a literal $L' \in C'$ such that $V(L') = \text{True}$. But $t'(L') \in D'$ and

clearly $V(t'(L')) = \text{True}$. Thus V satisfies any ground instance of D , so V satisfies D . Q.E.D.

Corollary (5.1). If C subsumes D and σ is any substitution, then C subsumes $\mathcal{R}(D\sigma)$.

Proof. The construction given in the first paragraph of the proof of the subsumption theorem establishes the corollary. Note that the condition that D' be a ground clause is only used to show that C' is a ground clause. It is not necessary for the construction.

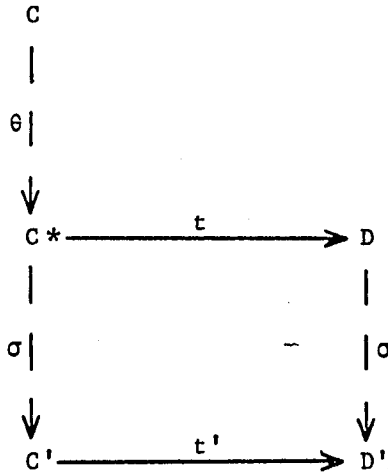


Figure 5. Diagram for the proof of the subsumption theorem

Corollary (6.2). If B subsumes C and C subsumes D , then B subsumes D (subsumption is transitive).

Proof. (see figure 6). Let θ', u be respectively the substitution and function required by the definition of ' C subsumes D ', and let $C' = \mathcal{R}(C\theta')$. By corollary (5.1) B subsumes C' . Let θ, t be the substitution and function required by the definition of ' B subsumes C ', and put $B' = \mathcal{R}(B\theta)$. Consider the function $r: B' \rightarrow D$ defined by $r(L) = u(t(L))$. If $L \in B'$ and L is not an inequality, $r(L) = u(t(L)) = u(L) = L$. If L is an inequality,

$$(r(L))^s = (u(t(L)))^s \supset (t(L))^s \supset L^s.$$

Thus θ, r satisfy the definition and B subsumes D .

An algorithm is given in Robinson (1965a) which can readily be modified to obtain an effective test for determining whether a clause C subsumes a clause D . We shall not, however, discuss this matter further here.

It is an immediate consequence of the subsumption theorem that if S is an unsatisfiable set of clauses, and $D \in S$ is subsumed by some clause in $S - \{D\}$, then $S - \{D\}$ is unsatisfiable. Thus we may contemplate including the subsumption principle in refutation procedures for problems with equality, namely: discard any clause $D \in S$ which is subsumed by a clause in $S - \{D\}$.

THEOREM PROVING

We must, however, show that the procedure is still complete when subsumption is included. This is by no means obvious, a point which seems to have been overlooked in earlier discussions of the subject. It is entirely conceivable that the subsumption principle might be too 'strong', i.e., that it could cause deletion of clauses that were essential to the refutation which the rules of inference would generate in its absence. Such is not the case, as we shall now demonstrate.

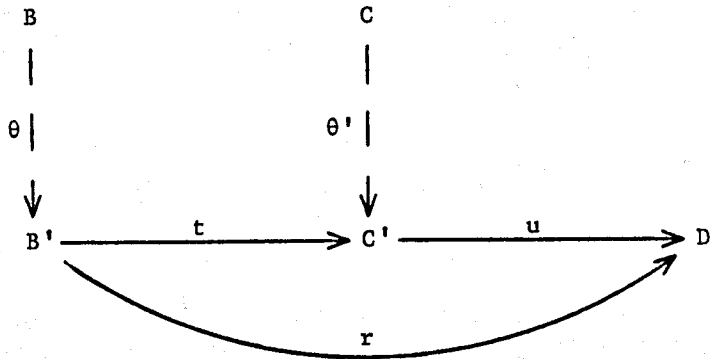


Figure 6. Diagram for the proof of corollary (6.2)

Let S' be a finite, reduced, unsatisfiable set of non-empty ground clauses. The clause constructed in the expansion theorem of section 3 (with either rule 1 or rule 1') is called the *expansion clause* of S' .

Lemma. Let S' be a finite, reduced, unsatisfiable set of non-empty ground clauses. Then the expansion clause of S' is not subsumed by any clause in S' .

Proof. Let C' be the expansion clause of S' . If C' arises by the application of rules 1, 2, or 3, then there is a partial interpretation V which falsifies C' but does not falsify any clause in S' , so C' is not subsumed by any clause in S' . Suppose, then, that C' arises by an application of rule 1', and let S'' be the set of representations of the clauses in S' which are not satisfied by the partial interpretation V , defined on T (see the discussion at the beginning of section 5).

Let C'' be the resolvent which is produced by the proof of the AM-clash theorem (applied to S''). In that proof another interpretation N is defined, and it is shown that C'' is falsified by both M and N , while every clause in S'' is satisfied either by M or by N . This fact is used to show that $C'' \notin S''$. (In Slagle's argument the literals have been renamed so that the M -true literals are just those with negation symbols.) It follows that no clause of S'' subsumes C'' (i.e., is a subset of C'').

Consider the corresponding clause C' which is the E -resolvent of an E -clash corresponding to the clash in S'' whose resolvent is C'' . If B' is any clause in

S' with a type I literal, then B' cannot subsume C' , for, if it did, the set of type I literals in B' would be contained in the set of type I literals of C' . Thus, denoting the representation of B' by B'' , we would have $B'' \subset C''$. On the other hand, if $B' \in S'$ has only type II literals, B' cannot subsume C' , for every type II literal in C' is false in the partial interpretation V , but V does not falsify any clause in S' .

We see from the lemma that *subsumption does not destroy completeness at the ground level*. The general case requires some argument, though. Let S be a finite, unsatisfiable set of clauses and S' a finite, unsatisfiable set of ground instances of clauses in S . The difficulty is that if $C, D \in S$, C subsumes D , and $D' \in S'$ is an instance of D , it does not follow that there is a clause in S' which subsumes D' ; so if we simply apply subsumption to both S and S' independently it may no longer be the case that the clauses remaining in S' are all instances of clauses left in S . The natural remedy for this condition is to enlarge S' by the addition of a suitable instance of C' which does subsume D' , and the argument about to be given consists mainly in the systematic application of this construction.

For any finite set of clauses S , let $F(S)$ denote the set obtained by applying rule 1 (or rule 1'), rule 2, and rule 3 to S in all possible ways, combining the clauses so obtained with S , reducing as usual, and applying the subsumption principle to the remainder. We define $F^0(S) = S$, $F^n(S) = F(F^{n-1}(S))$ as usual.

Modified completeness theorem. If S is a finite, unsatisfiable set of clauses then for some integer $n \geq 0$, $F^n(S)$ contains a terminal clause.

Proof. Let S' be a finite, unsatisfiable set of non-empty, reduced, ground instances of clauses in S . (We suppose that S does not contain a terminal clause, for otherwise the result is trivial.) Mark each clause of S' which is subsumed by another clause in S' , and leave the remaining clauses in S' unmarked. Denoting by $U(S')$ the set of unmarked clauses in S' , we have that $U(S')$ is unsatisfiable, and that the clauses in $U(S')$ are instances of clauses in S .

Denote by W' the set of all clauses which can be deduced from the clauses of $U(S')$ using the ground forms of the rules used in the operator F . Put $V' = W' - S'$. V' is not empty, for let C' be the expansion clause of $U(S')$. Then $C' \in W'$, and $C' \notin U(S')$ by the expansion theorem, and $C' \notin S' - U(S')$ by the preceding lemma, since each clause in $S' - U(S')$ is subsumed by a clause in $U(S')$. Put $T' = S' \cup V'$ and mark each clause in T' which is subsumed by another clause in T' . Clearly T' has more elements than S' and $U(T')$ is unsatisfiable.

From the justification of the rules of inference it follows that each clause in $U(T')$ is either an instance of a clause in $F(S)$ or is an instance of a clause which can be deduced from S and is subsumed by a clause in $F(S)$. Let us say that a set R of clauses *covers* a set R' of ground clauses (some of which may be marked) iff every marked clause in R' is subsumed by a clause

THEOREM PROVING

in $U(R')$ and each clause in $U(R')$ is an instance of, or subsumed by, some clause in R . When the clauses in $U(R')$ are all instances of clauses in R we say that R strictly covers R' . Thus $F(S)$ covers T' and S strictly covers S' .

Define a sequence T^0, T^1, \dots as follows: $T^0 = T'$. Having determined T^0, \dots, T^n so that $F(S)$ covers T^n and $U(T^n)$ is unsatisfiable, proceed as follows:

If every clause of $U(T^n)$ is an instance of a clause in $F(S)$, terminate the sequence. If not, let C' be the lexically earliest clause in $U(T^n)$ which is not an instance of a clause in $F(S)$, and let C be a clause in $F(S)$ which subsumes C' . Let θ be the substitution required by the definition, and put $C'' = \mathcal{R}(C\theta)$. Clearly C'' is a ground clause which subsumes C' , and C'' involves only terms appearing in C' . Obtain T^{n+1} by adding C'' to T^n and marking any unmarked clauses which are subsumed by C'' ; in particular, C' is marked. Note that $F(S)$ covers T^{n+1} and that $U(T^{n+1})$ is unsatisfiable. Also, T^{n+1} has more clauses than T^n .

The sequence must terminate after a finite number of steps, for at each stage we either terminate or reduce by at least one the number of unmarked clauses which are not instances of clauses in $F(S)$.

Let $G(S', S)$ denote the terminal set in the sequence T_0, \dots . Then we have established the following:

1. $F(S)$ strictly covers $G(S', S)$ and $U(G(S', S))$ is unsatisfiable.
2. $G(S', S)$ contains more clauses than S' .
3. $G(S', S)$ involves only terms appearing in S' .

The properties of S, S' that were used in constructing $G(S', S)$ and establishing 1–3 were:

- (1) S strictly covers S' and $U(S')$ is unsatisfiable.
- (2) $\square \notin S'$.

Thus, we may inductively define

$$G^n(S', S) = G(G^{n-1}(S', S), F^{n-1}(S))$$

provided that $\square \notin G^{n-1}(S', S)$. Moreover, each of the sets $G^n(S', S)$ satisfies 1 and 3 (with G^n instead of G , F^n instead of F) and satisfies

- 2' $G^n(S', S)$ contains more clauses than $G^{n-1}(S', S)$.

But by 3, there are only finitely many clauses which can ever appear in the sets $G^n(S', S)$. Thus for some integer $k > 0$, $\square \in G^k(S', S)$, but \square is not subsumed by any clause, so $\square \in U(G^k(S', S))$ and $F^k(S)$ contains a terminal clause. Q.E.D.

We have now fulfilled our promise to eliminate the trivial deduction in case 1 of rule 3, for the clause deduced there is obviously subsumed by the clause from which it was deduced (the empty substitution and the identity map satisfy the definition). The resulting form of the rule is not only more

economical than the previous version, but it also makes no reference to the system S , being entirely defined in terms of the clause to which it is applied.

7. AN EXAMPLE

To illustrate the use of rules 2 and 3 we present a refutation establishing the elementary result that, in a ring, $x \cdot 0 = 0$. Here sxy denotes $x + y$, mx denotes $-x$, and pxy denotes $x \cdot y$. The initial unsatisfiable set of clauses is as follows:

- S1. $\{sxsyz = \underline{ssxyz}\}$ (Associativity of +)
 S2. $\{sx0 = x\}$ (Additive identity)
 S3. $\{sxmx = 0\}$ (Additive inverse)
 D. $\{pxsyz = spxypxz\}$ (Distributive law)
 T. $\{pa0 \neq 0\}$ (Denial of conclusion)

Rule 3 applied to the underlined term in S1 yields

$$E1. \{t \neq sxy \vee u \neq z \vee \underline{sxsyz} = stu\}.$$

Rule 3 applied to the underlined term in E1 gives

$$E2. \{v \neq x \vee w \neq \underline{syz} \vee t \neq sxy \vee u \neq z \vee svw = \underline{stu}\}.$$

Rule 2 applied to S3 $\{y/x\}$ and E2 $\{y/t, my/u, my/z\}$, the underlined terms being unified and replaced, yields

$$E3. \{v \neq x \vee w \neq 0 \vee y \neq \underline{sxy} \vee svw = 0\}.$$

Rule 2 applied to D $\{u/x, x/y, y/z\}$ and E3 $\{pux/x, puy/y\}$ gives

$$E4. \{v \neq \underline{pux} \vee w \neq 0 \vee puy \neq \underline{pusxy} \vee svw = 0\}.$$

Rule 2 applied now to S2 $\{v/x\}$ and E4 $\{0/w\}$ gives

$$E5. \{v \neq \underline{pux} \vee puy \neq \underline{pusxy} \vee v = 0\}.$$

Rule 2 applied to E5 $\{pao/v, a/u, 0/x\}$ and T yields

$$E6. \{pay \neq \underline{pas0y}\}.$$

Applying rule 3 to the underlined term of E6 gives

$$E7. \{x \neq a \vee z \neq \underline{s0y} \vee pay \neq pxz\}.$$

Applying rule 2 to S2 $\{0/x\}$ and E7 $\{0/y\}$ we obtain

$$E8. \{x \neq a \vee z \neq 0 \vee pa0 \neq pxz\},$$

which is terminal, as can be seen by applying the substitution $\{a/x, 0/z\}$.

The principal deficiency in the theory just developed is the lack of an adequate device for 'steering' rules 2 and 3 toward the problem at hand. It will be noted that in the example the clause T does not enter the refutation until the deduction of $E6$, $E1$ - $E5$ being derived solely from the axioms. A suitable adaptation of the set-of-support strategy (Wos *et al.* 1965), added to the present system as a heuristic device, would probably yield a significant improvement in this respect.

THEOREM PROVING

APPENDIX

We present a simplified form of rule 2 in which only two clauses are used to make the inference. The ground form of the rule is as follows: let

$$A = \{A \vee \alpha = \beta_1 \vee \dots \vee \alpha = \beta_r\}$$

and B be clauses such that α appears in, and is maximal in, both A and B and α does not appear in A . Then from A and B infer

$$\{A \vee \alpha = \beta_2 \vee \dots \vee \alpha = \beta_r \vee B[\beta_1/\alpha]\}.$$

The soundness of the inference is apparent. For completeness, we need to reconsider subcase (a) of case 2 in the expansion theorem of section 3. We have a partial assignment V defined on T_1, \dots, T_{n-1} , which falsifies no clause in S ; V does not force the value of T_n , but any extension of V to T_n falsifies some clause in S .

As before, setting $V(T_n) = T_n$ must falsify a clause A of the form

$$\{A \vee T_n = \beta_1 \vee \dots \vee T_n = \beta_r\},$$

where A is already falsified by V . Moreover, we may choose A to be a clause in S which contains the minimum number of equations involving T_n that any clause in S contains which is falsified by putting $V(T_n) = T_n$.

Now setting $V(T_n) = V(\beta_1)$ must falsify some other clause $B \in S$. Certainly A and B satisfy the requirements of the rule, so we may deduce the clause

$$C = \{A \vee T_n = \beta_2 \vee \dots \vee T_n = \beta_r \vee B[\beta_1/T_n]\}.$$

But C is falsified if we put $V(T_n) = T_n$, and C contains only $r-1$ equations involving T_n , thus C is not in S and is not trivially satisfied.

A general form of the simplified rule 2 can be established in a manner entirely analogous to the generalization of the 'clash' form of rule 2 discussed in section 4.

In order that the results of section 6 hold when using the two-clause form of rule 2 it is only necessary to show that the ground clause C constructed above cannot be subsumed by any clause in S . Suppose that C were subsumed by a clause $D \in S$. Then putting $V(T_n) = T_n$ would also falsify D , but D could contain at most $r-1$ equations involving T_n , which contradicts the hypothesis on A .

Certainly the simplified rule is much easier to implement than the form discussed earlier. One should note that application of the rule can be further restricted, since, for a particular clause A and term α , one may choose the term β_1 in any way whatever, provided only that the manner of choosing β_1 can be 'lifted' appropriately in the general rule.

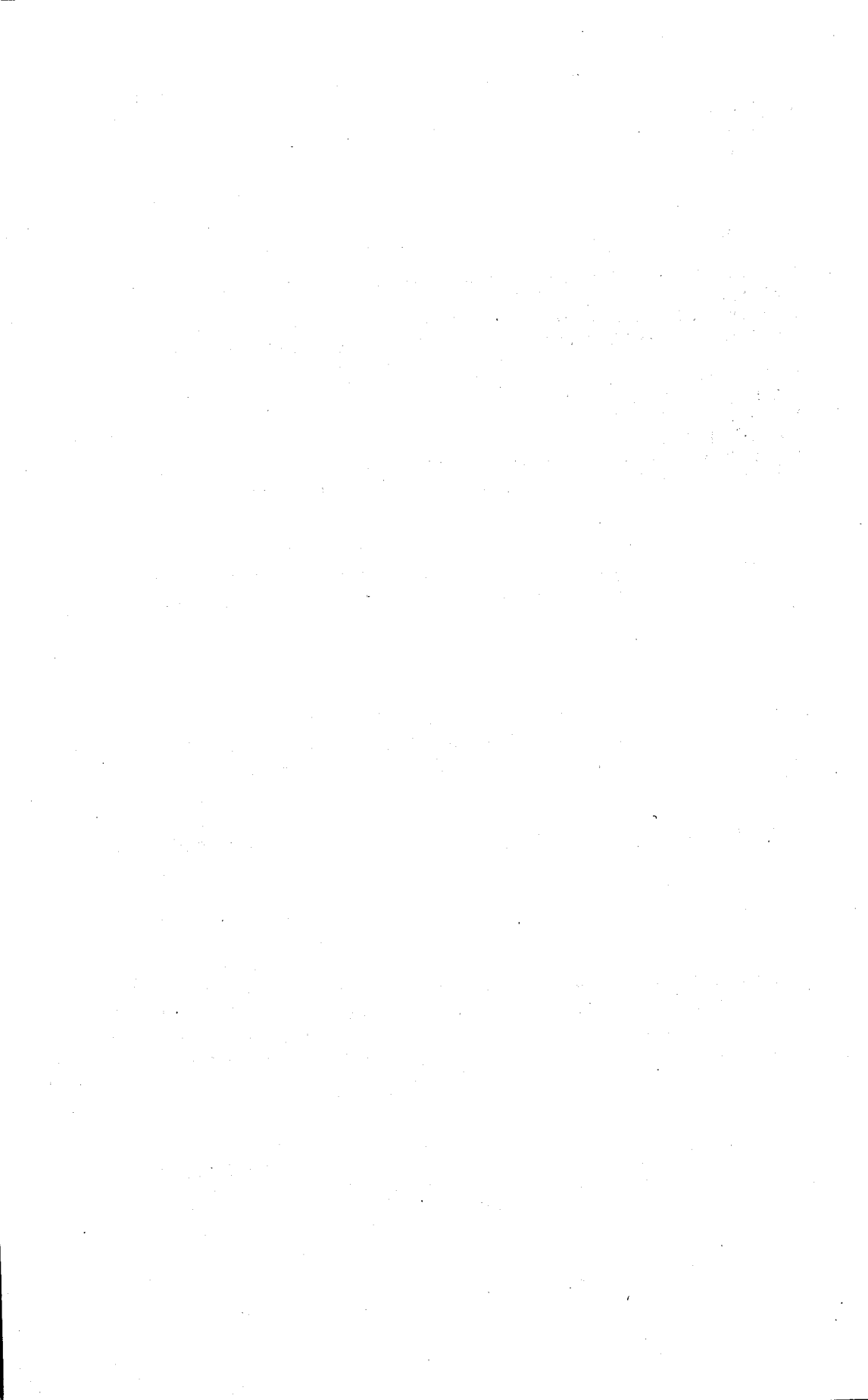
Acknowledgement

I am indebted to Professor J.A. Robinson for a number of important suggestions and criticisms, and also to Dr Robert K. Meyer. This work was supported in part by the Rice

University Computer Project under U.S. Atomic Energy Commission Contract No. AT-(40-1)-2572, and I should like to thank the director, Mr Walter Orvedahl, and staff of the project for their assistance.

REFERENCES

- Robinson, J. A. (1965a) A machine-oriented logic based on the resolution principle. *J. Ass. comput. Mach.*, **12**, 23-41.
- Robinson, J. A. (1965b) Automatic deduction with hyper-resolution. *Int. J. comput. Math.*, **1**, 227-34.
- Robinson, J. A. (1967) A review of automatic theorem proving. *Annual symposia in applied mathematics XIX*, 1-18. Providence, Rhode Island: American Mathematical Society.
- Slagle, J. R. (1967) Automatic theorem-proving with renamable and semantic resolution. *J. Ass. comput. Mach.*, **14**, 687-97.
- Wang, H. (1965) Formalization and automatic theorem-proving. *Proceedings of IFIP Congress 1965*, **1**, 51-8.
- Wos, L., Robinson, G. A., & Carson, D. F. (1965) Efficiency and completeness of the set of support strategy in theorem-proving. *J. Ass. comput. Mach.*, **12**, 536-41.



Paramodulation and Theorem-proving in First-Order Theories with Equality

G. Robinson

Stanford Linear Accelerator Center
Stanford, California

and

L. Wos

Argonne National Laboratory
Argonne, Illinois

INTRODUCTION

A *term* is an individual constant or variable or an n -adic function letter followed by n terms. An *atomic formula* is an n -adic predicate letter followed by n terms. A *literal* is an atomic formula or the negation thereof. A *clause* is a set of literals and is thought of as representing the universally-quantified disjunction of its members. It will sometimes be notationally convenient¹ to distinguish between the empty clause \square , viewed as a clause, and 'other' empty sets such as the empty set of clauses, even though all these empty sets are the same set-theoretic object ϕ . A *ground clause* (term, literal) is one with no variables. A *clause* C' (literal, term) is an *instance* of another clause C (literal, term) if there is a uniform replacement of the variables in C by terms that transform C into C' .

The *Herbrand universe* H_S of a set S of clauses is the set of all terms that can be formed from the function letters and individual constants occurring in S (with the proviso that if S contains no individual constant, the constant a is used). An *interpretation* I of a set S of clauses is a set of ground literals such that for each atomic formula F that can be formed from an n -adic predicate letter occurring in S and n terms from H_S , exactly one of the literals F or \bar{F} (the negation of F) is in I .

For any set J of literals, \bar{J} is the set of negations of members of J . The set J *satisfies* a ground clause C if $J \cap C \neq \phi$ and *condemns* C if $C - \bar{J} = \phi$. J *satisfies* a non-ground clause C if it satisfies every instance of C and *condemns* C if it condemns some instance of C . A clause (possibly ground) that is neither

¹ Note, for example, that the empty set is a satisfiable set of clauses but at the same time is an unsatisfiable clause.

THEOREM PROVING

satisfied nor condemned by J is said to be *undefined* for J ; otherwise it is *defined* for J . J *satisfies* a set S of clauses if it satisfies every clause in S and *condemns* S if it condemns some clause in S .

An R -*interpretation* of a set S of clauses is an interpretation I of S having the following properties: Let α , β , and γ be any terms in H_S and L any literal in I . Then

1. $(\alpha = \alpha) \in I$
2. If $(\alpha = \beta) \in I$ then $(\beta = \alpha) \in I$
3. If $(\alpha = \beta) \in I$ and $(\beta = \gamma) \in I$, then $(\alpha = \gamma) \in I$.
4. If L' is the result of replacing some one occurrence of α in L by β and $(\alpha = \beta) \in I$, then $L' \in I$.

An (R) -*model* of S is an (R) -interpretation of S that satisfies S .

A set S of clauses is (R) -*satisfiable* if there is an (R) -model of S ; otherwise it is (R) -*unsatisfiable*.

If S is a set of clauses or a single clause and T is a set of clauses or a single clause, $S(R)$ -*implies* T (abbreviation $S \vDash T$ or $S \vDash_R T$) if no (R) -model of S condemns T .

A deductive system W is (R) -*deduction-complete* if $S \vdash_W T$ (T is deducible from S in the system W) whenever $S \vDash T$ ($S \vDash_R T$). W is (R) -*refutation-complete* if $S \vdash_W \square$ whenever S is (R) -unsatisfiable.

EQUALITY IN AUTOMATIC THEOREM-PROVING

The methods for dealing with the concept of equality in theorem-proving can be grouped roughly into three classes: (1) those which employ a set of first-order axioms for equality, for example, the following set (which we shall call $E(K)$, where K is the set of first-order sentences under study):

- (i) $(x_1) (x_1 = x_1)$
- (ii) $(x_1) \dots (x_n) (x_0) (x_j \neq x_0 \vee \bar{P}x_1 \dots x_j \dots x_n \vee Px_1 \dots x_0 \dots x_n)$
($j = 1, \dots, n$)
- (iii) $(x_1) \dots (x_n) (x_0) (x_j \neq x_0 \vee f(x_1 \dots x_j \dots x_n) = f(x_1 \dots x_0 \dots x_n))$
($j = 1, \dots, n$)

where n axioms of the form (ii) are included for each n -adic ($n > 0$) predicate letter P occurring in K , and n axioms of the form (iii) are included for each n -adic ($n > 0$) function letter in K^1 ; (2) those which employ a smaller set of second-order axioms for equality; and (3) those which employ a substitution rule for equals as a rule of inference.

SOME DESIRABLE PROPERTIES FOR THEOREM-PROVING ALGORITHMS

In addition to the logical properties of soundness and completeness, two sets of somewhat more elusive properties are of interest in judging the usefulness of the inference apparatus for automatic theorem-proving.

¹ Note that an interpretation I of K is an R -interpretation of K iff it satisfies $E(K)$.

The first set, *efficiency*, *brevity*, and *naturalness*, are global properties in that they deal with the entire proof or proof-search, and are of interest in themselves. *Efficiency* refers to the ease or dispatch with which the search procedure locates a proof. *Brevity* refers to the lengths of proofs found. *Naturalness* refers to being in the spirit of what a human mathematician might write in a proof. Other factors being equal, a briefer proof might be considered more natural, but naturalness goes beyond this. For example, among proofs of roughly the same length, a unit resolution proof¹ might be considered more natural than a non-unit proof.

The second set, *immediacy*, *convergence*, and *generality*, are local properties in that they focus on only a small part of the proof or proof-search and are of interest primarily because they contribute to other properties such as efficiency.

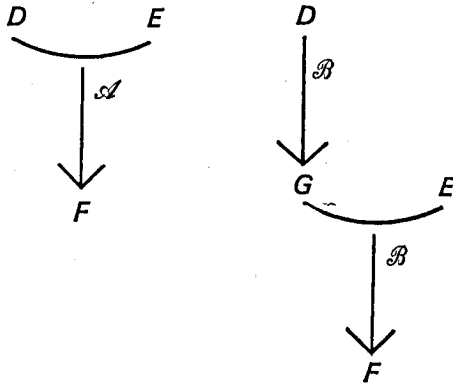


Figure 1

Immediacy is rather easily grasped. One inference apparatus \mathcal{A} is said to be more immediate than another apparatus \mathcal{B} (at least for the case in question) when \mathcal{A} enables one to deduce a given conclusion from a given set of hypotheses in fewer steps than \mathcal{B} . For example (see figure 1), if to infer F from D and E by \mathcal{B} one first had to infer G from D and only then infer F from E and G , while \mathcal{A} allowed the inference of F directly from D and E in one step without recourse to G , then \mathcal{A} would (for this case) be more immediate than \mathcal{B} .

Convergence is a slightly subtler but, for automatic theorem-proving, perhaps more important property. Consider the clause G in the example above. Often such an intermediate result will seriously detract from proof-search efficiency by interacting with other clauses to produce unnecessary 'noise' in the proof-search space, either by generating successive generations of less than helpful clauses, or, somewhat less seriously, by requiring additional

¹ In effect one that is free from simultaneous case-analysis type reasoning and which prefers *modus ponens* to syllogism—formally, one in which non-unit clauses are never resolved against each other.

THEOREM PROVING

machine time to determine that no interesting clauses can be inferred from G . Freedom from this generation of 'side-effect' clauses we call *convergence*. Thus in the example, \mathcal{A} is both more immediate and more convergent than \mathcal{B} .

Generality refers to choosing to infer a clause C rather than a proper instance of C , when either inference could be made from the premises without loss of soundness. For example, inferring from $f(xa) = g(x)$ and $Qf(xa)$ the conclusion $Qg(b)$, although sound, would be less general than inferring $Qg(x)$.

It is not difficult to see the advantage of inferring a clause rather than a proper instance of that clause, since the more general clause, being stronger, has greater potential for future inferences. Perhaps even easier to see is the problem of deciding which proper instance to select if a proper instance were to be preferred to the more general clause. Usually there is an infinite set of proper instances. For example, from $h(xyy) = g(x)$ and $Qh(zww)a$, we can infer $Qg(x)a$ by substitution. There is, however, an infinite set of proper instances of $Qg(x)a$ which could also be legitimately inferred. Among these are $Qg(a)a$, $Qg(g(a))a$, $Qg(g(g(a)))a$ We shall apply the phrase *most general* to a clause (or term) C with respect to some given condition when C satisfies the condition and no clause (term) which satisfies the condition has C as a proper instance.

Of the approaches to equality described above, approach 1 has three obvious disadvantages. One has to do with length of deduction chains in the proof. In order to infer from

- (1) Qa and
- (2) $a=b$

the result

- (3) Qb

one must first infer from the axiom

- (4) $x \neq y \vee \bar{Q}x \vee Qy$

and, say (1), the intermediate result

- (5) $a \neq y \vee Qy$,

before passing from (5) and (2) to (3). By contrast, approach 3 would allow us to go directly from (1) and (2) to (3) without ever inferring the intermediate result (5). Thus approach 3 contributes to *brevity of proofs*. More important for proof search, it contributes (by means of immediacy) to *brevity of deduction chains* within proofs.

A second, and perhaps more serious disadvantage of approach 1 as compared to approach 3, is that the intermediate debris such as step (5) tends to spawn increasingly larger generations of generally useless offspring, polluting the search space badly. We describe this difference by saying that approach 3 tends to be more *convergent* than approach 1. (Presence of various

subsidiary strategies, such as set-of-support, may possibly mitigate the severity of such non-convergence effects.)

The third disadvantage of approach 1 is perhaps the least important, although superficially the most obvious: the equality axioms $E(K)$ must be present. The clerical chore of writing them all down could be eliminated merely by incorporating into the theorem-prover a program to generate them. Alternatively they may be specified by means of a schema (we shall call this variation approach 1b), or in approach 2 by means of a few second-order axioms. We feel that this third disadvantage is so superficial and trivial (since one can simply place $E(K)$ outside the set of support as is done in the standard set-of-support variant of approach 1) as to be quite spurious.

The method given by Darlington (1968), whether it be classed as approach 1b or as approach 2, can be taken as typical of methods which avoid the third disadvantage (greater number of explicit axioms) but fail to dent the first and second disadvantages (longer deduction chains and non-convergence). In effect Darlington infers (5) from (1) and

$$(4') x \neq y \vee \varphi(x) \vee \varphi(y),$$

which is thought of either as a schema defining a set of first-order axioms including (4), or as a single second-order axiom having (4) as an instance.

PARAMODULATION

Since our automatic theorem-proving environment consists exclusively of clauses, we should like our rule of inference for equality to operate on two clauses and yield a clause. Furthermore, we should like it to apply to units and non-units alike¹ and to yield a most general clause that can be R -soundly inferred. We shall now describe the inference rule for paramodulation, which is asserted to have these properties. Examples of paramodulation are given in figure 2.2

Paramodulation: Given clauses A and $\alpha' = \beta' \vee B$ (or $\beta' = \alpha' \vee B$) having no variable in common and such that A contains a term δ , with δ and α' having a most general common instance α identical to $\alpha'[s_i/u_i]$ and to $\delta[t_j/w_j]$, form A' by replacing in $A[t_j/w_j]$ some single occurrence of α (resulting from an occurrence of δ)³ by $\beta'[s_i/u_i]$, and infer $A' \vee B[s_i/u_i]$.⁴

¹ Consider for example the set $S = \{c = d \vee \bar{Q}c, g(c) \neq g(d) \vee \bar{Q}c, a = b \vee Qc, g(a) \neq g(b) \vee Qc, x = x\}$. If the rule applied only to units, it would not be possible to this R -unsatisfiable set.

² These examples are primarily to give an intuitive idea of how paramodulation works. A comparison of the length and complexity of paramodulation proofs against resolution proofs can be obtained by considering the proofs of the theorem from group theory to the effect that $x^3 = e$ implies $((x, y), y) = e$. The resolution proof is 136 steps long while the paramodulation proof is 47 steps long. These proofs appear in the appendix.

³ Without this restriction one could infer from $a = b$ and $Qxa \vee Px$ the clause $Qab \vee Pa$ (a proper instance of the paramodulant $Qxb \vee Px$), resulting in a loss of generality.

⁴ Since every non-trivial immediate modulant (see Wos *et al.*, 1967b) of a clause is a paramodulant, any clause obtained by demodulation can be obtained by repeated paramodulation.

THEOREM PROVING

<p><i>Example 1</i></p> <ol style="list-style-type: none"> 1. $a=b$ 2. Qa 3. $\therefore Qb$ <p><i>Example 5</i></p> <ol style="list-style-type: none"> 1. $x=h(x)$ 2. $Qg(y)$ 3. $\therefore Qh(g(y))$ 	<p><i>Example 2</i></p> <ol style="list-style-type: none"> 1. $a=b$ 2. Qx 3. $\therefore Qb$ <p><i>Example 6</i></p> <ol style="list-style-type: none"> 1. $a=b$ 2. $Qf(g(h(j(a))))$ 3. $\therefore Qf(g(h(j(b))))$ 	<p><i>Example 3</i></p> <ol style="list-style-type: none"> 1. $a=b$ 2. $Qx \vee Px$ 3. $\therefore Qb \vee Pa$ <p><i>Example 7</i></p> <ol style="list-style-type: none"> 1. $f(xg(x))=e$ 2. $Pyf(g(y)z)z$ 3. $\therefore Pyeg(g(y))$ 	<p><i>Example 4</i></p> <ol style="list-style-type: none"> 1. $a=b$ 2. $Qx \vee Px$ 3. $\therefore Qa \vee Pb$
---	---	---	--

Example 8. If $x^2=e$ for all x in a group, the group is commutative.

<ol style="list-style-type: none"> 1. $f(ex)=x$ 2. $f(xe)=x$ 3. $f(xf(yz))=f(f(xy)z)$ 4. $f(xx)=e$ 5. $f(ab)=c$ 6. $c \neq f(ba)$ 7. $f(xe)=f(f(xy)y)$ 8. $x=f(f(xy)y)$ 9. $a=f(cb)$ 10. $f(yf(yz))=f(ez)$ 11. $f(yf(yz))=z$ 12. $f(ca)=b$ 13. $c=f(ba)$ 14. \square 	<ol style="list-style-type: none"> 4 into 3 with $\delta: f(yz)$ 2 into 7 on $f(xe)$ 5 into 8 on $f(xy)$ 4 into 3 on $f(xy)$ 1 into 10 on $f(ez)$ 9 into 11 on $f(yz)$ 12 into 8 on $f(xy)$ 13 resolved with 6
--	---

Figure 2

From a superficial point of view, paramodulation might be described as 'a substitution rule for equality'. Indeed, the motivation given above for studying the rule has dwelt principally on that aspect of paramodulation. But to consider it as only substitution of equals for equals would be to make a mistake analogous to characterizing resolution as merely syllogistic inference akin to that employed by Davis and Putnam (1960). The property of maximum generality provided by paramodulation must not be overlooked if the process is to be fully understood. Consider the following example:

From $f(xg(x))=e \vee Qx$ and $Pyf(g(y)z)z \vee Wz$ one can infer $Pyeg(g(y)) \vee Qg(y) \vee Wg(g(y))$ by paramodulating with $f(xg(x))$ as α' and $f(g(y)z)$ as δ .

COMPLETENESS OF PARAMODULATION FOR BASIC GROUP THEORY

Consider the following clauses from the first-order theory of groups:

- | | | |
|----|---|------------------------|
| A1 | $Pxyf(xy)$ | closure |
| A2 | $Pexx$ | left identity |
| A3 | $Pg(x)xe$ | left inverse |
| A4 | $\bar{P}xyu \vee \bar{P}yzv \vee \bar{P}uzw \vee Pxyvw$ | associativity (case 1) |

A5	$\bar{P}xyz \vee \bar{P}xyu \vee z=u$	uniqueness of product
A6	$z \neq u \vee \bar{P}xyz \vee Pxyu$	substitution (3rd position)
A7	$z \neq u \vee \bar{P}xzy \vee Pxuy$	substitution (2nd position)
A8	$z \neq u \vee \bar{P}zxy \vee Puxy$	substitution (1st position)
A9	$x=x$	reflexivity
A10	$x \neq y \vee y=x$	symmetry
A11	$x \neq y \vee y \neq z \vee x=z$	transitivity
A12	$x \neq y \vee f(xz)=f(yz)$	f -substitution (1st position)
A13	$x \neq y \vee f(zx)=f(zy)$	f -substitution (2nd position)
A14	$x \neq y \vee g(x)=g(y)$	g -substitution

Let us define a *basic* set S of clauses of group theory to be a set over the vocabulary of A1–A14 and such that $S \vdash \{A1, \dots, A5\}$. We then have the following completeness result for the special case of basic sets.

Theorem: If S is a satisfiable, fully paramodulated, fully factored, basic set of clauses of group theory, then S is R -satisfiable.

Proof: Let M be a maximal model¹ of S . Suppose that $\alpha=\beta$ and $P\gamma\delta\alpha$ are both in M . By the maximality of M , there must be clauses A and B in S having instances $A':\alpha=\beta \vee K$ and $B':P\gamma\delta\alpha \vee L$ with $K \cap M = \phi = L \cap M$. Then factors of A and B can be paramodulated on the arguments corresponding to α to give a clause in S having $P\gamma\delta\hat{\beta} \vee K \vee L$ as an instance. Since M satisfies S , $(P\gamma\delta\hat{\beta} \vee K \vee L) \cap M \neq \phi$. But $(K \vee L) \cap M = \phi$. Hence $P\gamma\delta\hat{\beta} \in M$. Thus M satisfies A6. It can be shown² that A1–A6 \vdash A7–A14. Hence M satisfies A6–A14 and is therefore an R -model of S .

This result is generalized to the case of what will be called functionally-reflexive systems in the next section.

COMPLETENESS OF PARAMODULATION FOR FUNCTIONALLY-REFLEXIVE SYSTEMS

Paramodulation is intended to be utilized, along with resolution, for theorem-proving in first-order theories with equality.³

We first give an algorithm for generating a refutation (of a finite set of clauses) employing paramodulation and resolution if such a refutation exists.

Full Search Algorithm (FSA): Let S_0 be the set of all factors of the given set S of clauses⁴. For odd $i > 0$ let S_i be formed from S_{i-1} by adding all clauses

¹ The concept of maximal model is defined and the pertinent existence theorem proved in Wos and Robinson (1968a). For the present purpose a maximal model of S may be thought of as a model M such that for each positive literal x in M there is an instance C' of some C in S with $C' \cap M = \{x\}$.

² Robinson and Wos (1967c).

³ The earliest formulations of paramodulation were designed to operate without resolution and could be shown to subsume resolution as a special case. It is felt, however, that the processes can be better understood if the inference apparatus not involving equality is isolated from the apparatus for equality, even if this means that some of the completeness theorems cannot be stated in quite as pat a fashion.

⁴ Every clause is a factor of itself as in G. Robinson *et al.* (1964b). For further definitions of factoring and resolution see Wos *et al.* (1964a) and J. Robinson (1965).

THEOREM PROVING

that can be obtained by paramodulating two clauses in S_{i-1} . For even $i > 0$ let S_i be formed from S_{i-1} by adding all factors of clauses that can be obtained by resolving two clauses in S_{i-1} . Since each deduction from S is contained in S_n for some n , each refutation of S must be contained in S_n for some n . Each S_j is finite. If S_j contains \square , a refutation has been found, so stop. Otherwise form S_{j+1} .

Now, to prove that paramodulation and resolution are complete for theorem-proving in first-order theories with equality, we would like to show that *FSA* is a semi-decision procedure for *R*-unsatisfiability. The difficult part is to show that, for *R*-unsatisfiable sets of clauses, there *exists* a refutation, namely, that paramodulation plus resolution is *R*-refutation complete. It will suffice to show that an unsatisfiable set can be deduced from an *R*-unsatisfiable set, since (due to the refutation-completeness of resolution) *FSA* will generate a refutation if it ever generates an unsatisfiable set.

A functionally-reflexive system S is defined as one for which $S \vdash x_1 = x_1$ and $S \vdash f(x_1, \dots, x_n) = f(x_1, \dots, x_n)$ for every function letter f occurring in the vocabulary of S , n being the degree of f . There are $h + 1$ such unit clauses, where h is the number of function letters in the vocabulary of S . For such systems refutation-completeness is proved in Wos and Robinson (1968c).¹ From that result one can obtain the following corollary: If S is a finite functionally-reflexive set of clauses, *FSA* is a semidecision procedure for *R*-unsatisfiability.

Even for theories that do not happen to be functionally reflexive, this result shows that adding the $h + 1$ functional-reflexivity unit clauses before applying *FSA* gives a general semi-decision procedure for *R*-unsatisfiability.

FURTHER COMPLETENESS RESULTS FOR PARAMODULATION

Since first-order theories are not usually functionally-reflexive when the only rules are resolution and paramodulation, and since adding the functional-reflexivity units to the theory may detract somewhat from proof-search efficiency, one would wish to show that some weaker assumption than functional-reflexivity will suffice for completeness. It seems that at least $S \vdash x = x$ will be needed. (Consider the case where S consists of $\{a \neq a\}$. S is *R*-unsatisfiable but cannot be refuted without some sort of help from reflexivity.) This is not surprising, since the standard texts on logic that use the substitution rule or schema approach to equality consistently supply a separate reflexivity axiom.²

But is simple reflexivity ($x = x$) enough? We think so,³ although a proof of this is not yet available.

¹ A weaker version of this result was given in the earlier (1968b) paper.

² See, e.g., Church (1956) or Quine (1963).

³ In the two years that paramodulation has been under study, no counterexample has been found to the *R*-refutation completeness of paramodulation and resolution for simply-reflexive systems.

To see where the difficulty lies in generalizing the proof given in Wos and Robinson (1968c) beyond the functionally-reflexive case, we examine the relation between deductions and refutations based on a given set S and those based on proper instances of clauses from S .

*Capturing lemma*¹: Let S be a fully paramodulated and fully resolved set of clauses such that $S \vdash x=x$, and let A' and B' be instances of clauses A and B in S and let C' be the result of paramodulating from a term α' in A' into an occurrence δ_0 of a term in B' . Then

Strong subterm form: There is a clause C in S with C' as an instance.

Restricted subterm form: If B has a term in the same position as that of δ_0 in B' , then there is a clause C in S with C' as an instance.

(Occurrences of terms in two literals are said to be in the *same position* if each is the i_1 -st argument of the i_2 -nd argument of . . . of the i_n -th argument of its literal.)

Argument form: If δ is an argument of B' (as opposed to a proper subterm of an argument), then there is a clause C in S with C' as an instance.

When the strong subterm form of the capturing lemma holds and $S \vdash x=x$, every maximal model (with respect to positive literals) of S is an R -model, and since every satisfiable set S has a maximal model, it follows that either $\square \in S$ or S is R -satisfiable. Thus the strong subterm form of the capturing lemma and simple reflexivity imply R -refutation-completeness. The line of proof given for R -refutation-completeness in functionally-reflexive systems in (1968c) depends (at least indirectly) on the strong subterm form, which happens to hold in such systems.² The following example will suffice to show however that the strong subterm form is not universally true:

$S: \{x=x, a=b, b=a, a=a, b=b, Qxg(x), Qag(a), Qbg(b),$
 $Qag(b), Qbg(a)\}$

$A: a=b$

$A': a=b$

$B: Qxg(x)$

$B': Qg(a)g(g(a))$

$C': Qg(b)g(g(a))$

S is fully paramodulated and (vacuously) fully resolved. A' and B' paramodulate on a into the first occurrence of a in B' to give C' . But C' is an instance of no clause in S . (The restricted subterm form of the lemma is not violated since B has no term in the same position as the first occurrence of a in B' . Neither is the argument form of the lemma, since a is not an argument

¹ The analogue of this capturing lemma for resolution alone plays a basic role in proving the refutation-completeness of resolution (see J. Robinson, 1965 and Slagle, 1967) and of set-of-support (Wos *et al.*, 1965).

² Alternatively, one can view the difficulty as resulting from the fact that it is not always possible to satisfy the hypotheses of the restricted subterm form.

THEOREM PROVING

of B' .) Functional-reflexivity of S , if present, would dispose of the difficulty, since, if $g(x) = g(x)$ were in S , so would $g(a) = g(b)$ be in S if it were fully paramodulated; and hence the result $Qg(b)g(g(a))$ of paramodulating $g(a) = g(b)$ and $Qxg(x)$ would be in S and serve as C .

Weakening the strong subterm capturing lemma in a different fashion leads to the

Refutation capturing lemma: If there exists a refutation of a set of instances of clauses in a set S by means of paramodulation and resolution, then there exists a refutation of S itself by means of paramodulation and resolution.

For functionally-reflexive S , this lemma may be proved by noting that the refutability of a set of instances of S and R -soundness of paramodulation and resolution yield the R -unsatisfiability of S ; so that the refutation-completeness of paramodulation and resolution for functionally-reflexive systems establishes the refutability of S itself.

Given the refutation capturing lemma one could prove the following:

General refutation-completeness: If S is a fully paramodulated and fully resolved R -unsatisfiable set and if $S \vdash x = x$, then $\square \in S$.

Corollary: FSA is a semi-decision procedure for R -unsatisfiability for finite sets S of clauses such that $S \vdash x = x$.

Conversely, given general refutation-completeness, one can prove the refutation capturing lemma (at least for systems S such that $S \vdash x = x$). In view of this equivalence, proof of the refutation capturing lemma can be considered the most pressing unsolved problem in the theory of paramodulation. Alternatively, one might seek a proof of general refutation-completeness based on the restricted subterm form of the capturing lemma, which holds even when the assumption of functional reflexivity is suppressed.

Acknowledgement

This work was supported by the US Atomic Energy Commission.

REFERENCES

- Church, A. (1956) *Introduction to mathematical logic I*. Princeton.
Darlington, J.L. (1968) Automatic theorem proving with equality substitutions and mathematical induction. *Machine Intelligence 3*, pp. 113-27 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
Davis, M. & Putnam, H. (1960) A computing procedure for quantification theory. *J. Assn. Comput. Mach.*, 7, 201-15.
Quine, W.V.O. (1963) *Set theory and its logic*. Cambridge, Mass: Harvard University Press.
Robinson, J.A. (1965) A machine-oriented logic based on the resolution principle. *J. Assn. Comput. Mach.*, 12, 23-41.
Slagle, J.R. (1967) Automatic theorem proving with renamable and semantic resolution. *J. Assn. Comput. Mach.*, 14, pp. 687-97.
Wos, L., Carson, D. & Robinson, G. (1964a) The unit-preference strategy in theorem proving. *AFIPS Conference Proceedings*, 26, Washington D.C.: Spartan Books, 615-21.

- Robinson, G. A., Wos, L. T. & Carson, D. (1964b) Some theorem-proving strategies and their implementation. *AMD Tech. Memo No. 72*, Argonne National Laboratory.
- Wos, L., Robinson, G. A. & Carson, D. F. (1965) Efficiency and completeness of the set of support strategy in theorem proving. *J. Assn. Comput. Mach.*, **12**, 536-41.
- Robinson, G. A., Wos, L. & Shalla, L. (1967a) Two inference rules for first-order predicate calculus with equality. *AMD Tech. Memo No. 142*, Argonne National Laboratory.
- Wos, L., Robinson, G. A., Carson, D. F. & Shalla, L. (1967b) The concept of demodulation in theorem proving. *J. Assn. Comput. Mach.*, **14**, 698-709.
- Robinson, G. & Wos, L. (1967c) Dependence of equality axioms in elementary group theory. *Comp. Group Tech. Memo No. 53*, Stanford Linear Accelerator Center.
- Wos, L. & Robinson, G. (1968a), The maximal model theorem. Spring 1968 meeting of Assn. for Symbolic Logic. Abstract to appear in *J. Symb. Logic*.
- Robinson, G. & Wos, L. (1968b) Completeness of paramodulation. Spring 1968 meeting of Assn. for Symbolic Logic. Abstract to appear in *J. Symb. Logic*.
- Wos, L. & Robinson, G. (1968c) Maximal models and refutation completeness (unpublished).

APPENDIX

Paramodulation versus resolution

Problem: $x^3 = e$ implies $((x, y), y) = e$ where $(x, y) = xyx^{-1}y^{-1}$

Reference: *Group Theory* by Marshall Hall, page 322, 18.2.8.

Refutation by Paramodulation

1. $f(ex) = x$
2. $f(xe) = x$
3. $f(g(x)x) = e$
4. $f(xg(x)) = e$
5. $f(xf(yz)) = f(f(xy)z)$
6. $x = x$
7. $f(f(xx)x) = e$
8. $h(xy) = f(f(f(xy)g(x))g(y))$
9. $h(h(ab)b) \neq e$
10. $f(xe) = f(f(xy)g(y)), f(xg(x))$ of 4 into $f(yz)$ of 5
11. $x = f(f(xy)g(y)), f(xe)$ of 2 into $f(xe)$ of 10
12. $x = f(eg(g(x))), f(xg(x))$ of 4 into $f(xy)$ of 11
13. $x = g(g(x)), f(ex)$ of 1 into $f(eg(g(x)))$ of 12
14. $f(f(xx)f(xz)) = f(ez), f(f(xx)x)$ of 7 into $f(xy)$ of 5
15. $f(f(xx)f(xz)) = z, f(ex)$ of 1 into $f(ez)$ of 14
16. $f(f(xx)e) = g(x), f(xg(x))$ of 4 into $f(xz)$ of 15
17. $f(xx) = g(x), f(xe)$ of 2 into $f(f(xx)e)$ of 16
18. $f(f(xy)f(g(y)z)) = f(xz), f(f(xy)g(y))$ of 11 into $f(xy)$ of 5
19. $f(f(xy)f(g(y)g(x))) = e, f(xg(x))$ of 4 into $f(xz)$ of 18
20. $f(we) = f(f(wf(xy))f(g(y)g(x))), f(f(xy)f(g(y)g(x)))$ of 19 into $f(yz)$ of 5
21. $w = f(f(wf(xy))f(g(y)g(x))), f(xe)$ of 2 into $f(we)$ of 20

THEOREM PROVING

22. $g(f(xy)) = f(ef(g(y)g(x))), f(g(x)x)$ of 3 into $f(wf(xy))$ of 21
23. $g(f(xy)) = f(g(y)g(x)), f(ex)$ of 1 into $f(ef(g(y)g(x)))$ of 22
24. $g(h(xy)) = f(g(g(y))g(f(f(xy)g(x))))$, $f(f(f(xy)g(x))g(y))$ of 8 into $f(xy)$ of 23
25. $g(h(xy)) = f(yf(f(f(xy)g(x))))$, $g(g(x))$ of 13 into $g(g(y))$ of 24
26. $g(h(xy)) = f(yf(g(g(x))g(f(xy))))$, $g(f(xy))$ of 23 into $g(f(f(xy)g(x)))$ of 25
27. $g(h(xy)) = f(yf(xg(f(xy))))$, $g(g(x))$ of 13 into $g(g(x))$ of 26
28. $g(h(xy)) = f(yf(xf(g(y)g(x))))$, $g(f(xy))$ of 23 into $g(f(xy))$ of 27
29. $f(f(f(h(ab)b)g(h(ab))))g(b) \neq e$, $h(xy)$ of 8 into $h(h(ab)b)$ of 9
30. $f(f(f(f(f(ab)g(a))g(b))b)g(h(ab)))g(b) \neq e$, $h(xy)$ of 8 into $h(ab)$ of 29
31. $f(f(f(f(f(ab)g(a))f(g(b)b))g(h(ab))))g(b) \neq e$, $f(f(xy)z)$ of 5 into $f(f(f(f(ab)g(a))g(b))b)$ of 30
32. $f(f(f(f(f(ab)g(a))e)g(h(ab))))g(b) \neq e$, $f(g(x)x)$ of 3 into $f(g(b)b)$ of 31
33. $f(f(f(f(ab)g(a))g(h(ab))))g(b) \neq e$, $f(xe)$ of 2 into $f(f(f(ab)g(a))e)$ of 32
34. $f(f(f(f(ab)g(a))f(bf(af(g(b)g(a))))))g(b) \neq e$, $g(h(xy))$ of 28 into $g(h(ab))$ of 33
35. $f(f(f(f(ab)f(aa))f(bf(af(g(b)g(a))))))g(b) \neq e$, $g(x)$ of 17 into $g(a)$ of 34
36. $f(f(f(f(f(ab)f(aa))b)f(af(g(b)g(a))))g(b) \neq e$, $f(xf(yz))$ of 5 into $f(f(f(ab)f(aa))f(bf(af(g(b)g(a)))))$ of 35
37. $f(f(f(f(f(f(ab)f(aa))b)a)f(g(b)g(a)))g(b) \neq e$, $f(xf(yz))$ of 5 into $f(f(f(f(ab)f(aa))b)f(af(g(b)g(a))))$ of 36
38. $f(f(f(f(f(f(f(ab)a)a)b)a)f(g(b)g(a)))g(b) \neq e$, $f(xf(yz))$ of 5 into $f(f(ab)f(aa))$ of 37
39. $f(f(f(f(f(f(ab)a)f(ab))a)f(g(b)g(a)))g(b) \neq e$, $f(f(xy)z)$ of 5 into $f(f(f(f(ab)a)a)b)$ of 38
40. $f(f(f(f(f(ab)a)f(f(ab)a))f(g(b)g(a)))g(b) \neq e$, $f(f(xy)z)$ of 5 into $f(f(f(f(ab)a)f(ab))a)$ of 39
41. $f(f(f(f(ab)a)f(f(ab)a))f(f(g(b)g(a))g(b))) \neq e$, $f(f(xy)z)$ of 5 into $f(f(f(f(f(ab)a)f(f(ab)a))f(g(b)g(a)))g(b))$ of 40
42. $f(f(f(f(ab)a)f(f(ab)a))f(g(f(ab))))g(b) \neq e$, $f(g(y)g(x))$ of 23 into $f(g(b)g(a))$ of 41
43. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(ab)f(ab))g(b))) \neq e$, $g(x)$ of 17 into $g(f(ab))$ of 42
44. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(f(ab)a)b)g(b))) \neq e$, $f(xf(yz))$ of 5 into $f(f(ab)f(ab))$ of 43
45. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(ab)a)f(bg(b)))) \neq e$, $f(f(xy)z)$ of 5 into $f(f(f(f(ab)a)b)g(b))$ of 44
46. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(ab)a)e)) \neq e$, $f(xg(x))$ of 4 into $f(bg(b))$ of 45

47. $f(f(f(f(ab)a)f(f(ab)a))f(f(ab)a)) \neq e$, $f(xe)$ of 2 into
 $f(f(f(f(ab)a)f(f(ab)a))f(f(f(ab)a)e))$ of 46
 7 contradicts 47.

Paramodulation versus resolution

Problem: $x^3=e$ implies $((x,y),y)=e$.

Refutation by Resolution

1. $f(ex) = x$
2. $f(xe) = x$
3. $f(e(x)x) = e$
4. $f(xg(x)) = e$
5. $f(xf(yz)) = f(f(xy)z)$
6. $x = x$
7. $x \neq y \quad y = x$
8. $x \neq y \quad y \neq z \quad x = z$
9. $u \neq w \quad f(ux) = f(wx)$
10. $u \neq w \quad f(xu) = f(xw)$
11. $u \neq w \quad g(u) = g(w)$
12. $f(f(xx)x) = e$
13. $h(xy) = f(f(f(xy)g(x))g(y))$
14. $h(h(ab)b) \neq e$
15. $x \neq f(ew) \quad x = w$, 1 and 8_2
16. $f(f(xg(x))w) = f(ew)$, 4 and 9_1
17. $f(f(xy)z) \neq w \quad f(xf(yz)) = w$, 5 and 8_1
18. $f(xf(g(x)z)) = f(ez)$, 16 and 17_1
19. $f(xf(g(x)z)) = z$, 18 and 15_1
20. $f(uf(yg(y))) = f(ue)$, 4 and 10_1
21. $f(ue) = f(uf(yg(y)))$, 20 and 7_1
22. $f(uf(yg(y))) \neq z \quad f(ue) = z$, 21 and 8_1
23. $f(xe) = g(g(x))$, 19 and 22_1
24. $x = f(xe)$, 2 and 7_1
25. $f(xe) \neq z \quad x = z$, 24 and 8_1
26. $x = g(g(x))$, 23 and 25_1
27. $f(f(f(uu)u)y) = f(ey)$, 12 and 9_1
28. $f(f(f(uu)u)y) = y$, 27 and 15_1
29. $f(f(xx)f(xy)) = y$, 28 and 17_1
30. $f(f(xx)e) = g(x)$, 29 and 22_1
31. $f(xx) = g(x)$, 30 and 25_1
32. $f(xe) = f(f(xy)g(y))$, 5 and 22_1
33. $x = f(f(xy)g(y))$, 32 and 25_1
34. $f(xz) = f(f(f(xy)g(y))z)$, 33 and 9_1
35. $f(f(f(xy)g(y))z) = f(xz)$, 34 and 7_1
36. $f(f(xy)f(g(y)z)) = f(xz)$, 35 and 17_1

THEOREM PROVING

37. $x \neq f(ug(u)) \quad x=e, 4 \text{ and } 8_2$
38. $f(f(xy)f(g(y)g(x)))=e, 36 \text{ and } 37_1$
39. $e=f(f(xy)f(g(y)g(x))), 38 \text{ and } 7_1$
40. $f(we)=f(wf(f(xy)f(g(y)g(x))))$, 39 and 10₁
41. $u \neq f(xf(yz)) \quad u=f(f(xy)z), 5 \text{ and } 8_2$
42. $f(ue)=f(f(uf(xy))f(g(y)g(x))), 40 \text{ and } 41_1$
43. $u=f(f(uf(xy))f(g(y)g(x))), 42 \text{ and } 25_1$
44. $f(f(g(x)x)u)=f(eu), 3 \text{ and } 9_1$
45. $z \neq f(f(g(x)x)u) \quad z=f(eu), 44 \text{ and } 8_2$
46. $g(f(xy))=f(ef(g(y)g(x))), 43 \text{ and } 45_1$
47. $g(f(xy))=f(g(y)g(x)), 46 \text{ and } 15_1$
48. $g(h(xy))=g(f(f(f(xy)g(x))g(y))), 13 \text{ and } 11_1$
49. $u \neq g(f(xy)) \quad u=f(g(y)g(x)), 47 \text{ and } 8_2$
50. $g(h(xy))=f(g(g(y))g(f(f(xy)g(x))))$, 48 and 49₁
51. $g(g(x))=x, 26 \text{ and } 7_1$
52. $f(g(g(u)z)=f(uz), 51 \text{ and } 9_1$
53. $x \neq f(g(g(u)z)=f(uz), 52 \text{ and } 8_2$
54. $g(h(xy))=f(yg(f(f(xy)g(x))))$, 50 and 53₁
55. $f(zg(f(xy)))=f(zf(g(y)g(x))), 47 \text{ and } 9_1$
56. $u \neq f(zg(f(xy))) \quad u=f(zf(g(y)g(x))), 55 \text{ and } 8_2$
57. $g(h(xy))=f(yf(g(g(x))g(f(xy))))$, 54 and 56₁
58. $f(yf(g(g(u)z))=f(yf(uz)), 52 \text{ and } 10_1$
59. $x \neq f(yf(g(g(u)z)) \quad x=f(yf(uz)), 58 \text{ and } 8_2$
60. $g(h(xy))=f(yf(xg(f(xy))))$, 57 and 59₁
61. $f(uf(zg(f(xy))))=f(uf(zf(g(y)g(x))))$, 55 and 10₁
62. $w \neq f(uf(zg(f(xy)))) \quad w=f(uf(zf(g(y)g(x))))$, 61 and 8₂
63. $g(h(xy))=f(yf(xf(g(y)g(x))))$, 60 and 62₁
64. $f(zg(h(xy)))=f(zf(yf(xf(g(y)g(x))))$, 60 and 62₁
65. $f(wf(zg(h(xy))))=f(wf(zf(yf(xf(g(y)g(x))))$, 64 and 10₁
66. $f(uf(wf(zg(h(xy))))=f(uf(wf(zf(yf(xf(g(y)g(x))))$, 65 and 10₁
67. $f(uf(wf(zg(h(xy))))=f(f(uw)f(zf(yf(xf(g(y)g(x))))$, 66 and 41₁
68. $f(uf(wf(zg(h(xy))))=f(f(f(uw)z)f(yf(xf(g(y)g(x))))$, 67 and 41₁
69. $f(f(xy)z)=f(xf(yz)), 5 \text{ and } 7_1$
70. $f(xf(yz)) \neq u \quad f(f(xy)z)=u, 69 \text{ and } 8_1$
71. $f(f(uw)f(zg(h(xy))))=f(f(f(uw)z)f(yf(xf(g(y)g(x))))$, 68 and 70₁
72. $f(f(f(uw)z)g(h(xy)))=f(f(f(uw)z)f(yf(xf(g(y)g(x))))$, 71 and 70₁
73. $f(f(f(f(xy)z)g(h(xy)))u)=f(f(f(f(xy)z)f(yf(xf(g(y)g(x))))u)$, 72 and 9₁
74. $f(h(xy)z)=f(f(f(f(xy)g(x))g(y))z)$, 13 and 9₁
75. $u \neq f(f(xy)z) \quad u=f(xf(yz)), 69 \text{ and } 8_2$
76. $f(h(xy)z)=f(f(f(xy)g(x))f(g(y)z))$, 74 and 75₁
77. $f(uf(g(x)x))=f(ue), 3 \text{ and } 10_1$
78. $z \neq f(uf(g(x)x)) \quad z=f(ue), 77 \text{ and } 8_2$

79. $f(h(xy)y) = f(f(f(xy)g(x))e)$, 76 and 78₁
 80. $u \neq f(xe) \quad u = x$, 2 and 8₂
 81. $f(h(xy)y) = f(f(xy)g(x))$, 79 and 80₁
 82. $f(f(h(xy)y)z) = f(f(f(xy)g(x))z)$, 81 and 9₁
 83. $f(f(f(h(xy)y)z)w) = f(f(f(f(xy)g(x))z)w)$, 82 and 9₁
 84. $h(h(ab)b) \neq y \quad y \neq e$, 14 and 8₂
 85. $f(f(f(h(ab)b)g(h(ab)))g(b)) \neq e$, 13 and 84₁
 86. $f(f(f(h(ab)b)g(h(ab)))g(b)) \neq y \quad y \neq e$, 85 and 8₃
 87. $f(f(f(f(ab)g(a))g(h(ab)))g(b)) \neq e$, 83 and 86₁
 88. $f(f(f(f(ab)b(a))b(h(ab)))g(b)) \neq y \quad y \neq e$, 87 and 8₃
 89. $f(f(f(f(ab)g(a))f(bf(af(g(b)g(a))))g(b)) \neq e$, 73 and 88₁
 90. $g(x) = f(xx)$, 31 and 7₁
 91. $f(wg(x)) = f(wf(xx))$, 90 and 10₁
 92. $f(uf(wg(x))) = f(uf(wf(xx)))$, 91 and 10₁
 93. $f(uf(wg(x))) = f(f(uw)f(xx))$, 92 and 41₁
 94. $f(f(uw)g(x)) = f(f(uw)f(xx))$, 93 and 70₁
 95. $f(f(f(uw)g(x))y) = f(f(f(uw)f(xx))y)$, 94 and 9₁
 96. $f(f(f(f(uw)g(x))y)z) = f(f(f(f(uw)f(xx))y)z)$, 95 and 9₁
 97. $f(f(f(f(ab)g(a))f(bf(af(g(b)g(a))))g(b)) \neq y \quad y \neq e$, 89 and 8₃
 98. $f(f(f(f(ab)f(aa))f(bf(af(g(b)g(a))))g(b)) \neq e$, 96 and 97₁
 99. $f(f(f(f(ab)f(aa))f(bf(af(g(b)g(a)))) \neq y \quad y \neq e$, 98 and 8₃
 100. $f(f(xf(yz))u) = f(f(f(xy)z)u)$, 5 and 9₁
 101. $f(f(f(f(f(ab)f(aa))b)f(af(g(b)g(a))))g(b)) \neq e$, 100 and 99₁
 102. $f(f(f(f(f(ab)f(aa))b)f(af(g(b)g(a))))g(b)) \neq y \quad y \neq e$, 101 and 8₃
 103. $f(f(f(f(f(f(ab)f(aa))b)a)f(g(b)g(a)))g(b)) \neq e$, 100 and 102₁
 104. $f(f(f(xf(yz))u)v) = f(f(f(f(xy)z)u)v)$, 100 and 9₁
 105. $f(f(f(f(xf(yz))u)v)w) = f(f(f(f(f(xy)z)u)v)w)$, 104 and 9₁
 106. $f(f(f(f(f(xf(yz))u)v)w)t) = f(f(f(f(f(f(xy)z)u)v)w)t)$, 105 and 9₁
 107. $f(f(f(f(f(f(ab)f(aa))b)a)f(g(b)g(a)))g(b)) \neq y \quad y \neq e$, 103 and 8₃
 108. $f(f(f(f(f(f(ab)a)a)b)a)f(g(b)g(a)))g(b)) \neq e$, 106 and 107₁
 109. $f(f(f(f(f(xy)z)u)v)w) = f(f(f(f(xf(yz))u)v)w)$, 105 and 7₁
 110. $f(f(f(f(f(f(ab)a)a)b)a)f(g(b)g(a)))g(b)) \neq y \quad y \neq e$, 108 and 8₃
 111. $f(f(f(f(f(f(ab)a)f(ab))a)f(g(b)g(a)))g(b)) \neq e$, 109 and 110₁
 112. $f(f(f(f(xy)z)u)v) = f(f(f(xf(yz))u)v)$, 104 and 7₁
 113. $f(f(f(f(f(f(ab)a)f(ab))a)f(g(b)g(a)))g(b)) \neq y \quad y \neq e$, 111 and 8₃
 114. $f(f(f(f(f(ab)a)f(f(ab)a))f(g(b)g(a)))g(b)) \neq e$, 112 and 113₁
 115. $f(f(f(f(ab)a)f(f(ab)a))f(f(g(b)g(a))g(b)) \neq e$, 114 and 70₂
 116. $f(f(f(f(ab)a)f(f(ab)a))f(f(g(b)g(a))g(b)) \neq y \quad y \neq e$, 115 and 8₃
 117. $f(g(y)g(x))g(f(xy))$, 47 and 7₁
 118. $f(f(g(y)g(x))z) = f(g(f(xy))z)$, 117 and 9₁
 119. $f(uf(f(g(y)g(x))z)) = f(uf(g(f(xy))z))$, 118 and 10₁
 120. $f(f(f(f(ab)a)f(f(ab)a))f(g(f(ab))g(b))) \neq e$, 119 and 116₁
 121. $f(g(x)z) = f(f(xx)z)$, 90 and 9₁
 122. $f(uf(g(x)z)) = f(uf(f(xx)z))$, 121 and 10₁

THEOREM PROVING

123. $f(f(f(f(ab)a)f(f(ab)a))f(g(f(ab))g(b))) \neq y \quad y \neq e$, 120 and 8_3
 124. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(ab)f(ab))g(b))) \neq e$, 122 and 123_1
 125. $f(wf(f(xf(yz))u)) = f(wf(f(f(xy)z)u))$, 100 and 10_1
 126. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(ab)f(ab))g(b))) \neq y \quad y \neq e$, 124 and 8_3
 127. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(f(ab)a)b)g(b))) \neq e$, 125 and 126_1
 128. $f(uf(f(xy)z)) = f(uf(xf(yz)))$, 69 and 10_1
 129. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(f(ab)a)b)g(b))) \neq y \quad y \neq e$, 127 and 8_3
 130. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(ab)a)f(bg(b)))) \neq e$, 128 and 129_1
 131. $f(zf(uf(yg(y)))) = f(zf(ue))$, 20 and 10_1
 132. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(ab)a)f(bg(b)))) \neq y \quad y \neq e$, 130 and 8_3
 133. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(a)b)a)e)) \neq e$, 131 and 132_1
 134. $f(uf(xe)) = f(ux)$, 2 and 10_1
 135. $f(f(f(f(ab)a)f(f(ab)a))f(f(f(a)b)a)e)) \neq y \quad y \neq e$, 133 and 8_3
 136. $f(f(f(f(ab)a)f(f(ab)a))f(f(ab)a)) \neq e$, 134 and 135_1

12 contradicts 136

Notes added in proof

1. In this paper we intend *fully resolved* sets to be fully factored also.
2. The reader may wish to note that in subsequent work we reserve the term *general* for clauses or terms and use *conservative* instead of *general* for inference systems, in order to avoid possible confusion arising from some misleading connotations of *general* when used in connection with inference systems.
3. A critical difference between functional-reflexive systems defined here as well as in Wos and Robinson (1968c) and those treated in Robinson and Wos (1968b), is that only $h+1$ functional-reflexivity unit clauses are required, where h is the number of function letters in the vocabulary of S ; whereas arbitrarily many instances of reflexivity may be required to satisfy the earlier, weaker completeness result.

Mechanizing Higher-Order Logic

J. A. Robinson

College of Liberal Arts
Syracuse University

The purpose of this paper is to describe a simple but extremely powerful system of higher-order logic; to describe a proof procedure for this system; and to discuss the implementation of the proof procedure on the computer.

The formalism of the system is essentially that of the lambda-calculus, from which it derives two of its three basic principles. These two principles are the *application* of a function to an object, and the *abstraction* of a function from an expression describing an object. The third basic principle of the system is the classification, into so-called *types*, of all objects in the universe of discourse and all expressions in the system. The reason for this classification is semantic. When the expressions of the system are interpreted so as to denote objects, an expression A denotes an object B only if A and B are of the same type. For example, it is not always meaningful to apply an expression F (denoting a function) to an expression A (denoting an object) to form the expression (FA) . The expression (FA) , when meaningful, denotes the object yielded by applying the function denoted by F to the object denoted by A . However, (FA) is meaningful only when the expressions F and A are of appropriate types. In particular the expression (FF) is never meaningful. The expression (λXA) is always meaningful whenever the expression A is meaningful and the expression X is an identifier. It denotes the function whose value, at the object B , is 'the object denoted by A when X denotes B '. (The remarks of this paragraph are entirely informal and intuitive, by way of a brief introductory sketch.)

The proof procedure for this formalism is exceedingly transparent and easy to program. It is put forward as a specific, concrete starting-point for future investigations, and as an immediately useful computing scheme when implemented in a man-machine interactive form such as that suggested in the sequel.

SYNTAX

We begin by describing the syntax of the formalism. As shown in figure 1, the formulae of the system fall into two categories; the type symbols and the

THEOREM PROVING

expressions. We shall denote the set of all type symbols by T , and the set of all expressions by E .

The rôle of the type symbols in the system is the purely auxiliary one of serving as part of the machinery necessary for classifying the expressions and the objects into types. This machinery is quite simple. There is a mapping L defined for all expressions in E and all objects in the universe of discourse H ,

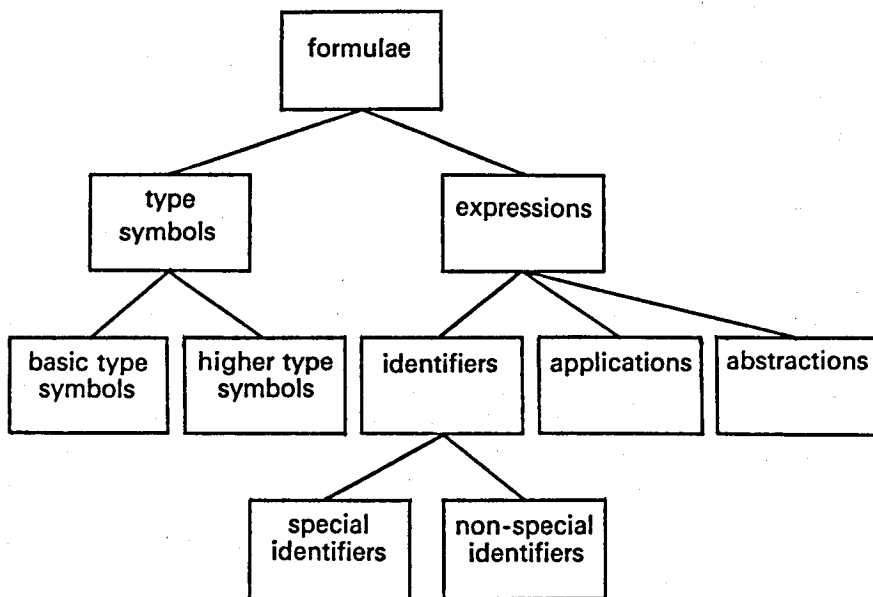


Figure 1. Syntax.

which takes its values in T . One thinks of L as a 'labelling' of the expressions and the objects with 'labels' taken from the set T . If α is a type symbol, and A is an object or an expression for which $LA = \alpha$, we say that A is 'of type α '. In order to explain how L is defined, we must first describe the set T of type symbols. Before we do so, we introduce a very convenient piece of notation. For each type symbol α , and each set S containing objects or expressions or both, we write $S\alpha$ to denote the set of things in S which are of type α . Thus, $E\alpha$ is the set of expressions of type α , and $H\alpha$ is the set of objects of type α .

Type symbols

The type symbols are of two kinds; basic type symbols and higher type symbols. The basic type symbols are simply identifiers, which can be freely chosen for their mnemonic properties, e.g., *truthvalue*, *real*, *integer*. We shall assume that the basic type symbols always include the first of these, *truthvalue*.

The higher type symbols are then all and only those inscriptions which can be constructed by finitely many applications of the following rule, starting

with the basic type symbols: if α and β are both type symbols then $(\alpha \rightarrow \beta)$ is a type symbol.

For example, if the basic type symbols include *truthvalue*, *real*, and *integer*, then the higher type symbols will include:

$(\text{truthvalue} \rightarrow \text{truthvalue})$
 $(\text{real} \rightarrow \text{integer})$
 $((\text{integer} \rightarrow \text{real}) \rightarrow \text{truthvalue})$
 $((\text{real} \rightarrow \text{real}) \rightarrow (\text{real} \rightarrow \text{real}))$

The intuitive significance of the type symbols is that an object of type $(\alpha \rightarrow \beta)$ is a function from objects of type α to objects of type β .

When α is a basic type symbol, the objects of type α are called *individuals*.

Objects of type *truthvalue* are truth values. There are always precisely two of them, namely, *true* and *false*.

To illustrate these ideas: if *Hreal* is the set of real numbers and *Hinteger* is the set of non-negative integers, and if, for each α and β in T , $H(\alpha \rightarrow \beta)$ is the set of all functions from $H\alpha$ to $H\beta$, then the individuals are the truth values, the real numbers and the non-negative integers, and among the objects of higher type there will be *the closed unit interval*, i.e., that object of type $(\text{real} \rightarrow \text{truthvalue})$ whose value, at a real number x , is *true* if $0 \leq x \leq 1$ and *false* otherwise; *the exponential series*, i.e., that object of type $(\text{integer} \rightarrow (\text{real} \rightarrow \text{real}))$ whose value, at a non-negative integer n , is the function $(\lambda t(t^n/n!))$; and *the set of primes*, i.e., that object of type $(\text{integer} \rightarrow \text{truthvalue})$ whose value, at a non-negative integer n , is *true* if n is prime and *false* otherwise.

Expressions

There are three kinds of expressions; *identifiers*, *abstractions* and *applications*. We define each of these kinds of expression as follows, simultaneously specifying the type of each expression.

Identifiers. These are of two kinds, special and nonspecial. The nonspecial identifiers constitute a denumerably infinite set of inscriptions which may be freely chosen for their mnemonic properties, consistent with the constraint that no nonspecial identifier can be mistaken for a special identifier, an abstraction, or an application. The types of the nonspecial identifiers may be assigned arbitrarily provided that for each type symbol α there are infinitely many nonspecial identifiers which are of type α .

The special identifiers, together with their types, are the following:

true, false: of type *truthvalue*,
not: of type $(\text{truthvalue} \rightarrow \text{truthvalue})$,
or, and, implies: of type $(\text{truthvalue} \rightarrow (\text{truthvalue} \rightarrow \text{truthvalue}))$,
equal α : of type $(\alpha \rightarrow (\alpha \rightarrow \text{truthvalue}))$,
choice α : of type $((\alpha \rightarrow \text{truthvalue}) \rightarrow \alpha)$.

THEOREM PROVING

The last two lines above are intended to mean that for each type symbol α there is a special identifier of the appearance and type shown.

The intuitive significance of the special identifiers is intended to be suggested by their mnemonic properties. For each type symbol α the identifier *choice* α will always denote a *choice function* for objects of type α , namely, a function whose value, at an object S of type $(\alpha \rightarrow \text{truthvalue})$, is an object B of type α having the property that, if there are any objects for which S takes on the value *true*, then B is among those objects. Briefly, a choice function for objects of type α 'chooses' an object from each nonempty set of objects of type α , and assigns, to the empty set of objects of type α , some object (it does not matter which) of type α .

Abstractions. If X is a nonspecial identifier of type α and A is an expression of type β then $(\lambda X A)$ is an expression of type $(\alpha \rightarrow \beta)$ and is an *abstraction*.

Applications. If F is an expression of type $(\alpha \rightarrow \beta)$ and A is an expression of type α then (FA) is an expression of type β and is an *application*.

Parentheses may be omitted in nested applications, with the convention of 'association to the left'. Thus $((FA)B)$ may be written (FAB) ; and $((((FA)B)C))$ may be written $(FABC)$ and so on. However, the expression $(F(AB))$ must be written so, on pain of being otherwise misread as $((FA)B)$.

Parentheses may also be omitted in nested abstractions with the convention of 'association to the right'. Thus $(\lambda X(\lambda Y A))$ may be written $(\lambda X \lambda Y A)$; $(\lambda X(\lambda Y(\lambda Z A)))$ may be written $(\lambda X \lambda Y \lambda Z A)$, and so on. A further abbreviation is then permitted, namely the omission of the repeated occurrences of λ in the resulting inscription. Thus $(\lambda X \lambda Y A)$ may be written $(\lambda X Y A)$; $(\lambda X \lambda Y \lambda Z A)$ may be written $(\lambda X Y Z A)$, and so on.

In connection with these conventions about the omission and restoration of parentheses, it may be remarked that the absence from our system of functions taking more than one argument causes no inconvenience, and brings much benefit in the form of greater simplicity. However, construing, for example, $(or AB)$ as the application of $(or A)$ to B , and hence construing *or* as a function from truth values to functions-from-truth-values-to-truth-values, instead of construing it as a function from pairs of truth values to truth values, does seem a little strange when it is first encountered. Fortunately, as the lambda calculus is becoming more widely familiar, this particular feature is likely to be strange to relatively few readers.

We may write $(choice P)$ instead of $(choice \alpha P)$, and $(equal AB)$ instead of $(equal \alpha AB)$, since the types of P, A, B , and the assumption that the expressions are well-formed, determine α uniquely.

The reader may at this point be wondering how quantification is managed in this system. The following explanation is intuitive and informal.

Expressions of type *truthvalue* are called *sentences*, and for each type symbol α the expressions of type $(\alpha \rightarrow \text{truthvalue})$ are called *predicates* (of objects of type α). One would expect to find in a system of logic the ability to write, e.g., $(for all XA)$, $(for some XA)$, where A is a sentence and X is a

nonspecial identifier (or 'variable'), with the meanings that A is true for all, or for at least one, respectively, of the objects which can be denoted by X . In fact in our system these two assertions are exactly expressed by the two expressions

$$((\lambda X A) (\text{choice } (\lambda X (\text{not } A))))$$

$$((\lambda X A) (\text{choice } (\lambda X A)))$$

respectively. The intuitive idea is this. The expression (*for some* $X A$) says that $(\lambda X A)$ is true of at least one object. However, (*choice* $(\lambda X A)$) is certainly such an object if there are any objects at all of which $(\lambda X A)$ is true, while if there are no objects of which $(\lambda X A)$ is true, then (*choice* $(\lambda X A)$) is an object of which $(\lambda X A)$ is false. Hence (*for some* $X A$) and $((\lambda X A) (\text{choice } (\lambda X A)))$ are either both true, or else both false. And so they are equivalent.

A similar analysis shows the intuitive equivalence of (*for all* $X A$) with $((\lambda X A) (\text{choice } (\lambda X (\text{not } A))))$.

In actual practice we write (*for some* $X A$), (*for all* $X A$) respectively as abbreviations of $((\lambda X A) (\text{choice } (\lambda X A)))$, $((\lambda X A) (\text{choice } (\lambda X (\text{not } A))))$. In the machine implementation to be described later, this abbreviation is used both for input and output of expressions, while the unabbreviated expressions are used internally. This is of course purely a matter of 'syntactic sugar'.

As a further abbreviation we write, for example,

$$(\text{for all } X Y Z A)$$

instead of

$$(\text{for all } X (\text{for all } Y (\text{for all } Z A)))$$

and in general

$$(\text{for all } X_1 \dots X_n A)$$

in place of

$$(\text{for all } X_1 (\dots (\text{for all } X_n A) \dots)).$$

And similarly for nested existential quantifications.

The usual notion of *free* and *bound* occurrences of identifiers in expressions prevails in the present system. Namely, for each identifier X and each expression A we classify each occurrence of X in A either as a *free* occurrence of X in A or as a *bound* occurrence of X in A , as follows:

if A is X , then the occurrence of X in A is free;

if A is an application (FB) then each occurrence of X in A is either an occurrence of X in F or an occurrence of X in B , and it is a free, or a bound, occurrence of X in A according as it is a free, or a bound, occurrence, respectively, in F or in B ;

if A is an abstraction $(\lambda Y B)$ then if X is Y , every occurrence of X in A is a bound occurrence of X in A ;

THEOREM PROVING

while if X is not Y then each occurrence of X in A is also an occurrence of X in B , and is a free, or a bound, occurrence of X in A according as it is a free, or a bound, occurrence of X in B .

The notion of free and bound occurrences within an expression A carries over from occurrences of identifiers in A to occurrences in A of expressions of any kind. An occurrence of the expression B within the expression A is a bound occurrence of B in A if there is an identifier which has a bound occurrence in A that is a free occurrence in B . An occurrence of B in A which is not a bound occurrence of B in A is a free occurrence of B in A .

For example: $(\lambda X(A X))$ occurs free in $(\lambda X(A X))$, but $(A X)$ occurs bound in $(\lambda X(A X))$.

If S is any set of expressions we define S^* to be the set of all expressions which have *free* occurrences in expressions in S . Obviously S^* is finite whenever S is finite, and can easily be computed from S .

For example, if S is the set containing only $(\lambda X(A X))$ then S^* contains the expressions: $(\lambda X(A X))$, A (assuming that A is an expression in which there are no free occurrences of X).

We say that two expressions A, B are *variants* of each other, and we write: $A \sim B$, if A and B are 'identical up to within changes of bound identifiers'. More precisely, $A \sim B$ if and only if either A and B are both identifiers, and $A = B$, or A and B are both applications (FC) and (GD) such that $F \sim G$ and $G \sim D$, or A and B are both abstractions, $(\lambda X C)$ and $(\lambda Y D)$ respectively, such that X and Y are of the same type, and $C' \sim D'$; here C', D' come respectively from C, D , when each free occurrence of X in C , and of Y in D , is replaced by an occurrence of Z , Z being an identifier of the same type as X and Y which does not occur in either C or D .

From the above inductive definition of $A \sim B$ it is straightforward to write a machine program to determine, given two expressions A, B , whether or not $A \sim B$.

Intuitively, two distinct expressions which are variants of each other are not 'really' distinct at all. They are, so to speak, the same expression in two different representations, and can be interchanged with one another within any expression without changing the meaning of the expression in the slightest.

A more general equivalence relation between expressions arises when we consider any partition K whose blocks are sets of expressions of the same type (although different blocks may have expressions of different types). For any such partition K and any two expressions A and B we say that A and B are *equivalent modulo* K , and write

$$A \equiv B \pmod{K},$$

when A and B are exactly like each other except for the fact that possibly, in one or more places, A may have a free occurrence of one expression and B may have a free occurrence of a different expression, provided that these expressions lie in the same block of K .

For example, $(FABC)$ is equivalent to $((\lambda XG)A(GD)C)$ modulo $[[F(\lambda XG)][B(GD)]]$, but $(\lambda F(F(GF)))$ is not equivalent to (λFA) modulo $[[A(F(GF))]]$.

(Here we have introduced a convenient notation for explicitly displaying partitions of expressions. We write such a partition as $[B_1 \dots B_n]$, each block B_j being written $[A_1 \dots A_m]$, where A_1, \dots, A_m are the expressions in the block B_j . Of course the order of the blocks and the order of the expressions within the blocks is immaterial.)

This notion of equivalence of expressions is a very natural and useful one. We use it in contexts where the partition K has the intuitive interpretation that expressions within any one block of K all denote the same object, so that if $A \equiv B \pmod{K}$ then, intuitively, A and B also denote the same object.

It is straightforward to write a machine program to determine rapidly, given A, B and K , whether or not $A \equiv B \pmod{K}$.

Finally, we define the operation of *substitution*. Let A and B be expressions and let X be an identifier of the same type as B . Then we write $A\{B/X\}$ to represent the expression which results from the replacement of *every* free occurrence of X in A by an occurrence of B , where A' is the earliest variant of A (in some canonical ordering of the expressions) which contains no bound occurrences of any identifier occurring free in B .

We write $A\{B_1/X_1, \dots, B_n/X_n\}$ to represent the *simultaneous replacement*, throughout A , of each *free* occurrence of each of the identifiers X_1, \dots, X_n in A by an occurrence of the corresponding one of the expressions B_1, \dots, B_n ; where A' is the earliest variant of A which contains no bound occurrences of any identifier occurring free in any of B_1, \dots, B_n .

SEMANTICS

Many of the informal remarks made during the explanation of the syntax of the system were semantical in character, intended to provide intuitive motivations for syntactical features. It is hoped that thereby the reader has already acquired some feel for what is intended in the semantical part of the system. The following definition constitutes the 'official' semantic machinery.

Interpretations. An interpretation is a pair (g, H) satisfying the conditions:

- (1) H is a set of objects to each of which is assigned a type symbol from the set T (by an extension to $E \cup H$ of the mapping L already defined for E) in such a way that every object in $H(\alpha \rightarrow \beta)$ is a function from $H\alpha$ to $H\beta$, for all type symbols α and β , and the only objects of type *truthvalue* are the two truth values, *true*, *false*;
- (2) g is a mapping of the nonspecial identifiers onto objects of H such that for all X the object gX is of the same type as X (such mappings are called *assignments in H*);
- (3) every assignment m in H is extendable to a *denotation map* m^* in H ; that is, for every expression A there is an object m^*A in H which satisfies the appropriate one of the following conditions:

THEOREM PROVING

- (4) if A is a nonspecial identifier, then: $m^*A = mA$;
 (5) if A is a special identifier, then m^*A is the object described on the right-hand side of the appropriate equation among the following:

$m^*true = true$

$m^*false = false$

$m^*not = negation$

$m^*or = disjunction$

$m^*and = conjunction$

$m^*implies = implication$

$m^*equal \alpha = equality \text{ between objects of type } \alpha$

$m^*choice \alpha = a \text{ choice function for objects of type } \alpha$

(Here it is understood that *negation* is the function n such that $(n \text{ true}) = false$, $(n \text{ false}) = true$: that *disjunction* is the function d such that $((d \text{ true}) \text{ true}) = ((d \text{ true}) \text{ false}) = ((d \text{ false}) \text{ true}) = true$ and $((d \text{ false}) \text{ false}) = false$: that *conjunction* is the function c such that $((c \text{ true}) \text{ true}) = true$ and $((c \text{ true}) \text{ false}) = ((c \text{ false}) \text{ true}) = ((c \text{ false}) \text{ false}) = false$: that *implication* is the function i such that $((i \text{ true}) \text{ false}) = false$ and $((i \text{ true}) \text{ true}) = ((i \text{ false}) \text{ true}) = ((i \text{ false}) \text{ false}) = true$; that *equality between objects of type α* is the function e such that $((ea)b) = true$ when a is b , $((ea)b) = false$ when a is not b , for all objects a, b of type α : and that *a choice function for objects of type α* is a function c such that for all objects s of type $(\alpha \rightarrow \text{truthvalue})$, either $(s(cs)) = true$ or $(sb) = false$ for all objects b of type α .)

(6) if A is an application (FB) then m^*A is the object yielded when m^*F is applied to m^*B ;

(7) if A is an abstraction (λXB) then m^*A is the function which yields n^*B when applied to nX , where n is any assignment in H which coincides with m at all nonspecial identifiers, except possibly at X .

Intuitively, the conditions (5), (6) and (7) impose on the set H of objects the constraint that whenever an object b can be defined in terms of (or constructed out of) objects which are in H , then b must also be in H . Defining an object 'in terms of objects in H ' is thus construed as choosing an assignment g in H and an expression D , and then declaring that the object is g^*D . The expression D is then the *definition* of the object g^*D . We are simply requiring that our universes of discourse must be closed under the operation of defining objects in terms of other objects.

Whenever (g, H) is an interpretation and A is an expression we say that A *denotes* the object g^*A in the interpretation (g, H) . If A is a sentence and g^*A is *true*, or *false*, respectively, then we say that A is true, or A is false, respectively, in the interpretation (g, H) , or that the interpretation *satisfies*, or *falsifies*, A .

THE PROOF PROCEDURE

Our proof procedure is based on the notion of a semantic partition of a set of expressions. If S is a set of expressions and K is a partition of S we say that K is a *semantic partition* of S if and only if there is an interpretation (g, H) such that, for all expressions A, B in S :

$[A]K = [B]K$ if and only if $g^*A = g^*B$.

Here we are using the notation $[A]K$ to represent the block of K in which A lies. It is straightforward to verify that the following *semantic partition conditions* are *necessary* (but not *sufficient*) for K to be a semantic partition of S . Each condition is assumed to be preceded by the hypothesis that each of the expressions exhibited in the statement of the condition is in S .

- (1) K is a refinement of the partition of S into types.
- (2) $[true]K \neq [false]K$.
- (3) $[A]K = [true]K$, or $[A]K = [false]K$, for all sentences A in S .
- (4) $[A]K \neq [(not A)]K$.
- (5) $[(or AB)]K = [false]K$ if and only if $[A]K = [B]K = [false]K$.
- (6) $[(and AB)]K = [true]K$ if and only if $[A]K = [B]K = [true]K$.
- (7) $[(implies AB)]K = [false]K$ if and only if $[A]K = [true]K$ and $[B]K = [false]K$.
- (8) $[(equal AB)]K = [true]K$ if and only if $[A]K = [B]K$.
- (9) if $A \equiv B \pmod{K}$ then $[A]K = [B]K$.
- (10) if $A \approx B$ then $[A]K = [B]K$.
- (11) if $[(\lambda X A)(choice(\lambda X A)))]K = [false]K$ then $[(\lambda X A)B)]K = [false]K$.
- (12) if $[(\lambda X A)(choice(\lambda X(not A)))]K = [true]K$ then $[(\lambda X A)B)]K = [true]K$.
- (13) if $[(for some X_1 \dots X_n A)]K = [false]K$ then $[A\{B_1/X_1, \dots, B_n/X_n\}]K = [false]K$.
- (14) if $[(for all X_1 \dots X_n A)]K = [true]K$ then $[A\{B_1/X_1, \dots, B_n/X_n\}]K = [true]K$.

In condition (10), $A \approx B$ means: if C comes from computing out A , and D comes from computing out B , then $C \sim D$. *Computing out* an expression is the process of persistently replacing subexpressions of the form $(\lambda X_1 \dots X_k M)N_1 \dots N_k$ by $M\{N_1/X_1, \dots, N_k/X_k\}$, and subexpressions of the form $(\lambda X_1 \dots X_k (FX_1 \dots X_k))$, in which none of X_1, \dots, X_k occur free in F , by F , until no such subexpressions remain. It can be shown that every expression can be computed out, and that different ways of computing it out will produce expressions which are at most variants of each other.

Mnemonically suitable names for these conditions are as follows:

- (1) *stratification*
- (2) *noncontradiction*

THEOREM PROVING

- (3) *tertium exclusion*
- (4) *negation*
- (5) *disjunction*
- (6) *conjunction*
- (7) *implication*
- (8) *equation*
- (9) *conflation*
- (10) *variation*
- (11) *existential generalization*
- (12) *universal instantiation*
- (13) *strong existential generalization*
- (14) *strong universal instantiation*

Now for any given finite set S of expressions one can compute $q(S)$, the set of all partitions of S which satisfy the semantic partition conditions.

Furthermore, for any given partition D of a subset of S , we can check each partition in $q(S)$ to see whether or not D is a *subpartition* of it, i.e., whether or not each block of D is wholly included in some block of it; and we can therefore compute the set $r(S, D)$ of partitions of S which satisfy the semantic partition conditions and of which D is a subpartition.

Our proof procedure exploits the fact that for any finite set S of expressions and any partition D of a subset of S we can compute the set $r(S, D)$.

The procedure is as follows, where W_0, W_1, \dots , is some enumeration of the set E of all expressions. The input to the procedure is a pair (S, D) in which S is a finite set of expressions and D is a partition of S . The procedure then consists of computing, for $j=0, 1, \dots$, the sets $r(S_j^*, D)$, where $S_0=S$, and for all $j \geq 0$, $S_{j+1} = S_j \cup \{W_j\}$. The reader will recall the definition of S^* as the set of all expressions which have free occurrences in expressions in S . The procedure terminates after the computation of $r(S_j^*, D)$ if and only if $r(S_j^*, D)$ is empty.

The procedure just described is completely determined by the enumeration W_0, W_1, \dots , of expressions. However, it is not necessary, for the proof of the fundamental theorem below, that this enumeration be uniformly the same for each pair (S, D) to which the procedure is applied. For each (S, D) an enumeration may be chosen which depends on the particular details of S and D , or upon any other circumstances one chooses. The theorem is:

Henkin's Theorem. D is a semantic partition of S if and only if the procedure does not terminate when it is applied to (S, D) .

Consider now, in view of Henkin's Theorem, how the procedure can be used directly as a proof procedure. Suppose one wishes to show that a sentence B follows from the sentences A_1, \dots, A_n . Then one wishes to show, in fact,

that there is no interpretation which simultaneously satisfies each of A_1, \dots, A_n but falsifies B . But this is the same as showing that the partition

$$D = [[A_1, \dots, A_n, \text{true}][B, \text{false}]]$$

is not a semantic partition.

But if D is not in fact a semantic partition of

$$S = \{A_1, \dots, A_n, B, \text{true}, \text{false}\}$$

then, by Henkin's Theorem, if the procedure is applied to (S, D) it will terminate eventually.

Moreover, if the procedure eventually terminates when it is applied to (S, D) then, by Henkin's Theorem, D is not a semantic partition of S .

Therefore B follows from A_1, \dots, A_n if and only if the procedure eventually terminates when applied to (S, D) , where S and D are as specified above.

Here follows a brief outline of the proof of Henkin's Theorem for the benefit of readers who may wish to study Henkin's classic paper (1950). This outline may be skipped without any loss of continuity in the present exposition.

To show that if D is a semantic partition of S then the procedure does not terminate when applied to (S, D) , one shows that none of the sets $r(S_j^*, D)$ is empty because each, in fact, contains a semantic partition. D is in the set $r(S_0^*, D)$ and is by hypothesis a semantic partition. So we show that, for all $j \geq 0$, if $r(S_j^*, D)$ contains a semantic partition then so does $r(S_{j+1}^*, D)$. Assume then that $r(S_j^*, D)$ contains the semantic partition P . By definition of r , D is a subpartition of P (written: $D \leq P$). Let J be an interpretation which induces P in S_j^* , and let Q be the partition induced by J in S_{j+1}^* . Then Q is a semantic partition which is in $r(S_{j+1}^*, D)$, because it satisfies the semantic partition conditions and has D as a subpartition.

To show that if the procedure does not terminate when applied to (S, D) then D is a semantic partition of S , one constructs an interpretation which induces in the set E of all expressions a partition P such that $D \leq P$, and which therefore induces D in S . This construction goes as follows.

If the procedure does not terminate when applied to (S, D) , then none of the sets $r(S_j^*, D)$ is empty. Since each partition in $r(S_{j+1}^*, D)$ has exactly one partition in $r(S_j^*, D)$ as a subpartition, the sets $r(S_0^*, D)$, $r(S_1^*, D)$, \dots , are the successive layers of nodes in a tree which is infinite, but finitely branching. Hence by König's Lemma there is an infinite sequence P_1, P_2, \dots , of partitions such that, for each j , P_j is in $r(S_j^*, D)$, and such that $D \leq P_1 \leq P_2 \leq \dots \leq P$, where P is that partition of the set E of all expressions such that, for any expressions A, B : $[A]P = [B]P$ if and only if for some $j \geq 1$, $[A]P_j = [B]P_j$.

One then shows that P is a semantic partition by constructing an interpretation (g, H) which induces P in E . H is defined simultaneously with the denotation map g^* by induction over the set T of type symbols.

THEOREM PROVING

For each basic type symbol α , the objects of type α are defined to be the blocks B of P such that $B \leq E\alpha$. This determines the individuals in H . The truth values *true*, *false* are identified with the blocks $[true]P$, $[false]P$ respectively. Then for each expression A of type α , g^*A is defined to be $[A]P$.

For each higher type symbol $(\alpha \rightarrow \beta)$ the definition of the objects of type $(\alpha \rightarrow \beta)$, and of g^* for all expressions of type $(\alpha \rightarrow \beta)$, is made on the inductive assumption that the objects of types α and β have already been defined, and that g^* has already been defined for all expressions of types α and β .

With this assumption, for each expression F of type $(\alpha \rightarrow \beta)$ g^*F is defined to be the function which yields $g^*(FA)$ when applied to g^*A , for all expressions A of type α . The function g^* having been thus defined for all expressions of type $(\alpha \rightarrow \beta)$, the objects of type $(\alpha \rightarrow \beta)$ are then defined to be just the objects g^*F , as F runs through the expressions of type $(\alpha \rightarrow \beta)$.

It can then be shown that (g, H) is an interpretation which induces P in E , where g is the restriction of g^* to the nonspecial identifiers.

IMPLEMENTING THE PROCEDURE AS A MAN-MACHINE INTERACTIVE PROCESS

We envisage the following implementation of the procedure just described.

There is a keyboard at which the user can type expressions into the machine, indicating as he does so, by pressing suitable control keys, whether the expression being typed is the next in the sequence W_0, W_1, \dots , or whether it is to be entered into the partition D , and if so, into which block of D it is to be placed.

When the process is *initialized* by the depression of the appropriate key, there are created in the store of the machine the following objects:

- the set $S = \{true, false\}$,
- the partition $D = [[true][false]]$,
- the set $r(S^*, D) = \{[[true][false]]\}$,
- the partition $M^* = [[true][false]]$,
- the partition $M = [[true][false]]$.

Thereafter the computation proceeds by successive stages, each of which consists of the entry of an expression from the keyboard followed by the immediate updating of the objects S , D , $r(S^*, D)$, M^* and M by the machine. The partition M^* is a partition of S^* . The partition M is a partition of S . M^* is the *meet* of the partitions in the set of partitions $r(S^*, D)$. That is to say, M^* is the unique partition which is a subpartition of every partition in $r(S^*, D)$ and which has the property that if K is any partition which is a subpartition of every partition in $r(S^*, D)$ then $K \leq M^*$. M is simply the restriction of M^* to S . That is to say, M is that partition of S such that $[A]M = [B]M$ if and only if $[A]M^* = [B]M^*$, for all A and B in S . The computation of the objects $r(S^*, D)$, M^* and M from S and D is quite straightforward.

The entries from the keyboard are either W -entries or D -entries. The user indicates which by pressing a suitable key. He then types in an expression A (if he is making a W -entry) or an expression A followed by an expression B (if he is making a D -entry).

When a W -entry has been made, the machine forms a new set S by adding the expression A to the old set S . The new D is the same as the old D . The machine then recomputes $r(S^*, D)$, M^* and M . If $r(S^*, D)$ is empty, the termination of the computation is signalled to the user. Otherwise the user is given the signal to begin the next stage by making another entry.

When a D -entry has been made, the machine forms a new set S by adding the expression A to the old set S , just as in the case of a W -entry. However, it now also forms a new partition D , in the following way: If the expression B is in one of the blocks of D , then the expression A is added to that block to form the new partition D . If the expression B is not in any of the blocks of D , then the block $[A]$ is added to D as a new block, to form the new partition D . Once the new S and the new D have been formed in this way, the machine then recomputes $r(S^*, D)$, M^* and M , and completes the stage in just the same way as was described above for the case of a W -entry.

It is of course an error for the user to make a D -entry in which the expressions A and B are both already in blocks of D , and $[A]D \neq [B]D$. It is otiose, but not an error, if $[A]D = [B]D$. In either case the machine will detect the anomaly and will output an appropriate message.

In the event that the computation stops as described, the user may correctly conclude that the (current) partition D is not a semantic partition of that subset of the (current) set S whose members are in its blocks. The completeness of the procedure when implemented in this way depends, of course, entirely on the sequence of expressions which are entered from the keyboard into the set S . If this sequence is such as to exhaust E eventually (in the sense that, for every expression A in E , there eventually comes a time when A occurs in the sequence) then completeness is achieved.

The significance of the partition M is that it shows 'what immediately follows from D ' when D is construed as a body of assertions. The general form of these assertions is that certain expressions denote the same object as certain other expressions (namely, every expression in each block B of D denotes the same object as every other expression in B); and the information conveyed by M is construed similarly as a body of assertions about expressions in S . When the block B is $[true]M$ or $[false]M$ the fact that an expression is in B means that it can be inferred to be true, or can be inferred to be false, from the information in D .

As was mentioned earlier, we envisage the user typing in expressions using the abbreviations made possible by the conventions concerning omission and restoration of parentheses and by the notation for existential and universal quantification which was explained earlier. Conversely, when the machine displays the partition M to the user, or otherwise outputs expressions to him.

THEOREM PROVING

we envisage the performing of an 'ensugaring' process whereby the expressions are abbreviated and otherwise made as readable as possible.

Naturally, the user is not prohibited from entering unabbreviated expressions should he wish to do so.

The following example illustrates how the process works. It is to be shown that the sentence

(for all P (implies $(KP)(PM)$))

follows from the two sentences

(for all R (implies $(RQ)(RK)$))

(for all S (implies $(QS)(SM)$)).

We first *initialize* the process. The following is then the record of the computation as it actually appears on the keyboard transcript, part of which is typed by the machine and part by the user, as will be explained in a moment:

- | | |
|---|----------|
| 1. (for all S (implies $(QS)(SM)$)) | : true |
| 2. (for all R (implies $(RQ)(RK)$)) | : true |
| 3. (for all P (implies $(KP)(PM)$)) | : false |
| 4. (implies ((λQ (for all S (implies $(QS)(SM)$))) Q
((λQ (for all S (implies $(QS)(SM)$))) K)) | : [true] |
| 5. ((λQ (for all S (implies $(QS)(SM)$))) Q) | : [true] |
| 6. ((λQ (for all S (implies $(QS)(SM)$))) K) | : [true] |
| 7. (for all S (implies $(KS)(SM)$)) | : STOP. |

The numerals in the left-hand column are typed by the machine; that is, as a signal to the user to begin the j th stage, the machine returns the carriage to begin a new line and types: j . It also types the colon after the first expression on each line, and in the case of the line being a W -entry (as lines 4 through 7 are in the example) types also the material to the right of the colon, the significance of which will shortly be explained. When the line is a D -entry (as are the first three lines above) it is the user who types the material following the colon. This consists of the expression B called for in a D -entry. The expression on each line following the numeral and preceding the colon is the expression A called for in the entry.

The explanation of the material typed by the machine to the right of the colon in the case of a W -entry is as follows. As soon as the machine has computed the new partition M it determines in which block of M the new expression A lies. If A lies in either $[true]M$ or $[false]M$, then the machine types $[true]$ or $[false]$ after the colon, respectively. This immediately tells the user that the sentence A which he has just typed in can be inferred to be true, or to be false, respectively, from the information in D . If A lies in some other block of M the machine types $[k_1, \dots, k_n]$, where k_1, \dots, k_n are the numbers of the lines at which the expressions in that block were entered into the machine. If $n=1$ then the only number thus typed by the machine is the number of the

current line itself, signifying to the user that the new expression A lies in a block all to itself in the newly computed M .

Note that the D -entries can be distinguished from the W -entries in the transcript by the fact that the material after the colon on a D -entry line is always an expression.

When the machine finds that the computation must terminate it types STOP immediately after the colon.

To understand this particular example it helps to realise that lines 4 through 7 follow from earlier lines by familiar principles of inference. Line 4 follows from line 2 by virtue of being an instance of it. Line 5 follows from line 1 by lambda-conversion. Line 6 follows from lines 4 and 5 by *modus ponens*. Line 7 is a variant of line 3, and hence can be inferred to be false, but also follows from line 6 by lambda-conversion, and hence can be inferred to be true. Therefore contradiction has arisen, and the computation accordingly terminates.

This example illustrates the way in which the record of a completed computation can be read as a *proof* of the semantic impossibility of the partition D . Even though the computation does not terminate in the normal way the record of the incomplete computation can still be read as a series of *inferences* made on the basis of D as premise. If D is in fact a semantic partition (i.e., a *consistent* collection of information) then of course the computation will not terminate in the normal way.

The intuitive feel of this process is that of a *question-and-answer system*. The user inserts information into the system by means of D -entries, and asks questions of the system by means of W -entries. In entering the expression A by a W -entry, the user is in effect asking 'what can be immediately inferred about this expression?'. His answer is contained most directly in the material typed by the machine immediately following his typing of A . A more detailed answer is found in the full partition M , and a still more detailed answer in the partition M^* .

One need not necessarily follow the *reductio ad absurdum* pattern of argument with this system in order to make a proof. If, in our example, line 3 had been omitted, and an eighth line added with the desired conclusion as the A of a W -entry, the machine would have typed [*true*] immediately. The resulting transcript would then have read as a deduction of the desired conclusion from the two given premises.

The following example shows rather more of the machinery in operation. The theorem to be proved is the proposition that, for all non-negative integers n ,

$$2 \sum_{k=0}^{k=n} k = n(n+1).$$

The proof is an elementary application of the principle of mathematical induction. We give the essential part of the proof, interpolating comments between entries:

THEOREM PROVING

1. $(\text{for all } n((\lambda n(\text{equal}(\text{times two}(\text{summation identity } n))(\text{times } n(\text{plus } n \text{ one}))))n)) : \text{false}$

The first entry is a *D*-entry stating the theorem and declaring it to be false in preparation for a *reductio ad absurdum* proof. Now we set out to prove the basis of the induction, which will be reached at line 16.

2. $(\text{times two}(\text{summation identity zero})) : [2]$
3. $(\text{for all } f(\text{equal}(\text{summation } f \text{ zero})(f \text{ zero}))) : \text{true}$
This is part of the definition of the summation operator.
4. $(\text{equal}(\text{summation identity zero})(\text{identity zero})) : [\text{true}]$
5. $(\text{times two}(\text{identity zero})) : [2,5]$
6. $(\text{equal identity}(\lambda xx)) : \text{true}$
This is simply a definition of the identity function.
7. $(\text{times two}((\lambda xx)\text{zero})) : [2,5,7]$
8. $(\text{times two zero}) : [2,5,7,8]$
9. $(\text{for all } n(\text{equal zero}(\text{times } n \text{ zero}))) : \text{true}$
This is part of the definition of multiplication.
10. $(\text{equal zero}(\text{times two zero})) : [\text{true}]$
11. $\text{zero} : [2,5,7,8,11]$
12. $(\text{times zero}(\text{plus zero one})) : [12]$
13. $(\text{for all } n(\text{equal zero}(\text{times zero } n))) : \text{true}$
14. $(\text{equal zero}(\text{times zero}(\text{plus zero one}))) : [\text{true}]$
15. $(\text{equal}(\text{times two}(\text{summation identity zero}))(\text{times zero}(\text{plus zero one}))) : [\text{true}]$
16. $((\lambda n(\text{equal}(\text{times two}(\text{summation identity } n))(\text{times } n(\text{plus } n \text{ one}))))\text{zero}) : [\text{true}]$

Here we have deduced the sentence which serves as the basis of the induction. Now we can appeal to the general principle of induction, stated as follows:

17. $(\text{for all } P(\text{implies}(\text{and}(P \text{ zero})(\text{for all } n(\text{implies}(Pn)(P(\text{plus } n \text{ one})))))(\text{for all } n(Pn)))) : \text{true}$
Then we introduce the appropriate instance of this principle:
18. $(\text{implies}(\text{and}(Q \text{ zero})(\text{for all } n(\text{implies}(Qn)(Q(\text{plus } n \text{ one})))))(\text{for all } n(Qn))) : [\text{true}]$

Here we have used the identifier *Q* to shorten the expression. We must therefore now say that *Q* is supposed to mean:

19. $(\text{equal } Q(\lambda n(\text{equal}(\text{times two}(\text{summation identity } n))(\text{times } n(\text{plus } n \text{ one})))) : \text{true}$

In terms of this abbreviation we then have, from lines 1 and 16 respectively:

20. $(\text{for all } n(Qn)) : [\text{false}]$

21. ($Q\ zero$) : [true]
 Now lines 18, 20 and 21 allow us to infer:
22. ($for\ all\ n(implies(Qn)(Q(plus\ n\ one)))$) : [false]
 If we now introduce the abbreviation:
23. ($equal\ b(choice(\lambda n(not(implies(Qn)(Q(plus\ n\ one))))))$) : true
 we can immediately infer from line 22 the following:
24. ($implies(Qb)(Q(plus\ b\ one))$) : [false]
 and therefore:
25. (Qb) : [true]
 26. ($Q(plus\ b\ one)$) : [false]
 Whence, putting in the definition of Q :
27. ($(\lambda n(equal(times\ two(summation\ identity\ n))(times\ n(plus\ n\ one))))b$):
 [true]
 28. ($(\lambda n(equal(times\ two(summation\ identity\ n))(times\ n(plus\ n\ one)))(plus\ b\ one)$) : [false]
 and therefore, by lambda-conversion:
29. ($equal(times\ two(summation\ identity\ b))(times\ b(plus\ b\ one))$) :
 [true]
 30. ($equal(times\ two(summation\ identity(plus\ b\ one))(times(plus\ b\ one)(plus(plus\ b\ one)\ one)))$) : [false]

The remainder of the proof consists of deducing the left- and right-hand sides of the equation in line 30 to be equal, using the distributivity of multiplication through addition, the associativity of addition, the definitions of *one*, *two* and *summation*, together with the equation in line 29, which is the premise for the induction step.

The introduction of the identifiers b and Q is entirely a matter of convenience. This example shows how the *choice* expressions automatically play the role of 'Skolem functions' in the present system. The intuitive interpretation of the *choice* expression in line 22 (unabbreviated) and line 23 is that its successor is the least number n for which (Qn) is false, as indeed lines 25 and 26 actually state explicitly.

Note that the D -entries (lines 1,3,6,9,13,17,19,23) are scattered through the proof, wherever it is convenient to put them. They do not have to be at the beginning. One can insert extra premises at any time, and in particular one can at any time introduce abbreviations for expressions (as we did in lines 19 and 23 of the example) so as to avoid the need to form unduly long expressions.

FURTHER DEVELOPMENTS OF THE SYSTEM

There are two main directions in which improvements can be made on the present system. As it stands, it is simply (1) a pedagogical device intended to

THEOREM PROVING

demonstrate the fundamental principles, and (2) an initial working system designed to serve as the nucleus of later, stronger systems.

The first kind of improvement is the formulation of more and more stringent sets of semantic conditions. The objective is, so to speak, to make the sets $r(S, D)$ as small as possible. The examples have illustrated how, in effect, the semantic partition conditions operate as a set of inference principles, providing the system with a certain immediate inference capability. The present system has only a very limited immediate inference capability, and systems having much more powerful ones are quite straightforward to construct.

The second kind of improvement is to introduce mechanisms whereby *the machine*, as well as the user, makes entries into the set S and the partition D . One such mechanism is the practically useless, but theoretically interesting, possibility of having the machine make all the entries except the D -entries, by simply generating and entering all expressions, in some particular order. The system would then be a complete, autonomous proof procedure. A less useless mechanism of this sort would be to have the machine construct new expressions systematically from the expressions in the set S^* , forming instances of universally quantified sentences by substituting expressions in S^* for bound identifiers; forming new abstractions from expressions in S^* by binding each free identifier in each expression; applying expressions to other expressions if these applications are not already present; and so on. A further such mechanism would be to have the machine introduce at appropriate times abbreviations for expressions which are too long. It would do this by equating a new identifier to the expression to be abbreviated, *via* a D -entry. There is a great variety of such mechanisms, and it is anticipated that most of the research into the mechanization of higher-order logic will be the investigation and development of suitable ones.

The two fundamental processes of our system are simply (1) the growing of the set S of expressions and (2) the simultaneous formation of the 'immediate inference partition' M of S . What we are saying, therefore, is that improvements in the system will be mainly directed at the way in which (1) and (2) are effected.

It is obviously desirable to make M as strong a partition of S as possible. For any S and D , let $\bar{r}(S, D)$ be the set of all semantic partitions of S , of which D is a subpartition; and let \bar{M} be the meet of the partitions in the set $\bar{r}(S, D)$. Then we shall always have $\bar{r}(S, D) \subseteq r(S, D)$, and therefore $M \leq \bar{M}$, for all S and D . Obviously we cannot have $M = \bar{M}$ for all S and D , for then we would have a decision procedure for higher-order logic. However, \bar{M} is a useful theoretical upper bound on M , to which we want M to approximate as closely as possible consistent with a reasonable computation cost.

Not falling under either of these broad categories of improvements, but extremely important from a practical point of view, are the possible improvements of the 'syntactic sugar' kind. The goal should be to have as *natural* as

possible a format for the input and output versions of expressions, and as *efficient* as possible a format for the internal versions of expressions.

For example, wherever it is natural to have an infix notation, as in A or B , A implies B , $A=B$, $A+B$, the user should be able to write it thus if he wishes. Far more drastic departures from the simple 'official' applicative and abstractive appearance of expressions are possible than those explicitly mentioned in the present account. In general, any natural mathematical notation is usually quite easily construed as 'syntactic sugar' coating of expressions of the present system. In this connection the reader would be well advised to read the beautiful studies of Landin (e.g., 1964), revealing the deep and universal relationship between mathematical notations on the one hand and the lambda-calculus on the other.

In our examples and the general description of the system to be implemented we have left in the background the question of how it deals with the types of identifiers. Within the general policy of making things feel natural for the user, the policy concerning types ought to be that they should not obtrude on the user's natural modes of writing and reading expressions. Evidently, the type of each identifier, and therefore of each expression, must be known to the user: this is part of what it means to say that he understands them. The machine must also have this information. Hence there must be mechanisms for specifying the types of identifiers when they are newly introduced. We envisage that, roughly speaking, the machine will seek to determine the types of all new identifiers by analyzing the context in which they are introduced, which very frequently is such that the type of each new identifier can be uniquely determined from the types of old identifiers, the types of special identifiers, and the assumption that expressions are well formed unless they can be proved not to be. If the machine's attempt to determine the type of a new identifier fails, then, and only then, the user will be specifically requested to state what its type is. The user may also of course *volunteer* redundant type information if he so chooses (for example, in order to have it recorded explicitly on the transcript of the computation).

Concluding remarks

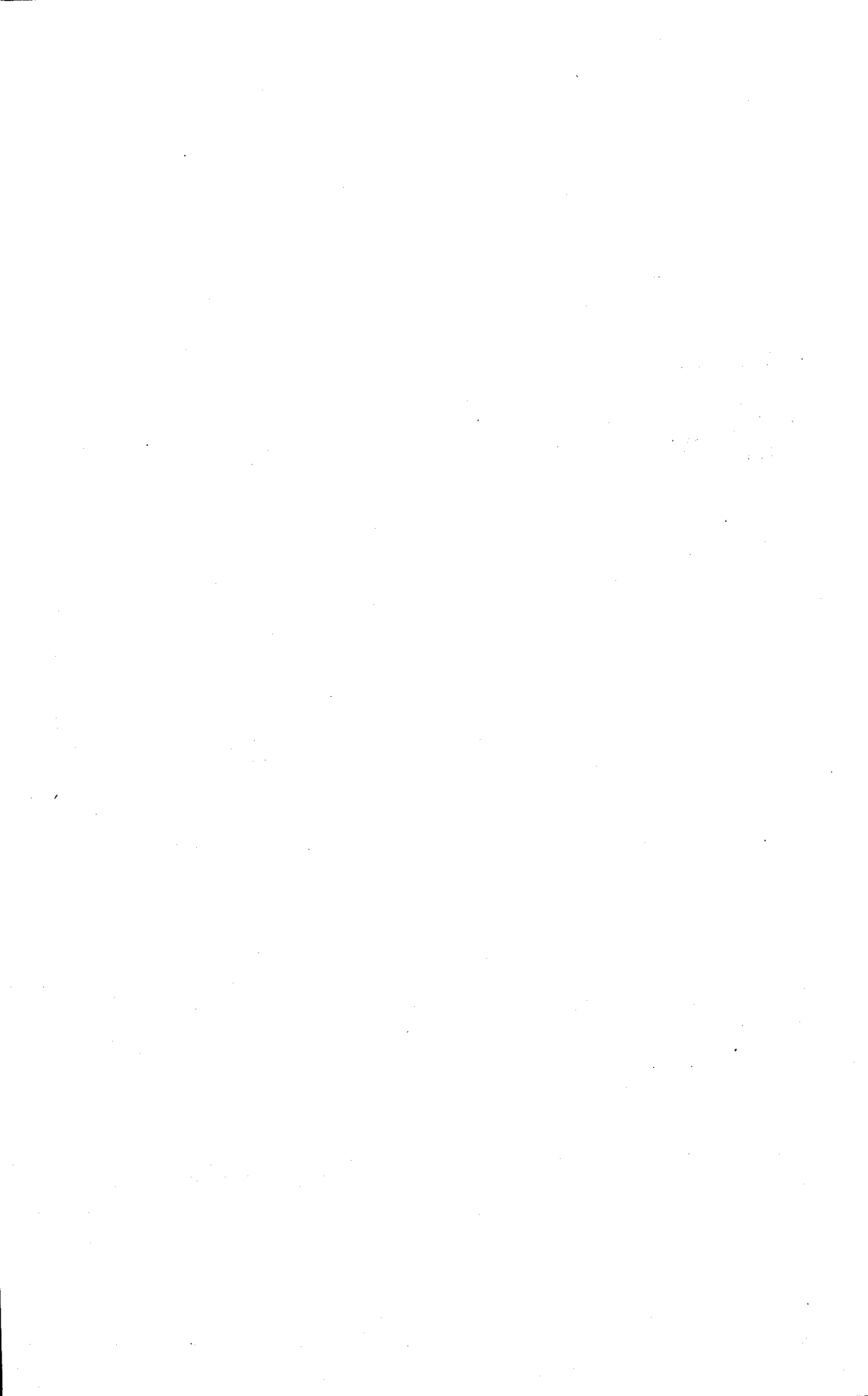
It is hoped that this discussion will serve as a starting-point for the development of useful man-machine interactive systems of the general sort we have been discussing. There is obviously immense scope for research. Part of the intention of the present discussion has been to try to indicate the scope, with the aim of attracting people to the subject no matter how theoretical, or how practical, their particular interests may be. It is important to recognise that it is higher-order logic, and not first-order logic, which is the natural technical framework for the 'mechanization of mathematics'. We have in fact attempted here to pursue further an effort begun previously (Robinson 1968) to persuade those engaged in mechanical theorem-proving research, and those proposing to start such research, to focus their attention henceforth on mechanizing higher-order logic.

THEOREM PROVING

REFERENCES

- Henkin, L. (1950) Completeness in the theory of types. *Journal of Symbolic Logic*, **15**, 81-91.
- Landin, P.J. (1964) The mechanical evaluation of expressions. *Comp. J.*, **6**, 308-20.
- Robinson, J.A. (1968) New directions in mechanical theorem proving. *Proceedings of the IFIP Congress, 1968*.

**DEDUCTIVE
INFORMATION
RETRIEVAL**



Theorem Proving and Information Retrieval

J. L. Darlington

Rheinisch Westfälisches Institut
für Instrumentelle Mathematik,
Bonn

Any information retrieval or IR system that does more than merely search files for stored facts must be able to perform logical inferences. These inferences or deductions are similar in nature to those performed by theorem-proving or question-answering programs, but the deductive systems currently in operation cannot automatically be taken as the model of an IR system, since they will prove theorems or answer questions efficiently only if they are provided with small sets of premises known in advance to be relevant, while the 'data base' of an IR system may be very large indeed, consisting possibly of 10^5 or 10^6 statements (cf. Levien & Maron 1967). Of the total set of premises in the data base, only a few may be relevant to any particular deduction, but some recent refinements in theorem-proving techniques suggest that IR systems may be able to make efficient deductions even in the presence of large sets of irrelevant premises.

We may specify the nature of relevant and irrelevant premises in a deductive system more precisely, as follows: Let S be the total set of premises in the data base, and \bar{T} be the negation of some specific 'fact' or theorem T that one is attempting to 'retrieve' or prove by *reductio ad absurdum*. We shall assume initially that T is a single (though possibly complex) fact, as opposed to a list of possible answers to a question. We shall also assume that S is consistent or 'satisfiable', and that S and \bar{T} are stated in the terminology of first-order predicate calculus with equality, 'Skolemised' and in conjunctive normal form, as required by resolution-type theorem-proving methods (Robinson 1965). Now for any given T , we may distinguish two subsets of S : $S(R)$, the (small) relevant set, and $S(I)$, the (large) irrelevant set, though the composition of these two subsets is not generally known in advance of computation. $S(R)$ may conveniently be defined as the subset of S that appears in the 'proof set' printed out by the 'proof recovery section' of a theorem-proving program. If there is more than one proof of T , that is, more than one way of

deriving the 'null clause' from $S \cup \bar{T}$, then any premise that participates in any of these derivations must be counted as relevant, but $S(R)$ will still be quite small in relation to $S(I)$ in an IR system. The irrelevant set $S(I)$ may in turn be divided into $S(I1)$, which contains no predicates in common with $S(R) \cup \bar{T}$, and $S(I2)$, which shares one or more predicates with $S(R) \cup \bar{T}$. $S(I2)$ may be further divided into $S(I21)$, which contains no literals that 'match' (in the resolution sense) the negations of any literals in $S(R) \cup \bar{T}$, and $S(I22)$, which contains at least one literal that matches the negation of some literal in $S(R) \cup \bar{T}$. The entire set $S \cup \bar{T}$, which one wishes to prove unsatisfiable, is thus represented as the logical sum of five mutually exclusive and jointly exhaustive subsets

$$S(I1) \cup S(I21) \cup S(I22) \cup S(R) \cup \bar{T}$$

in which $S(I1), \dots, S(R)$ are in order of increasing relevance to the proof of T , and in which the irrelevant subsets $S(I1), \dots, S(I22)$ range from 'most irrelevant' to 'least irrelevant', the former being least likely, and the latter most likely, to interfere with the retrieval of T from S . The 'least irrelevant' subset $S(I22)$ in fact plays a crucial role in this classification, since it provides the link between the relevant and the irrelevant premises. If $S(I22)$ is null, or can somehow be neutralised, then the entire irrelevant set $S(I)$ can also be neutralised merely by employing the 'set of support' strategy (Wos *et al.* 1965), the essence of which is that no two clauses in the data base or 'satisfiable set' S are resolved against each other. This strategy entails that all resolvents produced in the first generation will result from

$$S(I22) \cup S(R) \cup \bar{T}$$

with at least one parent in \bar{T} . If $S(I22)$ does not enter into any resolutions with \bar{T} or with any resolvents generated by $S(R) \cup \bar{T}$, as would be the case if for example $S(I22)$ were null, then $S(I1)$ and $S(I21)$ will in effect be neutralised, since no clause will be generated that can call them into play. But suppose on the contrary that $S(I22)$ is not null and contains some clause

$$P(\dots) \vee Q(\dots)$$

such that $P(\dots)$ is 'matchable' by some literal in $S(R) \cup \bar{T}$ and $Q(\dots)$ is not, then it can easily happen that some clause containing $Q(\dots)$ will be generated from $S(I22) \cup S(R) \cup \bar{T}$, and that this clause will resolve with one or more clauses containing $\bar{Q}(\dots)$ in $S(I1) \cup S(I21)$, thereby generating more irrelevant clauses.

The suppositions of the preceding paragraph were confirmed by a simple experiment, designed to test the effect of adding irrelevant premises to sets known in advance to contain only relevant premises. An existing theorem-proving program (Darlington 1968, 1968a), employing set of support with 'unit preference' (Wos *et al.* 1964), was used to solve ten simple problems in predicate calculus taken from pp. 140-1 of I. M. Copi's *Symbolic Logic* (1954),

for example: 'Any book which is approved by all critics is read by every literary person. Anyone who reads anything will talk about it. A critic will approve any book written by any person who flatters him. Therefore if someone flatters every critic then any book he writes will be talked about by all literary persons.' The premises and conclusions of the ten examples were translated by hand from ordinary English into predicate calculus, though computer programs exist for this purpose (Bohnert & Backer 1967; Darlington 1965), and were submitted to the program in two separate runs. In the first run, each conclusion was proved just on the basis of its 'own' premises, that is, $S(I)$ for each T was null, while in the second run the premises were combined, giving a total of 16 unit and 23 non-unit clauses, and each conclusion had to be proved on the basis of all the premises. For rapid access the premises were stored in a 'dictionary' and indexed by the predicates they contained, so that if the program wished to resolve (say) a unit clause $P(\dots)$ against a set of non-unit clauses, it would read in from the dictionary just those non-unit clauses containing $\bar{P}(\dots)$. A similar storage scheme is employed by Green and Raphael (1968). In six of the ten examples $S(I22)$ was null, and for these examples the proofs obtained by the second or 'all premises' run were no longer, in terms of time and of clauses generated, than those obtained by the first or 'own premises'-run, but the remaining four examples, in which $S(I22)$ was not null, took significantly longer in the second run than in the first. It may be assumed in fact-retrieval problems that $S(I22)$ will not in general be null. It is therefore essential to investigate ways of reducing or minimising the effects of $S(I22)$ in such cases, and it is to this problem that we now turn.

A recent strategy that seems to be particularly applicable to IR systems, though it was designed mainly for other theorem-proving applications, is Meltzer's (1968) 'P1-deduction with set of support'. The essence of P1-deduction (Robinson 1965) is that at least one parent of each resolvent must be a 'positive' clause (containing no negated literals). Although P1-deduction and set of support are both complete strategies, they cannot automatically be combined without sacrificing completeness, unless all the positive clauses are in the set of support. In order to bring this about, one may look for ways to 'rename' the predicates (i.e., replace one or more predicates throughout by the negations of their complements) of the data base S so as to make all the premises in S 'nonpositive' (containing at least one negated literal). Any positive clauses would then have to be supplied by the negation of the query addressed to the system. Of course, some satisfiable sets have no nonpositive renaming, e.g., $\{P(f(x)), \bar{P}(g(x))\}$, but there is in fact a strong possibility of finding a nonpositive renaming for the data bases of most IR systems, as the following considerations indicate.

The data bases of most IR systems, as opposed to most axiom sets used in theorem-proving, contain a large diversity of predicates (the set of ten Copi examples, though not strictly speaking an IR system, contains fifty distinct

predicates and their negations, and the examples solved by Burstall's (1967) question-answering program also contain many predicates); and the greater the amount of predicate diversity in a system the greater the chances are of finding a nonpositive renaming. Moreover, the vast majority of unit clauses state positive facts about individuals, and these would all become nonpositive under a renaming of all predicates. Most or all non-unit clauses would also come out nonpositive under a renaming of all predicates, since they are almost all of 'mixed' type, containing some positive and some negative literals. The reason for this is that they result largely from statements of logical implication or equivalence, such as definitions of terms, statements of the form 'all A are B ', statements of symmetry or transitivity of relations, etc. In fact, the conjunctive normal forms of ' A implies B ' and ' A if and only if B ' are always of mixed types so long as A and B are either positive literals or compounds of positive literals containing only conjunction and disjunction signs: in this case, the conjunctive normal form of the negation of A contains only negative literals, and that of B contains only positive literals, so joining the two and converting the result to conjunctive normal form gives mixed clauses.

There are, to be sure, certain irreducible cases in which no renaming will give a nonpositive set: for example, if S contains a positive clause $R(a,b)$ and a nonpositive clause stating that R is asymmetric ($\bar{R}(x,y) \vee \bar{R}(y,x)$) or intransitive ($\bar{R}(x,y) \vee \bar{R}(y,z) \vee \bar{R}(x,z)$); but there are also many cases in which one may avoid resolving a positive against a nonpositive clause in the data base and still preserve completeness. For example, one need not resolve positive unit clauses against nonpositive units, since this could only produce the null clause, contrary to the assumption that S is satisfiable. There is also a large class of 'positive against nonpositive' resolutions that one may be excused from performing by virtue of a method of inference described by Kowalski and Hayes (1969), which is based on the fact that, under certain conditions of order, resolving only on the first literals of clauses is complete. Thus, if $P(\dots)$ and $\bar{P}(\dots)$ do not both occur first in their respective clauses, then these two literals need not be matched. Even if $P(\dots)$ and $\bar{P}(\dots)$ do both occur first, it is frequently possible to change the order of at least the negative literals so that $\bar{P}(\dots)$ no longer occurs first. This may be done if the clause containing $\bar{P}(\dots)$ contains at least one other negative literal, say $\bar{Q}(\dots)$, but there would be no advantage in this if there were also a clause beginning with $Q(\dots)$.

The Kowalski-Hayes method requires one to set up an ' A -ordering' (Slagle 1967) for the literals. To do this, one in effect sets up a total ordering of all the 'ground instances' (containing no free variables) of the 'general level' literals (which may contain free variables). This ordering might for example be purely alphabetical, so that $P(\dots)$ precedes $Q(\dots)$, $P(f(\dots), \dots)$ precedes $P(g(\dots), \dots)$, $P(a)$ precedes $P(f(\dots), \dots)$, etc. A general level literal L_1 is said to precede another general level literal

L_2 in the A -ordering if all the ground instances of L_1 precede all the ground instances of L_2 . Two general level literals, e.g., $P(x)$ and $P(f(a))$, that cannot be so ordered are said to be 'equal' with respect to the A -ordering.

Once an A -ordering has been set up, the Kowalski-Hayes method resolves only on the first literal of each clause, in accordance with a set of conditions that include the following (in addition to certain requirements about 'factoring' of clauses):

- (i) negative literals occur before positive ones;
- (ii) the order for negative literals is fixed for each clause, but may vary from one clause to another;
- (iii) the order for positive literals must be fixed in advance and maintained throughout; and
- (iv) a positive clause whose first n literals are 'equal' with respect to the A -ordering (see above) must be listed n times, with each of these n literals in turn occurring first.

It may be noted that condition (i) and the 'first literal resolution' principle automatically entail $P1$ -deduction.

Condition (ii) allows a great deal of flexibility for those nonpositive clauses that contain more than one negative literal. It means in particular that one need not perform any resolution at all on a nonpositive clause unless there actually is a positive clause beginning with some literal of the form $P(\dots)$ for each negative literal $\bar{P}(\dots)$, since a negative literal that does not have such a 'mate' may always be put first in the clause. One might, of course, include a stronger condition, that each negative literal actually have a match, in which case one is in effect doing 'hyper-resolution' (Robinson 1965), but this procedure has the disadvantage of throwing away partial results that might be useful later on. In order to obtain the maximum advantage from condition (ii), it would be well to employ the most nonpositive renaming of the data base S that one can find. Thus, if a predicate P occurs once negatively and ten times positively, then it is a good idea to rename P and \bar{P} , provided of course that S remains nonpositive under this renaming.

Condition (iii) is best implemented by ordering the positive literals according to increasing ease of matching, so that the positive literal hardest to match occurs first. There are two considerations that determine how hard a literal $P(\dots)$ is to match: (a) the formal complexity of $P(\dots)$, and (b) the number of individuals in the extension of $P(\dots)$. Condition (a) justifies putting m -ary predicates ahead of n -ary ($m > n$), so that ' $R(x, y, z)$ ' (e.g., ' x receives y from z ') goes ahead of ' $G(x)$ ' (e.g., ' x is a gift') and ' $P(x)$ ' (e.g., ' x is a person'), while condition (b) justifies putting $G(x)$ ahead of $P(x)$ under these interpretations, and ' $Ga(x)$ ' (e.g., ' x is a great artist') ahead of both of these. There are certain problems that would arise in implementing conditions (a) and (b) together, such as deciding the relative order

of $Ga(x)$ and $R(x, y, z)$ under these interpretations, but the basic principle of putting the literal hardest to match first seems sound, and even if one makes a few mistakes the resulting system is still complete. If the A -ordering for the positive literals is in accordance with this basic principle, it reduces the chances that any given positive clause $P_1(\dots) \vee P_2(\dots) \vee \dots \vee P_n(\dots)$ will participate in any resolutions, and also increases the chances that if P_i is matched then the hitherto easier to match literals $P_{i+1}(\dots), \dots, P_n(\dots)$ will be more fully instantiated and therefore harder to match than before.

The merits of the above ordering principles may be shown by applying them to an example taken from Leven and Maron (1967), in which it is deduced that Smith graduated from UCLA from a large data base that includes the information (in its 'extensional file') that UCLA awarded Smith a Ph.D. degree, and the definition (in its 'intensional file') that 'graduates from' means 'is awarded a degree by'. Using ' $D(x)$ ' for ' x is a degree', ' $G(x, y)$ ' for ' x graduates from y ', and ' $A(x, y, z)$ ' for ' x awards y to z ', the relevant premises are:

- (1) $\bar{D}(x) \vee \bar{A}(y, x, z) \vee G(z, y)$
- (2) $D(\text{Ph.D.})$
- (3) $A(\text{UCLA}, \text{Ph.D.}, \text{Smith})$

and the negation of the fact to be retrieved is:

- (4) $\bar{G}(\text{Smith}, \text{UCLA}).$

If the premises are made nonpositive by implicitly renaming the three predicates, so that ' $D(x)$ ' is understood to mean ' x is *not* a degree', etc., and if the literals in (1) are ordered by conditions (ii) and (iii), then the set becomes:

- (1') $\bar{G}(z, y) \vee A(y, x, z) \vee D(x)$
- (2') $\bar{D}(\text{Ph.D.})$
- (3') $\bar{A}(\text{UCLA}, \text{Ph.D.}, \text{Smith})$
- (4') $G(\text{Smith}, \text{UCLA})$

and a contradiction may be deduced in three steps:

- (5) $A(\text{UCLA}, x, \text{Smith}) \vee D(x)$ (1') & (4')
- (6) $D(\text{Ph.D.})$ (3') & (5)
- (7) contradiction (2') & (6)

The reason for putting A before D in the A -ordering is obvious in terms of clause (5), since $A(\text{UCLA}, x, \text{Smith})$ will be much more difficult for the other statements in the data base to match than the highly vulnerable $D(x)$.

To summarise the preceding discussion, the method of inference recommended for information retrieval is a combination of Meltzer's 'P1-deduction with set of support' and 'first literal resolution with A -ordering' of Kowalski and Hayes, together with some empirical and logical criteria for choosing a favourable A -ordering. The method is designed to minimise the effect of the

'least irrelevant' set $S(I22)$ on the proof or retrieval of some specific T , since a clause C in $S(I22)$ cannot participate in any resolutions unless for each negative literal $\bar{P}(\dots)$ of C there is a positive clause beginning with $P(\dots)$. Furthermore, supposing that a positive clause D is generated from $S(I22)$, the A -ordering chosen for the positive literals guarantees that D will generate the minimum number of resolvents with the nonpositive clauses. By minimising the effect of $S(I22)$, we thereby minimise the effect of the entire irrelevant set $S(I)$, since $S(I22)$ is the link between $S(I)$ and the minimum unsatisfiable set $S(R) \cup \bar{T}$.

The ten examples mentioned earlier have been recomputed using the new method. The machine generated a total of 52 clauses, with 100 per cent relevance: each generated clause appeared in the proof set of one of the ten examples. This result is very encouraging, but it will have to be confirmed by testing the new method on systems in which $S(I)$ is much larger, and not merely ten times larger, than $S(R)$.

The preceding discussion was concerned with the retrieval of specific facts from a data base S by *reductio ad absurdum*. This method is applicable if one is trying to prove some particular fact T , so that the set $S \cup \bar{T}$ can be formed and is inconsistent or 'unsatisfiable', but there are many cases in information retrieval in which these conditions do not apply and in which one may still obtain useful answers to questions. The following example, based on those solved by Burstall's (1967) program, illustrates this point.

- | | |
|---|--------------------------------|
| (1) $\bar{R}(x) \vee B(x)$ | 'a robin is a bird' |
| (2) $\bar{R}(x) \vee E(x, w)$ | 'robins eat worms' |
| (3) $\bar{H}(x) \vee D(x, a)$ | 'horses drink water' |
| (4) $\bar{D}(x, y) \vee C(x, y)$ | } 'consume means eat or drink' |
| (5) $\bar{E}(x, y) \vee C(x, y)$ | |
| (6) $\bar{C}(x, y) \vee D(x, y) \vee E(x, y)$ | |

Question: 'what do birds consume?' (i.e., give a list of things which are consumed by some or all birds).

As a first approach, one might try to prove the assumption underlying the question, 'there exist birds and there exist things that birds consume', in the hope of generating at least one answer to the question, but if the negation of this assumption, i.e.,

$$(7) \bar{B}(x) \vee \bar{C}(x, y)$$

is added to (1)–(6), the resulting set is still satisfiable (by the $P1$ -deduction theorem), since all clauses are nonpositive.

A more satisfactory approach is to formulate a set $S(A)$ of clauses that express the general conditions that any x has to meet in order to be an answer to the question. Using the predicate Var introduced by Green and Raphael (1968), $S(A)$ for this example consists of the single clause

$$(7') \bar{B}(x) \vee \bar{C}(x, y) \vee Var(y)$$

where $Var(y)$ means essentially that y is an answer. Clause (7') may itself be regarded as an answer to the question, namely, the 'most general answer'. One then treats $S(A)$, or in this case (7'), exactly as if it were a set of support, generating resolvents from $S \cup S(A)$ but not from S alone, and these resolvents will all be in some sense answers to the question, ranging from the most general answer (7') to the most specific answers, that is, fully instantiated unit clauses beginning with Var . Even if, as in the present example, one cannot generate unit Var -clauses, one may still obtain useful answers of lesser specificity, e.g.

$$(8) \bar{R}(x) \vee Var(w)$$

which may be interpreted as 'if robins exist then "worms" is an answer'.

Some additional examples of 'most general answers' are: $\bar{B}(x) \vee Var(x)$ for 'what are birds?'; $\bar{P}(x) \vee \bar{C}(y) \vee \bar{V}(x,y) \vee Var(x)$ for 'what people vote for some candidate?'; and $\bar{P}(x) \vee \bar{C}(y) \vee \bar{V}(x,y) \vee Var(x,y)$ for 'what people vote for what candidates?'. In the latter case, Var is taken as a binary relation. This seems preferable to formulating $S(A)$ twice, with $Var(x)$ and $Var(y)$, since $Var(x,y)$ specifies the order in which x and y must occur. Var may actually be a function of any number of arguments, its degree being determined by the particular system in which it is used.

There is an obvious parallelism between the set $S(A)$ of most general answers and the set \bar{T} that expresses the negation of the assumptions underlying the question, to wit, each clause in $S(A)$ is simply a clause in \bar{T} with an appropriate Var -literal tacked on the end. This is evident, for example, in the case of (7) and (7'). In fact, the *reductio ad absurdum* method might be regarded as a special case of the most general answer method, in which each clause in \bar{T} has the null clause tacked on the end in place of a Var -literal: if all the literals in some clause of \bar{T} are 'cut', then the null clause provides a negative answer to the question 'is $S \cup \bar{T}$ satisfiable?'. Moreover, those cases in which the null clause is generated from some clause in \bar{T} are precisely those cases that provide maximally specific answers from $S(A)$, since they are the cases in which unit Var -clauses are generated, the difference, of course, being that the program keeps on going when it gets to this point and generates more Var -clauses instead of stopping at the first null clause. One would of course like a guarantee that the program will generate *all* answers, but the fact that resolution is not a complete method of forward deduction may complicate the task of proving this for all cases.

There are further important differences between the most general answer method and *reductio ad absurdum*. For example, restrictive strategies do not apply, at least not in the same way, to the most general answer method, since the point is to generate as many answers as possible, rather than to derive the null clause in as few ways as possible. $P1$ -deduction with set of support cannot be applied to a nonpositive satisfiable set, and first literal resolution with A -ordering does not apply either, since if the first literal in a most

general answer cannot be cut (as with $R(x)$ in the example), it may still be possible to obtain a more specific answer by cutting some subsequent literals. The strategy, then, should be geared to cutting as many literals from the most general answers in as many ways as possible, but trying at the same time to avoid deriving the same answer twice. If, however, one can assume that the set is unsatisfiable except for the *Var*-literals, then the *reductio ad absurdum* strategies become more applicable to the most general answer method. The advantages in being able to apply these strategies are very considerable, particularly in IR systems with a large measure of predicate diversity, where 'P1-deduction with set of support' and 'first literal resolution with *A*-ordering' offer a reasonable hope of obtaining efficient proofs from data bases containing large sets of irrelevant premises.

Acknowledgements

The author is indebted to Professor G. Hasenjaeger, Dr B. Meltzer, Dr C.A. Petri and Mr R. Kowalski for valuable discussions and useful ideas. Computing time for this project is being provided on the IBM 7090 by the Institut für Instrumentelle Mathematik in Bonn.

REFERENCES

- Bohnert, H.G. & Backer, P.O. (1967) *Automatic English-to-Logic Translation in a Simplified Model*. IBM Research Paper RC-1744. Yorktown Heights, New York: IBM Watson Research Center.
- Burstall, R.M. (1967) A combinatory approach to relational question answering and syntax analysis. Copenhagen: Nato Summer School.
- Copi, I.M. (1954) *Symbolic Logic*. New York: The Macmillan Company.
- Darlington, J.L. (1965) Machine methods for proving logical arguments expressed in English. *Mechanical Translation*, 8, pp. 41-67.
- (1968) Some theorem-proving strategies based on the resolution principle. *Machine Intelligence 2*, pp. 57-71 (ed. Dale, E. & Michie, D.). Edinburgh: Oliver & Boyd.
- (1968a) Automatic theorem proving with equality substitutions and mathematical induction. *Machine Intelligence 3*, pp. 113-27 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
- Green, C.C. & Raphael, B. (1968) The use of theorem-proving techniques in question-answering systems. 1968 ACM Proceedings, 27-29 August.
- Kowalski, R. & Hayes, P.J. (1969) Semantic trees in automatic theorem proving. *Machine Intelligence 4*, pp. 87-101 (eds. Meltzer, B. & Michie, D.) Edinburgh: Edinburgh University Press.
- Levien, R.E. & Maron, M.E. (1967) A computer system for inference execution and data retrieval. *C. Ass. Comput. Mach.*, 10, 715-21.
- Meltzer, B. (1968) Some notes on resolution strategies, *Machine Intelligence 3*, pp.71-5 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
- Robinson, J.A. (1965) Automatic deduction with hyperresolution. *Int. J. Comput. Math.*, 1, 227-34.
- Slagle, J.R. (1967) Automatic theorem proving with renamable and semantic resolution. *J. Ass. Comput. Mach.*, 14, 687-97.
- Wos, L.T., Carson D.F. & Robinson, G.A. (1964) The unit preference strategy in theorem proving, *AFIPS*, 26, Fall, J.C.C., 615-21. Washington, D.C.: Spartan Books.
- (1965) Efficiency and completeness of the set of support strategy in theorem proving. *J. Ass. Comput. Mach.*, 12, 536-41.



Theorem-Proving by Resolution as a Basis for Question-Answering Systems

Cordell Green

Stanford Research Institute
Menlo Park, California

ABSTRACT

This paper shows how a question-answering system can be constructed using first-order logic as its language and a resolution-type theorem-prover as its deductive mechanism. A working computer-program, QA3, based on these ideas is described. The performance of the program compares favorably with several other general question-answering systems.

1. QUESTION ANSWERING

A question-answering system accepts information about some subject areas and answers questions by utilizing this information. The type of question-answering system considered in this paper is ideally one having the following features:

1. A language general enough to describe any reasonable question-answering subjects and express desired questions and answers.
2. The ability to search efficiently the stored information and recognize items that are relevant to a particular query.
3. The ability to derive an answer that is not stored explicitly, but that is derivable by the use of moderate effort from the stored facts.
4. Interactions between subject areas; for example, if the system has facts about Subject A and Subject B, then it should be able to answer a question that requires the use of both sets of facts.
5. Capability of allowing the user to add new facts or replace old facts conveniently.

This paper argues the case for formal methods to achieve such a system and presents one particular approach in detail. A natural language facility is not one of the properties sought after or discussed (although Coles, 1968, has

added to the program described here a translator from a subset of English to first-order logic).

The name 'question-answering system' requires clarification. The system described above might be named an 'advice taker' or a 'multi-purpose problem-solving system' or 'general problem-solving system'. McCarthy (1958) proposed using formal languages and deduction to construct such a system, and suggested allowing the user to give hints or advice on how to answer a question; he referred to the proposed system as an 'advice taker'. Research on 'multi-purpose' or 'general problem-solving' tends to differ from question-answering as described above by placing more emphasis on solving deeper, more difficult problems and less emphasis on user interaction, formality, and efficient retrieval of relevant facts from a large data base. The situation is further confused by the use of 'question-answering' to refer sometimes to natural language systems, sometimes to information retrieval systems having little deductive ability, and sometimes to systems with deductive ability limited to the propositional calculus.

It is important to emphasize the distinction between general versus special-purpose question-answering. If the class of questions asked of a system is small, completely specified in advance, and concerned with a particular subject area, such as the question-answering system of Green, Wolf, Chomsky, and Laughery (1963) concerned with baseball, or that of Lindsay (1963) concerned with family relations, then we shall call such a system 'special-purpose'. Frequently the goal in designing a special-purpose system is to achieve good performance, measured in terms of running speed and memory utilization. In this case the best approach may be first to construct a special data base or memory that is optimized for that subject area and question class, and then to write special question-answering subroutines that are optimized for the particular data base and question class. On the other hand, a 'general' question-answering system is one that allows arbitrary subject areas, arbitrary questions, and arbitrary interactions between subject areas during the process of answering a question. This paper describes a rather formal approach to designing a general question-answering system. A precise name for our system is 'a general, formal, deductive, question-answering system.'

2. THEOREM-PROVING

The use of a theorem-prover as a question-answering system can be explained very simply. The question-answering system's knowledge of the world is expressed as a set of axioms, and the questions asked it are presented as theorems to be proved. The process of proving the theorem is the process of deducing the answer to the question. For example, the fact 'George is at home', is presented as the axiom, $AT(\text{George}, \text{home})$. The question 'Is George at home?' is presented as the conjectured theorem, $AT(\text{George}, \text{home})$. If this theorem is proved true, the answer is yes. (In this simple example the theorem is obviously true

since the axiom *is* the theorem.) The theorem-prover can also be used to find or construct an object satisfying some specified conditions. For example, the question 'Where is George?' requires finding the place x satisfying $AT(George, x)$. The theorem-prover is embedded in a system that controls the theorem-prover, manages the data base, and interacts with the user. These ideas are explained in more detail below.

Even though it is clear that theorem-proving can be used for question-answering, why should one use these very formal methods? Theorem-proving may be a good approach to the achievement of generality for several reasons:

1. The language is well defined, unambiguous, and rather general, so that one may hope to describe many desired subjects, questions, or answers.
2. The proof procedure used allows all possible interaction among the axioms and is logically 'complete' that is, if a theorem is a logical consequence of the axioms, then this procedure will find a proof, given enough time and space. This completeness property is important since several general question-answering programs have resulted in incomplete deductive systems, even in the practical sense of being unable to answer some simple types of questions that are short, reasonable deductions from the stored facts — for example, the author's QAI (Green and Raphael 1968), Raphael's SIR (1964), Slagle's DEDUCOM (1965), and Safier's SIMPLE SIMON (1965). (However, the fact that we use a first-order logic theorem-prover does impose certain important restrictions discussed in section 5.)
3. The theorem-prover is subject-independent, so that to describe a new subject or modify a previous description of a subject, only the axioms need to be changed, and it is not necessary to make any changes in the program.
4. Formal techniques such as those developed here may be generally valuable to the field of artificial intelligence. The use of a formal framework can lead to insights and generalizations that are difficult to develop while working with an *ad hoc* system. A common, well-defined framework facilitates communication between researchers, and helps to unify and relate diverse results that are difficult to compare.
5. Theorem-provers are becoming more efficient. Even though the theorem-proving method used is theoretically complete, in practice its ability to find proofs is limited by the availability of computer time and storage space. However, the method of 'Resolution' (Robinson 1965), used by the program described here, has been developed to the point of having several good heuristics. Further improvements in theorem-proving are very likely, and, hopefully, the improvements will carry over into corresponding improvements in question-answering.

It should be possible to communicate precisely new theorem-proving results to other researchers, and it is relatively easy to communicate precisely particular formalizations or axiomatizations of subjects.

3. EXTENDING THEOREM-PROVING TO QUESTION-ANSWERING

This section describes, in general, how questions can be asked in first-order logic, and how answers can be generated. Examples illustrating these methods are presented. The discussion in this section and the following two assumes that the reader is somewhat familiar with logic and automatic theorem-proving. An introduction to automatic theorem-proving is given in Cooper (1966) and Davis (1963). The theorem-proving methods mentioned in this paper use the Resolution Principle proposed by J. A. Robinson (1965 and 1967). Additional strategies for using the Resolution principle are presented by Wos *et al.* (1964, 1965 and 1967). This last paper defines terms the 'Extended Set of Support' strategy, 'degree', and 'singly connectedness', that are used in section 4.

The explanation of question-answering given in this section will be illustrated primarily by the techniques used in a working question-answering program called QA3. It is programmed in LISP on the SDS 940 computer, operating in the time-sharing mode. The user works at a teletype, entering statements and questions, and receiving replies. The notation in this paper is slightly different from the actual computer input and output, as the character set available on the teletype does not contain the symbols we use here. QA3 is an outgrowth of QA2 (Green and Raphael 1968), an earlier system, but is somewhat more sophisticated and practical, and is now being used for several applications.

1. Types of questions and answers

Facts are presented as statements of first-order logic. The statement is preceded by STATEMENT to indicate to the program that it is a statement. These statements (axioms) are automatically converted to clauses and stored in the memory of the computer. The memory is a list structure indexed by the predicate letters, function symbols, and constant symbols occurring in each clause. A statement can be a very specific fact such as

STATEMENT: *COLOR(book, red)*

corresponding to the common attribute-object-value triple. A statement can also be a more general description of relations, such as:

STATEMENT: $(\forall x)(\forall A)(\forall B)[A \subseteq B \wedge x \in A \Rightarrow x \in B]$

meaning that if A is a subset of B and if x is an element of A , then x is an element of B .

Questions are also presented as statements of first-order logic. QUESTION is typed before the question. This question becomes a conjecture and QA3

attempts to prove the conjecture in order to answer *YES*. If the conjecture is not proved, QA3 attempts to prove the negation of this question in order to answer *NO*. The theorem-prover attempts a proof by refutation. During the process of searching for a proof, clauses that may be relevant to a proof are extracted from memory and utilized as axioms. If the question is neither proved nor disproved, then a *NO PROOF FOUND* answer is returned. *ANSWER* indicates an answer.

We now present a very simple dialogue with QA3. The dialogue illustrates a 'yes' answer, a 'no' answer, and an 'or' answer. Questions 4, 7, and 8 below illustrate questions where the answer is a term generated by the proof procedure. These kinds of answers will be called 'constructive' answers.

1. The first fact is 'Smith is a man.'

STATEMENT: $MAN(Smith)$

OK

The OK response from QA3 indicates that the statement is accepted, converted to a clause, and stored in memory.

2. We ask the first question, 'Is Smith a man?'

QUESTION: $MAN(Smith)$

ANSWER: *YES*

3. We now state that 'Man is an animal,' or, more precisely, 'If x is a man then x is an animal.'

STATEMENT: $(\forall x)[MAN(x) \Rightarrow ANIMAL(x)]$

OK

4. We now ask 'Who is an animal?' This question can be restated as 'Find some y that is an animal' or 'Does there exist a y such that y is an animal? If so, exhibit such a y .'

QUESTION: $(\exists y)ANIMAL(y)$

ANSWER: *YES, $y = Smith$*

The *YES* answer indicates that the conjecture $(\exists y)ANIMAL(y)$ has been proved (from statements 1 and 3 above). ' $y = Smith$ ' indicates that '*Smith*' is an instance of y satisfying $ANIMAL(y)$ —i.e., $ANIMAL(Smith)$ is a theorem.

5. Fact: A robot is a machine.

STATEMENT: $(\forall x)[ROBOT(x) \Rightarrow MACHINE(x)]$

OK

6. Fact: Rob is a robot.

STATEMENT: $ROBOT(Rob)$

OK

DEDUCTIVE INFORMATION RETRIEVAL

7. Fact: No machine is an animal.

STATEMENT: $(\forall x)[MACHINE(x) \Rightarrow \sim ANIMAL(x)]$

OK

8. The question 'Is everything an animal?' is answered *NO*.
A counterexample is exhibited – namely, Rob the robot.

QUESTION: $(\forall x)ANIMAL(x)$

ANSWER: *NO*, $x = Rob$

The answer indicates that $\sim ANIMAL(Rob)$ is a theorem. Note that a *NO* answer produces a counterexample for the universally quantified variable x . This is the dual of the construction of a satisfying instance for an existentially quantified variable in a question answered *YES*.

9. Fact: Either Smith is at work or Jones is at work.

STATEMENT: $AT(Smith,work) \vee AT(Jones,work)$

OK

10. Question: 'Is any one at work?'

QUESTION: $(\exists x)(AT(x,work))$

ANSWER: *YES*, $x = Smith$

or $x = Jones$

From the previous statement it is possible to prove that someone is at work, although it is not possible to specify a unique individual.

Statements, questions, and answers can be more complex so that their corresponding English form is not so simple. Statements and questions can have many quantifiers and can contain functions. The answer can also contain functions. Consider the question 'Is it true that for all x there exists a y such that $P(x,y)$ is true?', where P is some predicate letter. Suppose QA3 is given the statement,

11. STATEMENT: $(\forall z)P(z,f(z))$

where f is some function. We ask the question

12. QUESTION: $(\forall x)(\exists y)P(x,y)$

ANSWER: *YES*, $y = f(x)$

Notice that the instance of y found to answer the question is a function of x , indicating the dependence of y on x . Suppose that instead of statement 11 above, QA3 has other statements about P . An answer to question 12 might be

ANSWER: *NO*, $x = a$

where a is some instance of x that is a counterexample.

The term(s) that is the answer can be either a constant, a function, a variable, or some combination thereof. If the answer is a constant or a known function, then the meaning of the answer is clear. However, the answer may

be a Skolem function generated by dropping existential quantifiers. In this case, the answer is an object asserted to exist by the existential quantifier that generated the Skolem function. To know the meaning of this Skolem function, the system must exhibit the original input statement that caused the production of the Skolem function. Free variables in clauses correspond to universally quantified variables, so if the answer is a free variable, then any term satisfies the formula and thus answers the question.

Two more types of answers are *NO PROOF FOUND* and *INSUFFICIENT INFORMATION*. Suppose the theorem-prover fails to prove some conjecture and also fails to disprove the conjecture. If the theorem-prover runs out of time or space during either the attempted 'yes' proof or the attempted 'no' proof, then there is the possibility that some proof is possible if more time or space is available. The answer in this case is *NO PROOF FOUND*.

Now suppose both proof attempts fail without exceeding any time or space limitations. The theorem-proving strategy is complete so that if no time or space limitation halts the search for a proof and the conjecture is a logical consequence of the axioms, then a proof will be found. So we know that neither a 'yes' nor a 'no' answer is possible from the given statements. The answer returned is *INSUFFICIENT INFORMATION*. For example, suppose QA3 has no statements containing the predicate letter 'R':

QUESTION: $(\exists x)R(x)$

The negated question is the clause $\{\sim R(x)\}$, and no other clauses in the memory of QA3 can resolve with it. Thus the system will respond

ANSWER: *INSUFFICIENT INFORMATION*.

2. Constructing answers

The Resolution method of proving theorems allows us to produce correct constructive answers. This means that if, for example, $(\exists x)P(x)$ is a theorem then the proof procedure can find terms t_1, t_2, \dots, t_n such that $P(t_1) \vee P(t_2) \vee \dots \vee P(t_n)$ is a theorem.

First, we shall present some examples of answer construction. After these examples we shall show how a proof by resolution can be used to generate an answer.

Examples of answer construction will be explained by means of the *ANSWER* predicate used by QA3 to keep track of instantiations. Consider the question

QUESTION: $(\exists y)ANIMAL(y)$

which is negated to produce the clause

$\{\sim ANIMAL(y)\}$.

The special literal, *ANSWER*(y), is added to this clause to give

$\{\sim ANIMAL(y) \vee ANSWER(y)\}$.

DEDUCTIVE INFORMATION RETRIEVAL

The proof process begins with this clause. When the literal $ANIMAL(x)$ is resolved against the literal $\sim ANIMAL(y)$, the term y is instantiated to yield the term x . In the new clause resulting from this resolution, the argument of $ANSWER$ is then x . In the next resolution the argument of $ANSWER$ becomes *Smith*. We list the complete modified proof that terminates with the clause

$\{ANSWER(Smith)\}$.

1. $\{\sim ANIMAL(y) \vee ANSWER(y)\}$ Modified negation of the question.
2. $\{\sim MAN(x) \vee ANIMAL(x)\}$ Axiom fetched from memory.
3. $\{\sim MAN(x) \vee ANSWER(x)\}$ From resolving 1 and 2.
4. $\{MAN(Smith)\}$ Axiom fetched from memory.
5. $\{ANSWER(Smith)\}$ 'Contradiction' from 3 and 4 for $y=Smith$.

The argument of the $ANSWER$ predicate is the instance of y – namely, *Smith* – that answers the question. QA3 returns

ANSWER: YES, $y=Smith$.

This answer means, as will be explained later, that

$ANIMAL(Smith)$

is a theorem.

The $ANSWER$ literal is added to each clause in the negation of the question. The arguments of $ANSWER$ are the existentially quantified variables in the question. When a new clause is created, each $ANSWER$ literal in the new clause is instantiated in the same manner as any other literal from the parent clause. However, the $ANSWER$ literal is treated specially; it is considered to be invisible to resolution in the sense that no literal is resolved against it and it does not contribute to the length (size) of the clause containing it. We call a clause containing only $ANSWER$ literals an 'answer clause.' The search for an answer (proof) successfully terminates when an answer clause is generated. The addition of the $ANSWER$ predicate to the clauses representing the negation of the theorem does not affect the completeness of this modified proof procedure. The theorem-prover generates the same clauses, except for the $ANSWER$ predicate, as the conventional theorem-prover. Thus in this system an answer clause is equivalent to the empty clause that establishes a contradiction in a conventional system.

An answer clause specifies the sets of values that the existentially quantified variables in the question may take in order to preserve the provability of the question. The precise meaning of the answer will be specified in terms of a question Q that is proved from a set of axioms $B = \{B_1, B_2, \dots, B_b\}$.

As an example illustrating some difficulties with Skolem functions, let the axioms B consist of a single statement,

STATEMENT: $(\forall z)(\exists w)P(z,w)$

Suppose this is converted to the clause

$\{P(z,f(z))\}$,

where $f(z)$ is the Skolem function due to the elimination of the quantifier $(\exists w)$. We ask the question Q ,

QUESTION: $(\forall y)(\exists x)P(y,x)$.

The negation of the question is $\sim Q$,

$(\exists y)(\forall x)\sim P(y,x)$.

The clause representing $\sim Q$ is

$\{\sim P(b,x)\}$,

where b is the constant (function of no variables) introduced by the elimination of $(\exists y)$. The proof, obtained by resolving these two clauses, yields the answer clause

$\{ANSWER(f(b))\}$.

The Skolem Function b is replaced by y , and the answer printed out is

ANSWER: YES, $x=f(y)$. (1)

At present in QA3 the Skolem function $f(y)$ is left in the answer. To help see the meaning of some Skolem function in the answer, the user can ask the system to display the original statement that, when converted to clauses, caused the generation of the Skolem function.

As an illustration, consider the following interpretation of the statement and question of this example. Let $P(u,v)$ be true if u is a person at work and v is this person's desk. Then the statement $(\forall z)(\exists w)P(z,w)$ asserts that every person at work has a desk, but the statement does not name the desk. The Skolem function $f(z)$ is created internally by the program during the process of converting the statement $(\forall z)(\exists w)P(z,w)$ into the clause $\{P(z,f(z))\}$. The function $f(z)$ may be thought of as the program's internal name for z 's desk. (The term $f(z)$ could perhaps be written more meaningfully in terms of the descriptive operator i as ' $iw.P(z,w)$,' i.e., 'the w such that $P(z,w)$ ', although w is not necessarily unique.)

The question $(\forall y)(\exists x)P(y,x)$ asks if for every person y there exists a corresponding desk. The denial of the question, $(\exists y)(\forall x)\sim P(y,x)$, postulates that there exists a person such that for all x , it is not the case that x is his desk. The Skolem function of no arguments, b , is also created internally by the

program as it generates the clause $\{\sim P(b,x)\}$. The function b is thus the program's internal name for the hypothetical person who has no desk.

The one-step proof merely finds that b does have a desk, namely $f(b)$. The user of the system does not normally see the internal clause representations unless he specifically requests such information. If the term $f(b)$ that appears in the answer clause were given to the user as the answer, e.g. *YES, $x=f(b)$* , the symbols f and b would be meaningless to him. But the program remembers that b corresponds to y , so b is replaced by y , yielding a slightly more meaningful answer, *YES, $x=f(y)$* . The user then knows that y is the same y he used in the question. The significance of the Skolem function f is slightly more difficult to express. The program must tell the user where f came from. This is done by returning the original statement $(\forall z)P(z,f(z))$ to the user (alternatively, the descriptive operator could be used to specify that $f(z)$ is *iw.P(z,w)*). As a rule, the user remembers, or has before his eyes, the question, but the specific form of the statements (axioms) is forgotten. In this very simple example the meaning of f is specified completely in terms of the question predicate P , but in general the meanings of Skolem functions will be expressed in terms of other predicates, constants, etc.

We will now show how to construct an 'answer statement', and then we will prove that the answer statement is a logical consequence of the axiom clauses. The user may require that an answer statement be exhibited, in order better to understand a complicated answer.

Consider a proof of question Q from the set of axioms $B = \{B_1, B_2, \dots, B_b\}$. B logically implies Q if and only if $B \wedge \sim Q$ is unsatisfiable. The statement $B \wedge \sim Q$ can be written in prenex form $PM(Y, X)$, where P is the quantifier prefix, $M(Y, X)$ is the matrix, $Y = \{y_1, y_2, \dots, y_u\}$ is the set of existentially quantified variables in P , and $X = \{x_1, x_2, \dots, x_e\}$ is the set of universally quantified variables in P .

Eliminating the quantifier prefix P by introducing Skolem functions to replace existential quantifiers and dropping the universal quantifiers produces the formula $M(U, X)$. Here U is the set of terms $\{u_1, u_2, \dots, u_u\}$, such that for each existentially quantified variable y_i in P , u_i is the corresponding Skolem function applied to all the universally quantified variables in P preceding y_i . Let $M(U, X)$ be called S . The statement $B \wedge \sim Q$ is unsatisfiable if and only if the corresponding statement S is unsatisfiable. Associated with S is a Herbrand Universe of terms H that includes X , the set of free variables of S . If $\phi = \{t_1/x_1, t_2/x_2, \dots, t_n/x_n\}$ represents a substitution of terms t_1, t_2, \dots, t_n from H for the variables x_1, x_2, \dots, x_n , then $S\phi$ denotes the instance of S over H formed by substituting the terms t_1, t_2, \dots, t_n from H for the corresponding variables x_1, x_2, \dots, x_n in S .

Let S_i represent a variant of S , i.e., a copy of S with the free variables renamed. Let the free variables be renamed in such a way that no two variants S_i and S_j have variables in common. By the Skolem-Löwenheim-Gödel theorem (Robinson 1967), S is unsatisfiable if and only if there exists an

instance of a finite conjunction of variants of S that is truth-functionally unsatisfiable. A resolution theorem prover can be interpreted as proving S unsatisfiable by finding such a finite conjunction.

Suppose the proof of Q from B finds the conjunction $S_1 \wedge S_2 \wedge \dots \wedge S_k$ and the substitution θ such that

$$(S_1 \wedge S_2 \wedge \dots \wedge S_k)\theta$$

is truth-functionally unsatisfiable. Let F_0 denote the formula $(S_1 \wedge S_2 \wedge \dots \wedge S_k)\theta$. Let L be the conjunction of variants of $M(Y, X)$,

$$L = M(Y_1, X_1) \wedge M(Y_2, X_2) \wedge \dots \wedge M(Y_k, X_k)$$

and let λ be the substitution of Skolem function terms for variables such that

$$\begin{aligned} L\lambda &= M(U_1, X_1) \wedge M(U_2, X_2) \wedge \dots \wedge M(U_k, X_k) \\ &= S_1 \wedge S_2 \wedge \dots \wedge S_k. \end{aligned}$$

Thus $L\lambda\theta = F_0$.

Before constructing the answer statement, observe that the Skolem functions of F_0 can be removed as follows. Consider the set $U = \{u_1, u_2, \dots, u_n\}$ of Skolem-function terms in S . Find in F_0 one instance, say u'_1 , of a term in U . Select a symbol, z_1 , that does not occur in F_0 . Replace every occurrence of u'_1 in F_0 by z_1 , producing statement F_1 . Now again apply this procedure to F_1 , substituting a new variable throughout F_1 for each occurrence of some remaining instance of a Skolem-function term in F_1 , yielding F_2 . This process can be continued until no further instances of terms from U are left in F_n , for some n .

The statement F_i for $0 \leq i \leq n$ is also truth-functionally unsatisfiable for the following reasons. Consider any two occurrences of atomic formulae, say m_a and m_b , in F_0 . If m_a and m_b in F_0 are identical, then the corresponding two transformed atomic formulae m_{a1} and m_{b1} in F_2 are identical. If m_a and m_b are not identical, then m_{a1} and m_{b1} are not identical. Thus, F_1 must have the same truth table, hence truth value, as F_0 . This property holds at each step in the construction, so F_0, F_1, \dots, F_n must each be truth-functionally unsatisfiable.

This term replacement operation can be carried out directly on the substitutions, i.e., for each statement F_i , $0 \leq i \leq n$, there exists a substitution σ_i such that $F_i = L\sigma_i$. We prove this by showing how such a σ_i is constructed. Let $\sigma_0 = \lambda\theta = \{t_1/v_1, t_2/v_2, \dots, t_p/v_p\}$.

By definition, $F_0 = L\sigma_0$. Let t'_j denote the term formed by replacing every occurrence of u'_1 in t_j by z_1 . The substitution $\sigma_1 = \{t'_1/v_1, t'_2/v_2, \dots, t'_p/v_p\}$ applied to L yields F_1 , i.e., $F_1 = L\sigma_1$. Similarly one constructs σ_i and shows, by induction, $F_i = L\sigma_i$, for $0 \leq i \leq n$.

Now let us examine some of the internal structure of F_0 . Assume that $S = M(U, X)$ is formed as follows. The axioms may be represented as $P_B B(Y_B, X_B)$, where P_B is the quantifier prefix, Y_B is the set of universally-

quantified variables, and X_B is the set of existentially-quantified variables. These axioms are converted to a set of clauses denoted by $B(Y_B, U_B)$, where U_B is the set of Skolem-function terms created by eliminating X_B .

The question may be represented as $P_Q Q(Y_Q, X_Q)$, where P_Q is the quantifier prefix, Y_Q is the set of universally-quantified variables, and X_Q is the set of existentially-quantified variables. Assume that the variables of the question are distinct from the variables of the axioms. The negation of the question is converted into a set of clauses denoted by $\sim Q(U_Q, X_Q)$, where U_Q is the set of Skolem-function terms created by eliminating Y_Q . The function symbols in U_Q are distinct from the function symbols in U_B . Thus $M(U, X) = [B(Y_B, U_B) \wedge \sim Q(U_Q, X_Q)]$. Now let $L_B = [B(Y_{B1}, X_{B1}) \wedge B(Y_{B2}, X_{B2}) \wedge \dots \wedge B(Y_{Bk}, X_{Bk})]$ and let $\sim L_Q = [\sim Q(Y_{Q1}, X_{Q1}) \wedge \sim Q(Y_{Q2}, X_{Q2}) \wedge \dots \wedge \sim Q(Y_{Qk}, X_{Qk})]$. Thus $L = L_B \wedge \sim L_Q$.

Observe that one can construct a sequence of statements F_0, F'_1, \dots, F'_m similar to F_0, F_1, \dots, F_n in which the only terms replaced by variables are instances of terms in U_Q . This construction terminates when for some m the set of clauses F'_m contains no further instances of terms in U_Q . By the same argument given earlier for the formulas F_i , each formula F'_i is truth-functionally unsatisfiable. Similarly one can construct a sequence of substitutions $\sigma_0, \sigma'_1, \dots, \sigma'_m$ such that $L\sigma'_i = F'_i$ for $0 \leq i \leq m$. Let $\sigma = \sigma'_m$. Substitute σ into L_Q , forming

$$L_Q\sigma = [Q(Y_{Q1}, X_{Q1})\sigma \vee Q(Y_{Q2}, X_{Q2})\sigma \vee \dots \vee Q(Y_{Qk}, X_{Qk})\sigma].$$

Since σ replaces the elements of Y_{Qj} by variables, let the set of variables Z_{Qj} denote $Y_{Qj}\sigma$. Thus

$$L_Q\sigma = [Q(Z_{Q1}, X_{Q1}\sigma) \vee Q(Z_{Q2}, X_{Q2}\sigma) \vee \dots \vee Q(Z_{Qk}, X_{Qk}\sigma)].$$

Now, let Z be the set of all variables occurring in $L_Q\sigma$. The *answer statement* is defined to be $(\forall Z) L_Q\sigma$. In its expanded form the answer statement is

$$(\forall Z)[Q(Z_{Q1}, X_{Q1}\sigma) \vee Q(Z_{Q2}, X_{Q2}\sigma) \vee \dots \vee Q(Z_{Qk}, X_{Qk}\sigma)]. \quad (2)$$

We now prove that the answer statement is a logical consequence of the axioms in their clausal form. Suppose not, then $B(Y_B, U_B) \wedge \sim(\forall Z) L_Q\sigma$ is satisfiable, thus $B(U_B, X_B) \wedge (\exists Z) \sim L_Q\sigma$ is satisfiable, implying that the conjunction of its instances $L_B\lambda \wedge (\exists Z) \sim L_Q\sigma$ is satisfiable. Now drop the existential quantifiers $(\exists Z)$. Letting the elements of Z in $\sim L_Q\sigma$ denote a set of constant symbols or Skolem functions of no arguments, the resulting formula $L_B\lambda \wedge \sim L_Q\sigma$ is also satisfiable.

Note that $L_B\sigma$ is an instance of $L_B\lambda$. To see this, let λ_B be the restriction of λ to variables in L_B . Thus, $L_B\lambda = L_B\lambda_B$. Suppose $\theta = \{r_1/w_1, r_2/w_2, \dots, r_n/w_n\}$. Recall that σ is formed from $\lambda\theta$ by replacing in the terms of $\lambda\theta$ occurrences of instances u'_q of 'question' Skolem terms by appropriate variables. (The 'axiom' Skolem functions are distinct from question Skolem functions and occur only in the terms of λ_B .) Thus no such u'_q is an instance of an axiom Skolem term, therefore each occurrence of each such u'_q in $\lambda_B\theta$ must arise

from an occurrence of u'_i in some r_j in θ . It follows then that $L_B\sigma = L_B\lambda_B\phi$ where $\phi = \{r'_1/w_1, r'_2/w_2, \dots, r'_n/w_n\}$ is formed from θ by replacing each u'_i in each r_j by an appropriate variable. Since $L_B\lambda = L_B\lambda_B$, $L_B\lambda\phi = L_B\sigma$. Since the only free variables of $L_B\lambda \wedge \sim L_Q\sigma$ occur in $L_B\lambda$, $[L_B\lambda \wedge \sim L_Q\sigma] \phi = L_B\lambda\phi \wedge \sim L_Q\sigma$.

The formula $L_B\lambda \wedge \sim L_Q\sigma$ logically implies all of its instances, in particular the instance $L_B\lambda\phi \wedge \sim L_Q\sigma$. Thus, if $L_B\lambda \wedge \sim L_Q\sigma$ is satisfiable, its instance $L_B\lambda\phi \wedge \sim L_Q\sigma$ is satisfiable. Since $[L_B\lambda\phi \wedge \sim L_Q\sigma] = [L_B\sigma \wedge \sim L_Q\sigma] = [L_B \wedge \sim L_Q]\sigma = L\sigma = F'_m$ for some m , F'_m must be satisfiable. This contradicts our earlier result that F'_m is truth-functionally unsatisfiable, and thus proves that the answer statement is a logical consequence of the axioms.

We make one further refinement of the answer statement (2). It is unnecessary to include the j th disjunct if $X_{Q_j}\sigma = X_{Q_j}$, i.e., if σ does not instantiate X_{Q_j} . Without loss of generality, we can assume that for $r \leq k$, the last $k-r$ disjuncts are not instantiated, i.e.,

$$X_{Q_{r+1}}\sigma = X_{Q_{r+1}}, X_{Q_{r+2}}\sigma = X_{Q_{r+2}}, \dots, X_{Q_k}\sigma = X_{Q_k}.$$

Then the stronger answer statement

$$(\forall Z)[Q(Z_{Q_1}, X_{Q_1}\sigma) \vee Q(Z_{Q_1}, X_{Q_2}\sigma) \vee \dots \vee Q(Z_{Q_r}, X_{Q_r}\sigma)] \quad (3)$$

is logically equivalent to (2). (Since the matrix of (3) is a sub-disjunct of (2), (3) implies (2). If $j \leq r$, the j th disjunct of (2) implies the j th disjunct of (3). If $r < j \leq k$, the j th disjunct of (2) implies all of its instances, in particular all disjuncts of (3).)

The *ANSWER* predicate provides a simple means of finding the instances of Q in (3). Before the proof attempt begins, the literal *ANSWER*(X_Q) is added to each clause in $\sim Q(U_Q, X_Q)$. The normal resolution proof procedure then has the effect of creating new variants of X_Q as needed. The j th variant, *ANSWER*(X_{Q_j}), thus receives the instantiations of $\sim Q(U_{Q_j}, X_{Q_j})$. When a proof is found, the answer clause will be

$$\{ANSWER(X_{Q_1}\theta) \vee ANSWER(X_{Q_2}\theta) \dots \vee ANSWER(X_{Q_r}\theta)\}.$$

Variables are then substituted for the appropriate Skolem functions to yield

$$\{ANSWER(X_{Q_1}\sigma) \vee ANSWER(X_{Q_2}\sigma) \dots \vee ANSWER(X_{Q_r}\sigma)\}.$$

Let $X_{Q_j} = \{x_{j1}, x_{j2}, \dots, x_{jm}\}$.

Let σ restricted to X_{Q_j} be $\{t_{j1}/x_{j1}, t_{j2}/x_{j2}, \dots, t_{jm}/x_{jm}\}$.

The answer terms printed out by QA3 are

$$[x_{11} = t_{11} \text{ and } x_{12} = t_{12} \dots \text{ and } x_{1m} = t_{1m}] \quad (4)$$

or $[x_{21} = t_{21} \text{ and } x_{22} = t_{22} \dots \text{ and } x_{2m} = t_{2m}]$

\vdots

or $[x_{r1} = t_{r1} \text{ and } x_{r2} = t_{r2} \dots \text{ and } x_{rm} = t_{rm}]$.

According to (3), all the free variables in the set Z that appear in the answer

DEDUCTIVE INFORMATION RETRIEVAL

are universally quantified. Thus any two occurrences of some free variable in two terms must take on the same value in any interpretation of the answer.

In the example given above, whose answer (1) had the single answer term $f(y)$, the complete answer statement is

$$(\forall y)P(y, f(y)).$$

In section 3.3 we present two more examples. The answer in the second example has four answer terms, illustrating the subcase of (4),

$$\begin{aligned} & [x_{11}=t_{11} \text{ and } x_{12}=t_{12}] \\ \text{or} & [x_{21}=t_{21} \text{ and } x_{22}=t_{22}]. \end{aligned}$$

The answer statement proved can sometimes be simplified. For example, consider

$$\begin{aligned} \text{QUESTION: } & (\exists x)P(x) \\ \text{ANSWER: } & \text{YES, } x=a \\ & \text{or } x=b, \end{aligned}$$

meaning that the answer statement proved is

$$[P(a) \vee P(b)].$$

Suppose it is possible to prove $\sim P(b)$ from other axioms. Then a simpler answer is provable, namely

$$\text{ANSWER: YES, } x=a.$$

3. Processes described as a state transformation

In some of the applications of QA3 mentioned in section 5 it is necessary to solve problems of the kind: 'Find a sequence of actions that will achieve some goal.' One method for solving this type of problem is to use the notion of transformations of states. We show here how processes involving changes of state can be described in first-order logic and how this formalism is used. The process of finding the values of existentially quantified variables by theorem-proving can be used to find the sequence of actions necessary to reach a goal.

The basic mechanism is very simple. A first-order logic function corresponds to an action or operator. This function maps states into new states. An axiom takes the following form:

$$P(s_1) \wedge (f(s_1) = s_2) \Rightarrow Q(s_2)$$

where

s_1 is the initial state

$P(s_1)$ is a predicate describing the initial state

$f(s_1)$ is a function (corresponding to an action)

s_2 is the value of the function, the new state

$Q(s_2)$ is a predicate describing the new state.

The equality can be eliminated, giving

$$P(s_1) \Rightarrow Q(f(s_1)).$$

As an example, consider how one might describe the movements of a robot. Each state will correspond to one possible position of the robot. Consider the statement 'If the robot is at point a in some state s_1 , and performs the action of moving from a to b , then the robot will be at position b in some resulting state s_2 .' The axiom is

$$(\forall s_1)(\forall s_2)[AT(a,s_1) \wedge (move(a,b,s_1) = s_2) \Rightarrow AT(b,s_2)].$$

The function $move(a,b,s_1)$ is the action corresponding to moving from a to b . The predicate $AT(a,s_1)$ is true if and only if the robot is at point a in state s_1 . The predicate $AT(b,s_2)$ is true if and only if the robot is at point b in state s_2 .

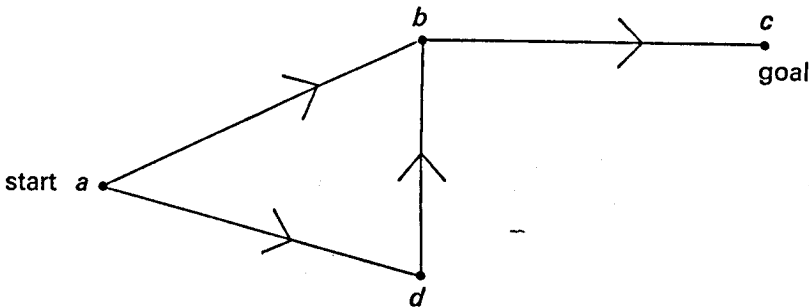


Figure 1

Now consider an example showing how the theorem-prover can be used to find a sequence of actions that reach a goal. The robot starts at position a in initial state s_0 . From a he can move either to b or d . From b he can move to c . From d he can move to b . The allowed moves are shown in figure 1.

The axioms are:

- $A_1. AT(a,s_0)$
- $A_2. (\forall s_1)[AT(a,s_1) \Rightarrow AT(b,move(a,b,s_1))]$
- $A_3. (\forall s_2)[AT(a,s_2) \Rightarrow AT(d,move(a,d,s_2))]$
- $A_4. (\forall s_3)[AT(b,s_3) \Rightarrow AT(c,move(b,c,s_3))]$
- $A_5. (\forall s_4)[AT(d,s_4) \Rightarrow AT(b,move(d,b,s_4))]$

Axiom A_1 states that the robot starts at position a in State s_0 . Axioms A_2 , A_3 , A_4 , and A_5 describe the allowed moves.

We now ask for a sequence of actions that will move the robot to position c . We present this question in the form 'Does there exist a state in which the robot is at position c ?'

QUESTION: $(\exists s)AT(c,s)$

ANSWER: YES, $s = move(b,c,move(a,b,s_0))$

By executing this resulting function $move(b,c,move(a,b,s_0))$ our hypothetical robot could effect the desired sequence of actions. The normal order of

DEDUCTIVE INFORMATION RETRIEVAL

evaluating functions, starting with the innermost and working outward, gives the order of performing the actions: move from a to b and then move from b to c . In general, this technique of function composition can be used to specify sequences of actions.

The proof of the answer by resolution is given below, with comments. The negation of the question is $(\forall s) \sim AT(c,s)$, and the refutation process finds, by instantiation, the value of s that leads to a contradiction. The successive instantiations of s appear as arguments of the special predicate, *ANSWER*. The constants are a, b, c , and s_0 . The free variables are s, s_1, s_2, s_3 , and s_4 .

Proof

- | | |
|--|------------------------|
| 1. $\{ \sim AT(c,s \vee ANSWER(s)) \}$ | Negation of question |
| 2. $\{ \sim AT(b,s_3) \vee AT(c,move(b,c,s_3)) \}$ | Axiom A_4 |
| 3. $\{ \sim AT(b,s_3) \vee ANSWER(move(b,c,s_3)) \}$ | From resolving 1 and 2 |
| 4. $\{ \sim AT(a,s_1) \vee AT(b,move(a,b,s_1)) \}$ | Axiom A_2 |
| 5. $\{ \sim AT(a,s_1) \vee ANSWER(move(b,c,move(a,b,s_1))) \}$ | From resolving 3 and 4 |
| 6. $\{ AT(a,s_0) \}$ | Axiom A_1 |
| 7. $\{ ANSWER(move(b,c,move(a,b,s_0))) \}$ | From resolving 5 and 6 |

Note that the process of proving the theorem corresponds to starting at the goal node c and finding a path back to the initial node a .

Consider a second example. Two players p_1 and p_2 play a game. In some state s_1 , player p_1 is either at position a or position b .

$$B1. \quad AT(p_1,a,s_1) \vee AT(p_1,b,s_1).$$

If in state s_1 , player p_2 can move anywhere.

$$B2. \quad (\forall y) AT(p_2,y,move(p_2,y,s_1))$$

The position of player p_1 is not affected by p_2 's movement.

$$B3. \quad (\forall x)(\forall y)(\forall s)[AT(p_1,x,s) \Rightarrow AT(p_1,x,move(p_2,y,s))]$$

Does there exist some state (sequence) such that p_1 and p_2 are together?

$$\text{QUESTION: } (\exists x)(\exists s)[AT(p_1,x,s) \vee AT(p_2,x,s)]$$

$$\text{ANSWER: } \text{YES, } [x=a \text{ and } s=move(p_2,a,s_1)]$$

or

$$[x=b \text{ and } s=move(p_2,b,s_1)]$$

This answer indicates that two meeting possibilities exist; either (1) player p_1 is at position a and player p_2 moves to a , meeting p_1 at a , or (2) player p_1 is at position b and player p_2 moves to b , meeting p_1 at b . However, the 'or' answer indicates that we do not know which one move will lead to a meeting.

The 'or' answer is due to the fact that Axiom B_1 did not specify player p_1 's position. The answer statement that has been proved is

$$[AT(p_1, a, move(p_2, a, s_1)) \wedge AT(p_2, a, move(p_2, a, s_1))] \\ \vee [AT(p_1, b, move(p_2, b, s_1)) \wedge AT(p_2, b, move(p_2, b, s_1))].$$

Proof

1. $\{ \sim AT(p_1, x, s) \vee \sim AT(p_2, x, s) \vee ANSWER(x, s) \}$ Negation of question
2. $\{ AT(p_2, y, move(p_2, y, s)) \}$ Axiom B_2
3. $\{ \sim AT(p_1, x, move(p_2, x, s_1)) \vee ANSWER(x, move(p_2, x, s_1)) \}$ From 1, 2
4. $\{ \sim AT(p_1, x, s) \vee AT(p_1, x, move(p_2, y, s)) \}$ Axiom B_3
5. $\{ \sim AT(p_1, y, s_1) \vee ANSWER(y, move(p_2, y, s_1)) \}$ From 3, 4
6. $\{ AT(p_1, a, s_1) \vee AT(p_1, b, s_1) \}$ Axiom B_1
7. $\{ AT(p_1, b, s_1) \vee ANSWER(a, move(p_2, a, s_1)) \}$ From 5, 6
8. $\{ ANSWER(a, move(p_2, a, s_1)) \vee ANSWER(b, move(p_2, b, s_1)) \}$ From 5, 7

It is possible to formalize other general problem-solving tasks in first-order logic, so that theorem-proving methods can be used to produce solutions. For a discussion of formalizations of several general concepts including cause, 'can', knowledge, time, and situations, see McCarthy and Hayes (1969).

4. PROGRAM ORGANIZATION

The organization of the question-answering program QA3 differs from that of a 'pure' theorem-proving program in some of the capabilities it emphasizes: a proof strategy intended for the quick answering of easy questions even with a large data base of axioms, a high level of interaction between the user and both the question-answering program and the data base in a suitable command language, and some flexibility in the question-answering process so that the program can be fitted to various applications. In this section we describe the principal features of the system.

1. Program control

The user can control the proof process in several ways.

1. The user can request a search for just a 'yes' answer, instead of both 'yes' and 'no'.
2. The user can request the program to keep trying, by increasing its effort if no proof is found within preset limits. This lets QA3 search for a more difficult proof.
3. When a proof is found it can be printed out. Included with the proof are statistics on the search: the number of clauses generated, the number of clauses subsumed out of the number attempted, the number of successful resolutions out of the number attempted, and the number of successful factors generated out of the number attempted.

4. The user can request that the course of the search be exhibited as it is in progress by printing out each new clause as it is generated or selected from memory, along with specified information about the clause.
5. The user can request that existentially quantified variables in the question be not traced.
6. The user can designate predicates and functions that are to be evaluated by LISP programs. For example, the predicate $1 \leq 2$ might be evaluated by LISP to yield the truth value T . This feature also allows the transfer of control to peripheral devices.
7. Parameters controlling the proof strategy, such as degree and set of support are accessible to the more knowledgeable user.
8. A number of editing facilities on the clauses in memory are useful:
 - (a) A new axiom can be entered into memory,
 - (b) An axiom in memory can be deleted, and
 - (c) The axioms containing any predicate letter can be listed.

2. Special uses of the theorem-prover

'The theorem-prover' refers to a collection of LISP functions used during the theorem-proving process - e.g. RESOLVE, FACTOR, PROVE, PRENEX, CHECKSUBSUMPTION, etc.

The management of the data in memory is aided by the theorem-prover. A statement is stored in memory only if it is neither a tautology nor a contradiction. A new clause is not stored in memory if there already exists in memory another clause of equal length or shorter length that subsumes the new clause. Two other acceptance tests are possible although they are not now implemented. A statement given the system can be checked for consistency with the current data base by attempting to prove the negation of the statement. If the statement is proved inconsistent, it would not be stored. As another possible test, the theorem-prover could attempt to prove a new statement in only 1 or 2 steps. If the proof is sufficiently easy, the new statement could be considered redundant and could be rejected.

The theorem-prover can also be used to simplify the answer, as described in section 3.

3. Strategy

The theorem-proving strategy used in QA3 is similar to the unit-preference strategy, using an extended set-of-support and subsumption.

The principal modification for the purposes of the question-answering system is to have two sets of clauses during an attempted proof. The first set, called 'Memory', contains all the statements (axioms) given the system. The second set, called 'Clausetlist' is the active set of clauses containing only

the axioms being used in the current proof attempt and the new clauses being generated. Clauselist is intended to contain only the clauses most relevant to the question.

There is a high cost, in computer time and space, for each clause actively associated with the theorem-prover. The cost is due to the search time spent when the clause is considered as a candidate for resolution, factoring, or subsumption, and the extra space necessary for book-keeping on the clause. Since most clauses in Memory are irrelevant to the current proof, it is undesirable to have them in Clauselist, unnecessarily consuming this time and space. So the basic strategy is to work only on the clauses in Clauselist, periodically transferring new, possibly relevant clauses from Memory into Clauselist. If a clause that cannot lead to a proof is brought into Clauselist, this clause can generate many unusable clauses. To help avoid this problem the strategy is reluctant to enter a non-unit clause into Clauselist.

The proof strategy of the program is modified frequently, but we shall present an approximate overview of the proof strategy. When a question is asked, Clauselist will initially contain only the negation of the question, which is the set-of-support. A modified unit preference strategy is followed on Clauselist, using a bound on degree. As this strategy is being carried out, clauses from Memory that resolve with clauses in Clauselist are added to Clauselist. This strategy is carried out on Clauselist until no more resolutions are possible for a given degree bound.

Finally, the bound is reached. Clauselist, with all of its book-keeping, is temporarily saved. If the theorem-prover was attempting a 'yes' answer, it now attempts a 'no' answer. If attempting a 'no' answer, it also saves the 'no' Clauselist, and returns a *NO PROOF FOUND* answer. The user may then continue the search requesting *CONTINUE*. If the bound is not reached in either the yes or no case, the *INSUFFICIENT INFORMATION* answer is returned. The strategy has the following refinements:

1. After a newly created unit fails to resolve with any units in Clauselist, it is checked against the units in Memory for a contradiction. This helps to find short proofs quickly.
2. Frequently, in the question-answering applications being studied, a proof consists of a chain of applications of two-clauses, i.e., clauses of length two. Semantically it usually means that set-membership of some element is being found by chaining through successive supersets or subsets. To speed up this process, a special fast section is included that resolves units in Clauselist with two-clauses in Memory. Our experience so far is that this heuristic is worthwhile.
3. Each new clause generated is checked to see if it is subsumed by another shorter clause in Clauselist. All longer clauses in Clauselist are checked to see if they are subsumed by the new clause. The longer subsumed clauses are deleted.

4. Hart's theorem (1965) shows how binary resolution can generate redundant equivalent proofs. Equivalent proofs are eliminated from the unit section. Wos terms this property, 'Singly-connectedness'. Currently this has not yet been implemented for the non-unit section.
5. An extended set-of-support is used, allowing pairs of clauses in Clauselist but not in the set-of-support to resolve with one another up to a level of 2.
6. The sets, Memory and Clauselist, are indexed to facilitate search. The clauses in Memory are indexed by predicate letters and, under each predicate letter, by length. The clauses in Clauselist are indexed by length.

In searching Memory for relevant clauses to add to Clauselist, clauses already in Clauselist are not considered. The clauses of each length are kept on a list, with new clauses being added at the end of the list. Pointers, or place-keepers, are kept for these lists, and are used to prevent reconsidering resolving two clauses and also to prevent generating equivalent proofs.

The strategy is 'complete' in the sense that it will eventually find any proof that exists within the degree and space bound.

5. PERFORMANCE OF QA3

1. Applications

The program has been tested on several question sets used by earlier question-answering programs. In addition, QA3 is now being used in other applications. The subjects for the first question set given QA2, reported in Green and Raphael (1968), consisted of some set-membership, set-inclusion, part-whole relationship and similar problems.

Raphael's SIR (1964b) program gave a similar but larger problem set also having the interesting feature of requiring facts or axioms from several subjects to interact in answering a question. SIR used a different subroutine to answer each type of question, and when a new relation was added to the system, not only was a new subroutine required to deal with that relation but also changes throughout the system were usually necessary to handle the interaction of the new relation with the previous relations. This programming difficulty was the basic obstacle in enlarging SIR. Raphael proposed a 'formalized question-answerer' as the solution. QA3 was tested on the SIR problem set with the following results: in two hours of sitting at the teletype all the facts programmed into or told to SIR were entered into the QA3 memory as axioms of first-order logic and QA3 answered essentially all the questions answered by SIR. The questions skipped used the special SIR heuristic, the 'exception principle'. It was possible to translate, as they were read, questions and facts stated in SIR's restricted English into first-order logic.

Slagle, in his paper on DEDUCOM, a question-answering system (1965), presented a broader, though less interactive, problem set consisting of gathered questions either answered by programs of, or else proposed by, Raphael (1964a), Black (1964), Safier (1963), McCarthy (1963), Cooper (1964), and Simon (1963). Included in this set were several examples of sequential processes, including one of McCarthy's End Game Questions (1963), Safier's Mikado Question (1963), McCarthy's Monkey-and-Bananas Question (1963), and one of Simon's State Description Compiler Questions (1963). Using the technique discussed in section 3.3 to describe processes, it was possible to axiomatize all the facts and answer all the questions printed in Slagle's paper. Furthermore, QA3 overcame some of the defects of DEDUCOM: QA3 could answer all answerable questions, the order of presenting the axioms did not affect its ability to answer questions, and no redundant facts were required. QA3 was then tested on the entire set of twenty-three questions presented in Cooper (1964). QA3 correctly answered all the questions, including four not answered by Cooper's program and sixteen not answered by DEDUCOM.

QA3 also solved the Wolf, Goat, and Cabbage puzzle in which a farmer must transport the wolf, goat, and cabbage across the river in a boat that can hold only himself and one other. The wolf cannot be left alone with the goat and the goat cannot be left alone with the cabbage.

In all of the problems mentioned above, QA3 was given the facts and questions in first-order logic. Raphael's program and Cooper's program used a restricted English input.

Using the English-to-logic translator developed by Coles (1968), Coles and Raphael have begun studying some medical question-answering applications of QA3.

QA3 is being tested in the Stanford Research Institute Automaton (robot) on problem-solving tasks.

2. Limitations

A few limitations should be emphasized. Firstly, QA3 is still not a finished system. One very important feature that is missing is the automatic handling of the equality relation, and this is not a trivial problem. Without an automatic equality capability, QA3 is very awkward on certain problems that are conveniently stated in terms of equality. The equality relation is but one instance of other 'higher-order' concepts (e.g. set theory) that either (i) cannot be described in first-order logic, or (ii) require some meta-level operations such as an axiom schema, or (iii) are awkward and impractical in first-order logic. However, it is not yet clear just what are the practical limitations of a first-order logic system having suitable 'tricks'.

One of the virtues of QA3 is that relatively subject-independent heuristics are used. All subject dependence comes from the particular axioms stored in memory, the theorem being proved, and the particular representation chosen

for each statement. This adds elegance and generality, yet yields a reasonably powerful system. However, for harder problems it may be necessary to be able to add subject-dependent search heuristics, or 'advice' for particular problems. Such an advice-taking capability will require a flexible and easily modifiable search strategy.

The particular heuristics used in QA3 are experimental and have not been thoroughly tested in question-answering applications (although the changes and heuristics added appear to have improved the system). As each modification of the strategy was added, the performance did improve on a particular class of problems. To help remedy some of this uncertainty several measures of performance are now automatically printed out after each question and will be used to evaluate questionable heuristics.

Another qualification is that the questions and subjects investigated were chosen from conjectured test problems or else from test problems used by other question-answering or problem-solving systems. This facilitates comparison, but does not necessarily indicate performance on more practical problems.

The new and more difficult applications being considered might lead to a better understanding of the exact limitations of QA3, or of theorem-proving techniques, for question-answering.

3. Performance

To answer any of the questions mentioned above, QA3 requires from a few seconds to a few minutes. We can roughly measure the problem-solving capacity of QA3 by giving the depth of search allowed and the free space available for storing clauses produced in searching for a proof. The space available for storing clauses produced during a proof typically allows a few hundred clauses to be stored. The depth of search is given by degree bound, normally set at 10. It is interesting to note that the many 'common sense' reasoning problems mentioned herein were within these bounds of QA3, and thus were not difficult proofs, compared to some of the mathematical proofs attempted by theorem-provers.

Acknowledgements

I would like to acknowledge the advice and guidance of Dr Bertram Raphael and Professor John McCarthy, as well as help from Mr. R. Kowalski in correcting an earlier draft of section 3; also the programming and ideas of Robert A. Yates.

The work reported here was supported under Contract AF 30(602)-4147, Rome Air Development Center, and the Advanced Research Projects Agency, Advanced Sensors Group; and also under Contract No. AF 19(628)-5919, Air Force Cambridge Research Laboratories.

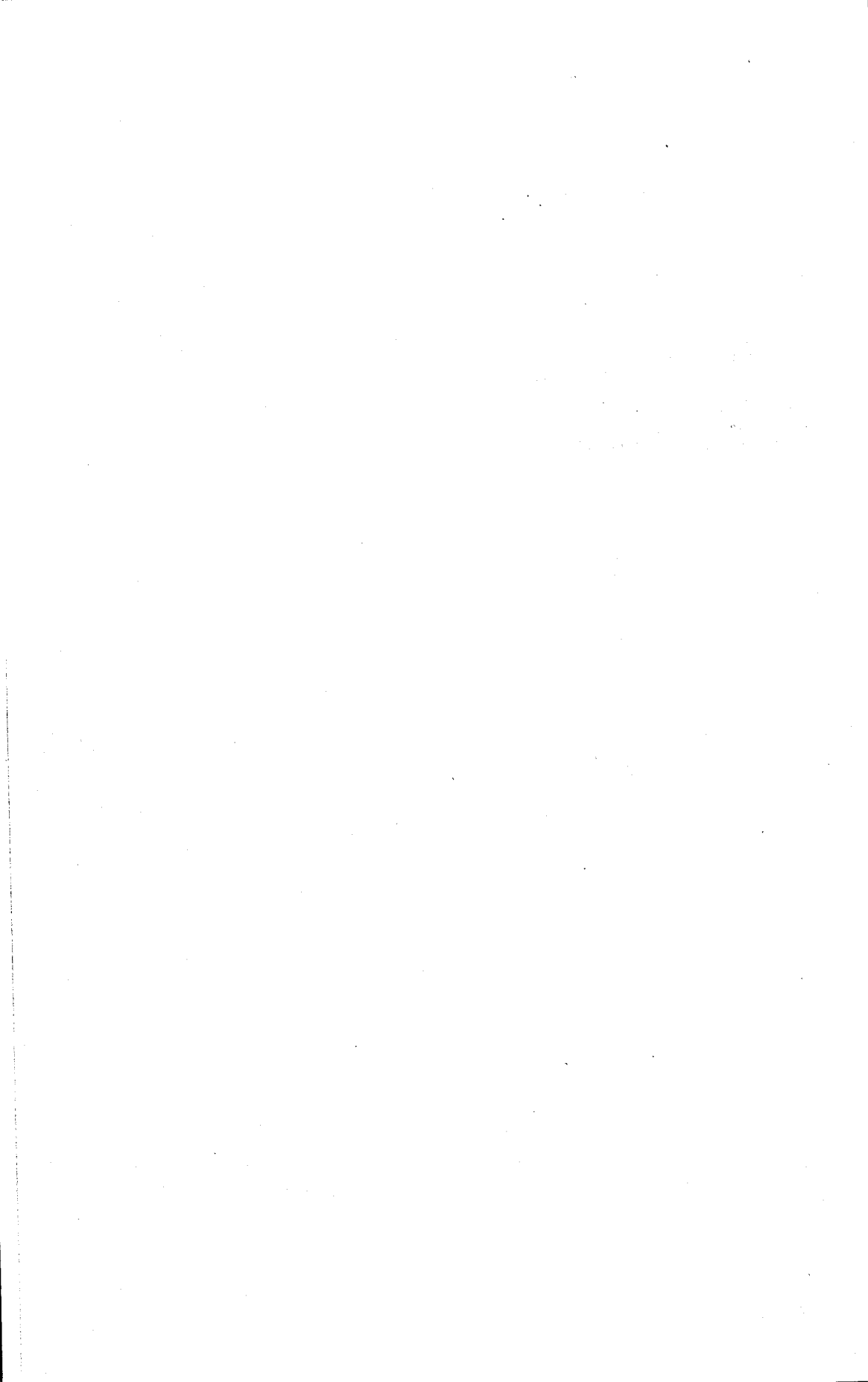
REFERENCES

Black, F.S. (1964) A deductive question-answering system. Ph.D. thesis, Harvard.

- Coles, L. S. (1968) An on-line question-answering system with natural language and pictorial input. *Proceedings 23rd ACM National Conference*.
- Cooper, D. C. (1966) Theorem proving in computers, *Advances in programming and non-numerical computation* (ed. Fox, L.). London: Pergamon Press.
- Cooper, W.S. (1964) Fact retrieval and deductive question answering information retrieval systems, *J. Ass. comput. Mach.*, **11**, 117-37.
- Davis, M. (1963) Eliminating the irrelevant from mechanical proofs. *Annual symposia in applied mathematics XIX*. Providence, Rhode Island: American Mathematical Society.
- Green, B.F., Jr., Wolf, A.K., Chomsky, C. & Laughery, K. (1963) **BASEBALL: an automatic question answerer**. *Computers and thought* (eds Feigenbaum, E.A., & Feldman, J.). New York: McGraw-Hill.
- Green, C.C., & Raphael, B. (1968) The use of theorem-proving techniques in question-answering systems. *Proceedings 23rd ACM National Conference*.
- Hart, T.P. (1965) A useful algebraic property of Robinson's Unification Algorithm. Memo No. 91, AI Project, Project MAC, MIT.
- Lindsay, R.K. (1963) Inferential memory as the basis of machines which understand natural language. *Computers and thought* (eds Feigenbaum, E.A. & Feldman, J.). New York: McGraw-Hill.
- McCarthy, J. (1958) Programs with common sense. *Symposium mechanization of thought processes*. Teddington: Nat. Physical Lab.
- McCarthy, J. (1963) Situations, actions & causal laws. *Stanford artificial intelligence project memorandum*, No. 2.
- McCarthy, J. & Hayes, P. (1969) Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, p. 463-502 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.
- Raphael, B. (1964a) A computer program which 'understands'. *Proc. FJCC* Washington, D.C.: Spartan Books.
- Raphael, B. (1964b) **SIR: a computer program for semantic information retrieval**. MAC-TR2, Project MAC, MIT.
- Robinson, J.A. (1965) A machine-oriented logic based on the resolution principle. *J. Ass. comput. Mach.*, **12**, 23-41.
- Robinson, J.A. (1967) A review of automatic theorem-proving. *Annual symposia in applied mathematics XIX*. Providence, Rhode Island: American Mathematical Society.
- Safier, F. (1963) The Mikado as an advice taker problem. *Stanford artificial intelligence project memorandum*, no. 3.
- Safier, F. (1965) Simple Simon. *Stanford artificial intelligence project memorandum*, no. 35.
- Simon, H. (1963) Experiments with a heuristic compiler, *J. Ass. comput. Mach.*, **10**, 493-50
- Slagle, J.R. (1965) Experiments with a deductive, question-answering program. *Communications of the ACM* **8**, 792-8.
- Wos, L., Carson, D. F. & Robinson, G.A. (1964) The unit preference strategy in theorem proving. *AFIPS* **26**, 615-21, Fall, J.C.C. Washington, D.C.: Spartan Books.
- Wos, L. T., Carson, D.F. & Robinson, G.A. (1965) Efficiency and completeness of the set-of-support strategy in theorem-proving. *J. Ass. comput. Mach.*, **12**, 536-41.
- Wos, L.T., Robinson, G.A., Carson, D.F. & Shalla, L. (1967) The concept of demodulation in theorem-proving. *J. Ass. comput. Mach.*, **14**, 698-709.



**MACHINE LEARNING
AND HEURISTIC
PROGRAMMING**



HEURISTIC DENDRAL : a Program for Generating Explanatory Hypotheses in Organic Chemistry

B. Buchanan

Georgia Sutherland and

E. A. Feigenbaum

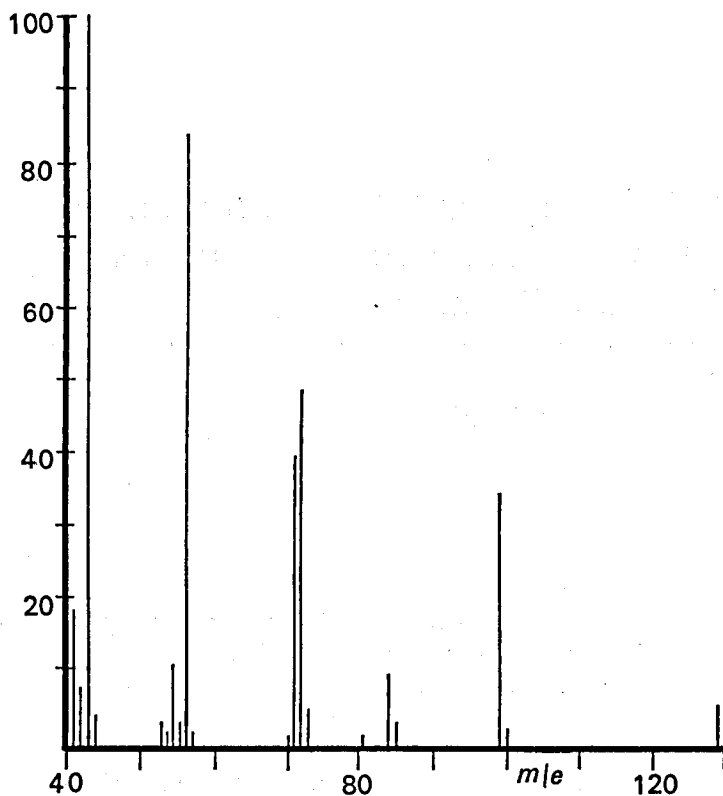
Computer Science Department, Stanford University

A computer program has been written which can formulate hypotheses from a given set of scientific data. The data consist of the mass spectrum and the empirical formula of an organic chemical compound. The hypotheses which are produced describe molecular structures which are plausible explanations of the data. The hypotheses are generated systematically within the program's theory of chemical stability and within limiting constraints which are inferred from the data by heuristic rules. The program excludes hypotheses inconsistent with the data and lists its candidate explanatory hypotheses in order of decreasing plausibility. The computer program is heuristic in that it searches for plausible hypotheses in a small subset of the total hypothesis space according to heuristic rules learned from chemists.

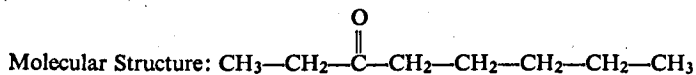
INTRODUCTION

The computer program described below resulted from an interest in studying scientific hypothesis formation as a decision-making process. To make progress on this broad and general problem, it seemed useful to choose a particular scientific task involving inductive behaviour and to explore it in as much detail as possible. The task chosen is in a well defined but relatively new and complex area of organic chemistry, namely the analysis of mass spectra of organic molecules. HEURISTIC DENDRAL is a computer program which generates molecular 'graphs' (i.e., structures) as hypotheses to explain the data produced by a mass spectrometer.

The data produced when a mass spectrometer fragments molecules of a chemical sample can be interpreted as a list of masses of fragments paired



Composition: $C_8H_{16}O$



Computer Representation of the Mass Spectrum:

((41 . 18)(42 . 7)(43 . 100)(44 . 3)
 (53 . 3)(54 . 1)(55 . 11)(56 . 3)
 (57 . 80)(58 . 2)(70 . 1)(71 . 36)
 (72 . 44)(73 . 5)(81 . 1)(85 . 6)
 (86 . 2)(99 . 31)(100 . 2)(128 . 5))

Figure 1. The mass spectrum for 3-OCTANONE

with their relative abundances. An example of a mass spectrum is shown in figure 1. By studying the resulting list of number pairs, chemists can infer the molecular structure of the chemical sample. Some of the decision processes which chemists use in making such inferences are incorporated into a computer program along with a structure-generating algorithm which provides a systematic approach to the problem of deducing the structure of a chemical sample. The computer program is HEURISTIC DENDRAL; and it is now

capable of making inferences from mass spectra to molecular structures in a restricted domain.

The foundation for the **HEURISTIC DENDRAL** program is Lederberg's (1964) **DENDRAL** Algorithm (section 7 contains a summary of this algorithm). The algorithm gives a way of representing and ordering chemical molecules uniquely; thus it gives a method for generating all topologically possible molecules of a given composition without redundancy. It is a systematic and exhaustive topologist which can generate all non-cyclical graphs that can be made with the atoms of the composition, knowing no chemistry other than the valences of these atoms. The **DENDRAL** algorithm defines the hypothesis space in much the same way as a legal move generator for a chess-playing program defines the total move space within which good chess moves will be sought.

The computer program is written in the **LISP** language on the **PDP-6** computer at the Stanford University Artificial Intelligence Laboratory. It occupies approximately eighty thousand words of memory. Working with Professor Joshua Lederberg at Stanford, William Weiher and William White developed the basic representation and wrote the initial program. The program has also benefited greatly from the attention of members of the Stanford Mass Spectrometry Laboratory: Professors Carl Djerassi, Alex Robertson, Jerry Meinwald, and especially Dr Alan Duffield.

The program itself is segmented into five subprograms: the **PRELIMINARY INFERENCE MAKER**, the **DATA ADJUSTOR**, the **STRUCTURE GENERATOR**, the **PREDICTOR**, and the **EVALUATION FUNCTION**. The interrelation of these subprograms is shown in figure 2.

The **PRELIMINARY INFERENCE MAKER** (described in section 1) examines a spectrum and determines what general classes of chemical substructures are confirmed or disconfirmed by the data. All hypothesized structures generated later by **HEURISTIC DENDRAL** must contain all the indicated substructures (all of which are put on a list called **GOODLIST**); and no structure may contain any substructure which is disconfirmed by the spectrum. (All the forbidden substructures are put on a list called **BADLIST**.)

The **DATA ADJUSTOR** subprogram (described in section 2) chooses significant spectral peaks for the **STRUCTURE GENERATOR** to use for its Zero Order Theory. At present there are four independent ways of interpreting the spectrum.

The **STRUCTURE GENERATOR** (see section 3) uses the information deduced by the **PRELIMINARY INFERENCE MAKER** and the **DATA ADJUSTOR** to produce a list of all topologically possible chemical structures which are consistent with the spectrum. The consistency criteria are the lists of 'good' and 'bad' substructures and the Zero-Order Spectral Theory, described in detail in section 3.2.

The **PREDICTOR** subprogram is a rough model of a mass spectrometer (see section 4). It is used to predict significant features of the mass spectrum

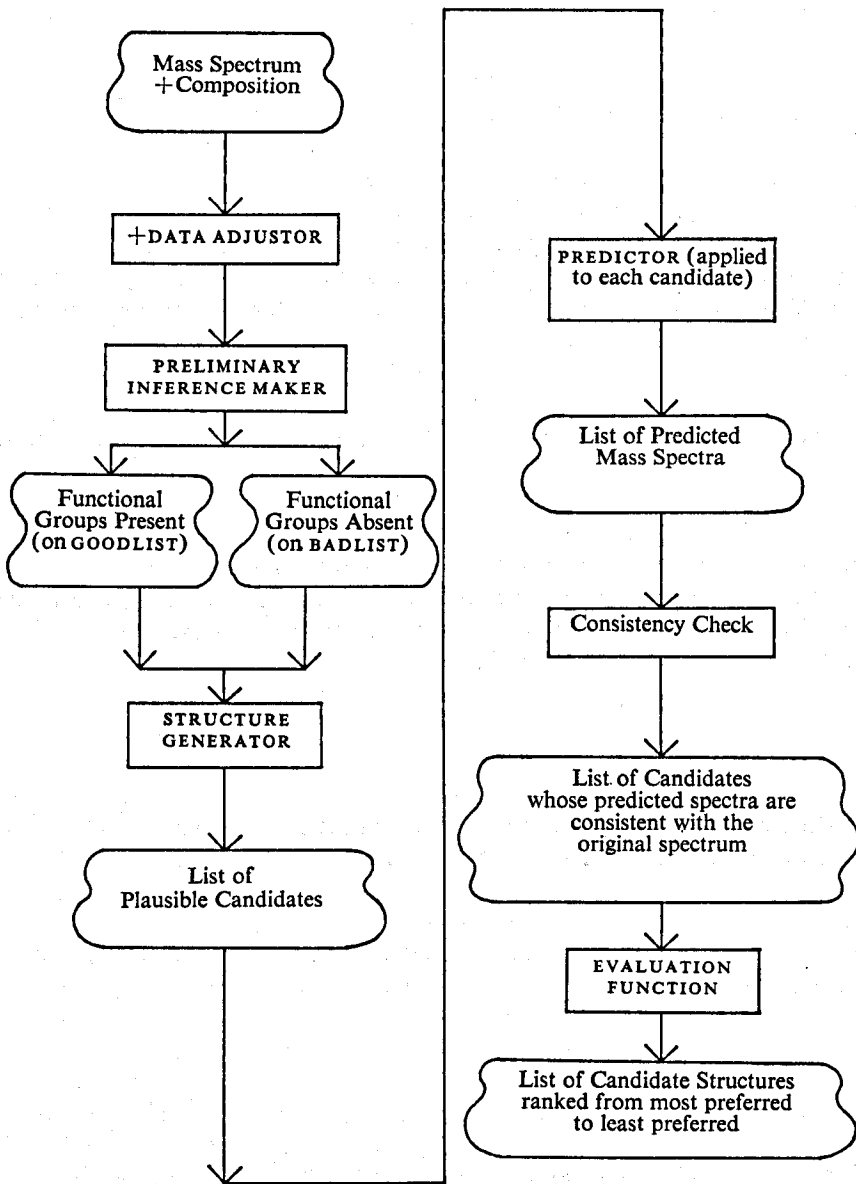


Figure 2. General design of HEURISTIC DENDRAL

corresponding to each candidate structure output by the STRUCTURE GENERATOR.

The EVALUATION FUNCTION (see section 5) compares each predicted spectrum against the original spectral data and assigns a score representing similarity of the two spectra. This enables the candidate hypotheses output by the STRUCTURE GENERATOR to be listed in order of their 'plausibility' or estimated degree of confirmation.

An ideal program for deducing the structure of a chemical sample would output exactly one structure as *the* explanation for the spectral data. Up to now the usual case has been that several different structures are suggested as plausible explanations for the data. However, even a short list is a far better result than was obtained by the original program, which listed all the topologically possible structures and made no use of any real data at all.

Because the constraints which have been included in the program to limit the search space are heuristic, nothing guarantees that the correct structure will not be bypassed. When a test run does fail, however, the program is modified after our chemist-informants study the output and analyze their own decision procedures. The purpose of this report is merely to describe the current state of HEURISTIC DENDRAL and to sketch some of our plans for future program developments.

1. THE PRELIMINARY INFERENCE MAKER

The PRELIMINARY INFERENCE MAKER is conceptually very simple: it looks for the presence or absence of sets of peaks in a mass spectrum and updates GOODLIST or BADLIST, thus constraining HEURISTIC DENDRAL from generating large numbers of molecular structures as possible explanations of a given mass spectrum. By looking for patterns of peaks in the spectrum which are characteristic of some structural fragment, such as the keto group, this preliminary program can tell the STRUCTURE GENERATOR to concentrate on some fragments and to avoid others. It does this by temporarily putting desirable structures on GOODLIST (see section 3.5) and undesirable ones on BADLIST (see Section 3.3).

The program has access to translations of Tables 1 and 2. As Table 1 indicates implicitly, this program knows the name, structure, valence, valence locations, empirical formula, and symmetrical atoms, as well as some characteristic peaks for several functional groups. It also recognizes priorities of groups, as Table 2 indicates. Addition of new information is simplified by a short routine (QUEST) which asks the chemist at the console for the essential information - and explains what it wants if he does not understand. An example is shown in Table 3.

In this example the function QUEST is called to prompt information about identifying a new group, in this case the ester group. The lines preceded by asterisks are messages from the machine. Lines following a machine prompt (i.e., after a colon) were typed in from the console. This dialogue is much

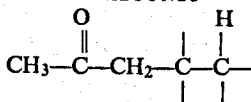
MACHINE LEARNING AND HEURISTIC PROGRAMMING

functional group and characteristic subgraph

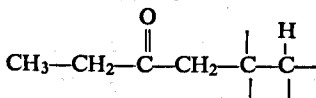
A. KETONE



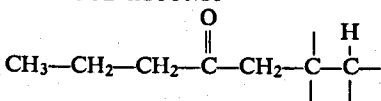
B. METHYL-KETONE₃



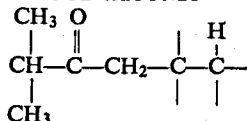
C. ETHYL-KETONE₃



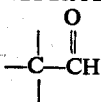
D. N-PROPYL-KETONE₃



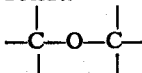
E. ISO-PROPYL-KETONE₃



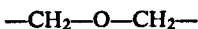
F. ALDEHYDE



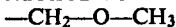
G. ETHER



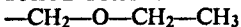
H. ETHER₂



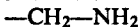
I. METHYL-ETHER₂



J. ETHYL-ETHER₂



K. PRIMARY-AMINE₂



identifying conditions

1. There are 2 peaks at mass units x_1 & x_2 such that

- a. $x_1 + x_2 = M + 28$
- b. $x_1 - 28$ is a high peak
- c. $x_2 - 28$ is a high peak
- d. At least one of x_1 or x_2 is high

1. Ketone conditions are satisfied
2. 43 is a high peak
3. 58 is a high peak
4. $M - 43$ is a low peak
5. $M - 15$ is low or possibly zero

1. Ketone conditions are satisfied
2. 57 is a high peak
3. 72 is a high peak
4. $M - 29$ is a high peak
5. $M - 57$ is a high peak

1. 71 is a high peak
2. 43 is a high peak
3. 86 is a high peak
4. 58 appears with any intensity

1. 71 is a high peak
2. 43 is a high peak
3. 86 is a high peak
4. There is no peak at 58

1. $M - 44$ is a high peak
2. 44 is a high peak

1. $M - 17$ is absent
2. $M - 18$ is absent

1. Ether conditions are satisfied
2. There are 2 peaks at x_1 & x_2 such that
 - a. $x_1 + x_2 = M + 44$
 - b. At least one of x_1 or x_2 is high

1. Ether₂ conditions are satisfied
2. 45 is a high peak
3. $M - 15$ is low or possibly zero
4. $M - 1$ appears (any intensity)

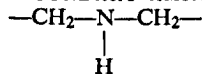
1. Ether₂ conditions are satisfied
2. 59 is a high peak
3. $M - 15$ appears (any intensity)

1. 30 is a high peak
2. No other peak is high

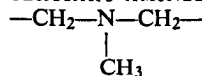
Table 1. Important chemical groups and their identifying conditions

functional group and characteristic subgraph

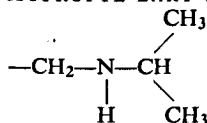
L. SECONDARY-AMINE²



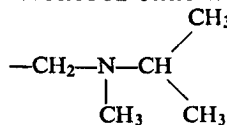
M. TERTIARY-AMINE²



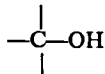
N. ISOPROPYL-2ARY-AMINE²



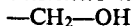
O. ISOPROPYL-3ARY-AMINE²



P. ALCOHOL



Q. PRIMARY-ALCOHOL



R. C-2-ALCOHOL

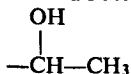


Table 1. (contd.)

identifying conditions

1. There are 2 peaks at x_1 & x_2 such that

- a. $x_1 + x_2 = M + 43$
- b. At least one of x_1 or x_2 is high

2. 30 is a high peak

1. There are 2 peaks at x_1 & x_2 such that

- a. $x_1 + x_2 = M + 71$
- b. At least one of x_1 or x_2 is high

2. 44 is a high peak

1. 44 is a high peak

2. 72 is a high peak

3. $M - 15$ is a high peak

1. 58 is a high peak

2. 86 is a high peak

3. $M - 15$ is a high peak

1. M is low or possibly zero

2. Either $M - 18$ or $M - 17$ appears (any intensity)

3. $M - 46$ appears (any intensity)

1. Alcohol conditions are satisfied

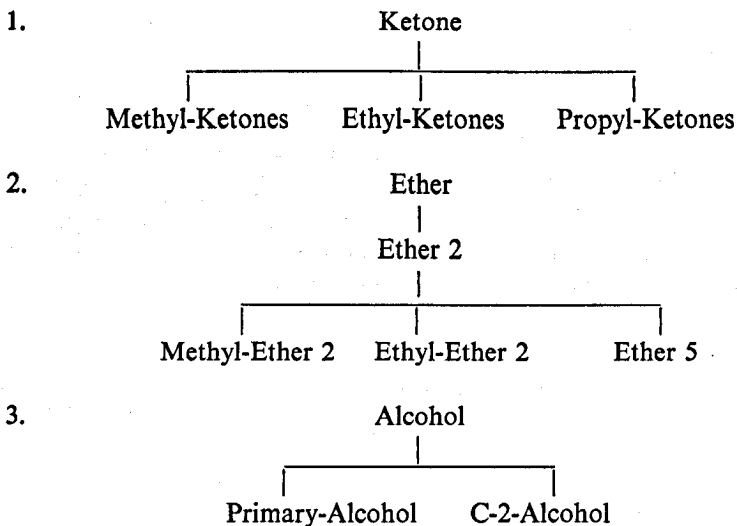
2. The 31 peak is approximately 10%

1. Alcohol conditions are satisfied

2. 45 is a high peak

MACHINE LEARNING AND HEURISTIC PROGRAMMING

A. Family Trees: priorities within families



B. Hierarchies of families

Ketone > Ether

(i.e., check Ethers only if the Ketone tests fail).

Table 2. Family priorities

(QUEST)

* THIS PGM REQUESTS INFORMATION TO ALLOW THE INFERENCE MAKER TO PUT RADICALS ON GOODLIST OR BADLIST ON THE STRENGTH OF THE APPEARANCE OR NON-APPEARANCE OF A FEW SPECTRAL LINES. IF YOU DO NOT KNOW THE PROPER FORM FOR YOUR ANSWER TO ANY REQUEST TYPE A QUESTION MARK. IF YOU MAKE A MISTAKE IN A LINE YOU CAN CORRECT IT WHEN QUEST IS DONE BY CALLING THE FUNCTION "CHGPROPS" OF ONE ARGUMENT, THE GROUP NAME. E.G., (CHGPROPS (QUOTE KETONE))

* NAME OF FUNCTIONAL GROUP OR RADICAL:
?

* ANY ATOMIC NAME WILL SUFFICE:
ESTER

* PEAKS WHOSE ABSENCE INDICATES THE ABSENCE OF THIS GROUP:

?* THE PGM WANTS A LIST OF DOTTED PAIRS INDICATING MASS-INTENSITY PAIRS IT SHOULD LOOK FOR. MASS UNITS MAY BE SPECIFIED AS

1. A NUMBER,
2. M (THE MOLECULAR WT), OR
3. A LIST OF THREE ELEMENTS:
 - 3.1 THE LETTER M .
 - 3.2 SLASH AND MINUS (OR PLUS) SIGN
 - 3.3 A NUMBER (TO BE SUBTRACTED OR ADDED TO THE M).

Table 3. Conveying information to the PRELIMINARY INFERENCE MAKER

BUCHANAN, SUTHERLAND AND FEIGENBAUM

INTENSITY UNITS MAY BE SPECIFIED AS

1. A NUMBER BETWEEN 0 AND 100 INCLUSIVE,
2. THE WORD "ANY" (ANY INTENSITY ABOVE 0),
3. THE WORD "LOW" (INTEN BETWEEN 0 AND 5 INCL),
4. THE WORD "HIGH" (INTEN BTW 11 AND 100 INCL), OR
5. THE WORD "POSS0" (INTEN LOW OR ZERO)

FOR EXAMPLE THIS LIST WOULD BE ACCEPTABLE:

((45 . HIGH) ((M / -45) . 0) (M . LOW))

*PEAKS:

((45 . 0) (60 . 0))

*PEAKS SUFFICIENT TO INDICATE THIS GROUP:

?*TYPE "SAME" IF THE NECESSARY

CONDITIONS ARE ALSO SUFFICIENT, "NIL" IF THERE ARE

NO SUFF CONDITNS, OR "??" IF YOU WANT THE

CONDITION LIST EXPLAINED:

SAME

*STRUCTURE IN LIST NOTATION: ? *TYPE A LIST WHOSE FIRST ELMT IS 1 AND INDICATE ALL HYDROGEN ATOMS. E.G., (1 C (1 H) (1 H) (1 H)) FOR THE METHYL RADICAL.

*STRUCTURE: (1 C(2 0) (1 0(1 C)))

*VALENCE: ? *IF ALL FREE BONDS ARE ON ONE ATOM, TYPE THIS NUMBER. OTHERWISE TYPE A LIST OF NUMBERS INDICATING HOW THE FREE BONDS ARE SPLIT UP, READING THE STRUCTURE YOU TYPED FROM LEFT TO RIGHT, IGNORING ATOMS WITH NO FREE VALENCE: (1 2)

*LIST OF SYMMETRIES: ?

*TYPE A LIST OF THE FORM

((3 2 1)) TO INDICATE THAT THE FIRST & THIRD ATOMS WITH FREE VALENCE ARE SYMMETRICAL:()

*PLACE OF GROUP IN ITS FAMILY: ? *TYPE A LIST CONSISTING OF (1) THE NAME OF THE NEXT HIGHER FAMILY MEMBER OR NIL

(2) THIS GROUP NAME

(3), (4), . . . (N) NAMES OF ALL NEXT

LOWER MEMBERS. E.G., FOR ETHER2:

(ETHER ETHER2 METHYL-ETHER2 ETHYL-ETHER2) *PLACE OF GROUP IN ITS FAMILY: NIL

*THANKS CALL AGAIN

(CHGROPS (QUOTE ESTER))

*PROPERTY TO CHANGE: ?

*PROPERTY NAME *DESCRIPTION

(TYPE ONE)

NESS LIST OF PEAKS WHOSE ABSENCE INDICATES ABSENCE OF GROUP

SUFF LIST OF PEAKS INDICATING THE PRESENCE OF THIS GROUP

FORM EMPIRICAL FORMULA AS A LIST OF DOTTED PAIRS

VALENCE A SINGLE NUMBER OR A VECTOR

STRUCT STRUCTURE IN LIST NOTATION

SYM LIST OF SYMMETRIES

*PROPERTY: VALENCE

*VALUE: (1 3)

*REPLACE (R) OR ADD (A)? R

(1 3)

*PROPERTY TO CHANGE: NIL

DONE

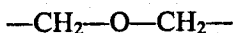
Table 3. (contd.)

shorter when the user knows the correct form for his response (and does not type a question mark). If the user calls QUEST1 instead of QUEST, the machine begins with the first prompt, bypassing the initial descriptive sentences. If the user wishes to change any information typed in previously, he calls the function CHGPROPS.

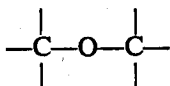
The PRELIMINARY INFERENCE MAKER is given as input the spectrum, the empirical formula of the molecule, and a noise threshold to apply to the spectrum.¹ The LISP function INFER accepts this information and controls the subsequent inferences about the presence or absence of the structural groups. The program performs the following three tests for each structure:

1. Is the empirical formula of the structure compatible with the empirical formula of the molecule? If not, get the next structure.
2. Is any necessary condition falsified by the spectrum? If so, put this structure on BADLIST and get the next structure.
3. Are all sufficient conditions satisfied by the data? If so, put this structure on GOODLIST and get the next structure. Note: at present all sets of conditions are both necessary and sufficient.

The Family Trees shown in Table 2 reduce the effort of the PRELIMINARY INFERENCE MAKER and eliminate redundant effort in the STRUCTURE GENERATOR. When the spectral data indicate that a group is absent from the structure (resulting in the addition of this group to BADLIST), no lower members of the same family are even checked. On the other hand, if both a higher and a lower member of a family are indicated by the data (resulting in the addition of both groups to GOODLIST), only the lower, more specific, group is used. For example, if both of the subgraphs named ETHER and ETHER2 are on GOODLIST, the program deletes the more general one, ETHER, since ETHER2 constrains structure generation more; that is, there are fewer isomers of a given composition containing



than there are which contain

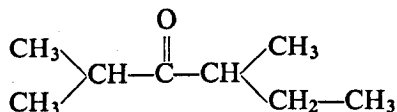


The family hierarchy list also reduces the effort of this program. If *any* member of the first family is on GOODLIST, no members of the second family are even checked.

In the cases of the general Ketone, Ether2, Secondary-Amine2, and Tertiary-Amine2 subgraphs, the preliminary inference maker can, in fact, isolate the position of the functional group as well as determine which

¹ Spectral peaks are deleted if their amplitudes are lower than the threshold. This option has not yet been exercised since it may confuse the inference maker. A threshold value of 1 bypasses this option.

functional group is present. It can do this because of highly favourable alpha-cleavage (cleavage of the bond between the carbon atom attached to the heteroatom and the rest of the molecule) which is an identifying condition for each of these subgraphs. For example, in the ketone shown below, the program can tell that the keto radical (C=O) is between some C₃H₇ structure and some C₄H₉ structure, even though it cannot specify terminal radicals uniquely.



This positional information is passed to the STRUCTURE GENERATOR's partitioning routine which is discussed in section 3.4. The effect in this case is that the only ketones which will be generated are those with the keto group bounded by three carbon atoms on one side and four carbon atoms on the other.

```

(INFER (QUOTE C8H16O) s:09046 1)
*GOODLIST>(*ETHYL-KETONE3*)
*BADLIST>(*C-2-ALCOHOL* *PRIMARY-ALCOHOL* *ETHYL-ETHER2* *METHYL-
ETHER2* *ETHER2* *ALDEHYDE* *ALCOHOL* *ISO-PROPYL-KETONE3* *N-PROPYL-
KETONE3* *METHYL-KETONE3*)

```

```

-----
(JULY-4-1968 VERSION)
C2*ETHYL-KETONE3*H8
MOLECULES NO DOUBLE BOND EQUIVS
1. CH2.. CH2.C3H7 C=O C2H5,
2. CH2.. CH..CH3 C2H5 C=O C2H5,
3. CH2.. CH2.CH..CH3 CH3 C=O C2H5,

```

DONE

s:09046

```

((41.. 18.) (42.. 7.) (43.. 100.) (44.. 3.) (53.. 3.) (54.. 1.)
(55.. 11.) (56.. 3.) (57.. 80.) (58.. 2.) (70.. 1.) (71.. 36.)
(72.. 44.) (73.. 5.) (81.. 1.) (85.. 6.) (86.. 2.) (99.. 31.)
(100.. 2.) (128.. 5.))

```

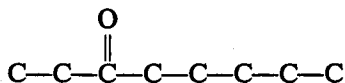
Table 4. Example from the PRELIMINARY INFERENCE MAKER

Although the chemical heuristics used in this program are more like suggestions than rules, they have demonstrated their usefulness in a number of trials. The results of one of these trials appear in Table 4. The dashed line separates the lines printed by the PRELIMINARY INFERENCE MAKER from the lines printed by the STRUCTURE GENERATOR. The complete output for this example is discussed in detail in section 6. In this case, total output is reduced from 698 isomers¹ to 3 isomers as a result of applying the PRELIMINARY INFERENCE MAKER.

¹The number of chemically stable acyclic structures with empirical formula C₈H₁₆O is 698; the total number of topologically possible graphs which satisfy just the valence restrictions is 790. Section 3 discusses the program which generates these structures.

MACHINE LEARNING AND HEURISTIC PROGRAMMING

The program was given the mass spectrum and empirical formula of 3-octanone. S:09046 is the mass spectrum for the structure:



The first output structure is the correct structure for this spectrum. The rest of the output structures are other ethyl-ketones, because of GOODLIST.

2. THE DATA ADJUSTOR

The DATA ADJUSTOR subprogram determines which mass points of a real spectrum are significant enough to be used by later programs. This process is separable into three steps:

1. Determine the mass of the molecular ion (M). If this number is not in the real spectrum, insert it with large amplitude.
2. Delete peaks at impossible mass points. Specifically, delete peaks at 1, 2, . . . , 10, 11, 19, . . . , 23, $M-1$, $M-2$, . . . , $M-23$.
3. Delete all but the most significant peaks. Significance has to be decided without knowledge of the molecular structure of the sample producing the spectrum. Four methods of determining significance are included at present, with the choice of method being left to the program user.
 - (i) The Threshold Method selects those mass points which have amplitudes higher than a certain number.
 - (ii) The Biemann Method selects the two mass numbers with highest amplitudes in each interval of 14 mass numbers.
 - (iii) The Lederberg Method selects the n mass numbers with highest amplitudes. The number n depends upon the number of atoms in the chemical composition.

$$n = \frac{12(\text{count} - 1)}{5} - 1$$

- (iv) The fourth method allows the user to specify the number of mass points to be used (the n highest peaks).

Each of these four methods reduces the real spectrum to a set of mass numbers judged to be the 'significant peaks' in the data. This revised spectrum is then given to the STRUCTURE GENERATOR which treats it as the data to guide the process of generating structures.

The data-adjusting routine is invoked by calling the function REALSPEC with three arguments. The first two arguments specify the composition and the spectrum. The third argument indicates which method to use. The four possibilities for the third argument are:

- (i) $-m$ - use threshold m
- (ii) T - use Bieman's method
- (iii) NIL - use Lederberg's method
- (iv) n - take the n highest peaks.

The relative merit of these methods has not been determined. In the examples processed so far, it appears that the STRUCTURE GENERATOR needs only a few of the mass points in a typical spectrum.

3. THE STRUCTURE GENERATOR

The DENDRAL Algorithm described in section 7 is a procedure for generating all of the topologically possible acyclic structures (isomers) of a chemical composition. This algorithm is based on a canonical notation for chemical structures and an ordering procedure which determines which of two canonical structures is 'higher'.

The STRUCTURE GENERATOR is a computer program implementing the DENDRAL algorithm but with the inclusion of heuristic constraints to prevent the program from generating structures which are incompatible with chemical theory or mass spectral data. Applying these constraints in the course of structure generation greatly increases the efficiency of the program, decreasing amount of output and total run time by several orders of magnitude.

The STRUCTURE GENERATOR is designed to solve the following problem:

GIVEN: a list of defined atoms with their valences and weights

a composition (empirical formula)

a spectrum (mass numbers only)

a list of likely substructures

a list of impossible substructures

TASK: generate all structures compatible with the given data. If there are no data-oriented lists of likely or impossible substructures and no spectral data, the program generates all structural variants (isomers) of the given composition.

INSTRUCTIONS:

1. Make certain that the composition is compatible with the spectrum, if there is one.
2. Consider only those structures which have exactly the types and amounts of atoms specified by the composition.
3. If certain substructures are required, remove their atoms from the composition and insert the name of a 'superatom' to represent that substructure. Be sure the superatom substructure is compatible with the spectrum. After generating a structure containing superatoms, translate superatoms into the original substructures before printing the output.
4. If the partitioning option is to be exercised, consider all subgroupings (partitions) of the composition. Determine whether a given partition is 'plausible'. Generate only those structures which come from plausible partitions.
5. Generate substructures to combine into isomers. The isomers must contain no forbidden substructures; and each substructure must be compatible with the spectrum, if there is one.

6. Provide the user of the program periodic opportunities to observe and change the direction of structure generation. (Optional)
7. Remember past work. (Optional)

The mechanisms for following these instructions are described in the following sections.

3.1. Brief description of the structure-generating algorithm

The basic steps for generating chemical structures are to generate radicals (structures with a free bond) and to connect radicals to make larger structures. Radicals are generated recursively from a composition list of atoms by deciding on the first atom (apical node) and free bond (afferent link) and then making one or more radicals out of the remaining composition. The function `GENRAD`¹ constructs a single radical by this method; `MAKERADS` constructs two, three, or four radicals from a single composition; and `GENMOL` determines the center of a molecular structure and causes two or more radicals to be constructed to attach to the center.

The function `UPRAD` takes a radical and returns the next higher radical which can be made from the same elements. `UPMOL` does the same for molecules. The function `ISOMERS` causes all the structures for a given composition to be generated and printed in ascending canonical order.

The program's constraints are controlled by a number of switches (global variables) which are pre-set before calling `ISOMERS`. The switches are named: `SPECTRUM`, `GOODLIST`, `BADLIST`, `NOPARTS`, `DIALOG`, `DICTSWITCH`, and `OUTCONTROL`. Individual constraints may be bypassed at the discretion of the user of the program. When all constraints are turned off, the `STRUCTURE GENERATOR` becomes a routine graph maker, generating an exhaustive list of all possible acyclic graph structures of n nodes, where different nodes may have different numbers of links (valence). The switch settings for unconstrained program operation are:

```
(SETQ SPECTRUM NIL)
(SETQ GOODLIST NIL)
(SETQ BADLIST NIL)
(SETQ NOPARTS T)
(SETQ DIALOG (QUOTE OFF))
(SETQ DICTSWITCH (QUOTE OFF))
(SETQ OUTCONTROL (QUOTE OFF))
```

¹ The LISP functions which perform certain operations will be identified in this report. To simplify the discussion, however, their arguments and operation will not be discussed. A separate paper lists all LISP functions contained in the `STRUCTURE GENERATOR` and outlines their use.

3.2. The SPECTRUM and the Zero-Order Theory of Mass Spectrometry

The SPECTRUM of the STRUCTURE GENERATOR is a single list of numbers, corresponding to significant mass numbers in the real spectrum. The DATA ADJUSTOR sub-program provides the STRUCTURE GENERATOR with this list of numbers, all of which have equal importance as far as the STRUCTURE GENERATOR is concerned.

The SPECTRUM is consulted to confirm the presence of compositions and radicals. The first reference to SPECTRUM is by the function ISOMERS which must make certain that the mass of the input composition is present. If it is not, then no structures can be generated for that composition. Any smaller composition can be made into structures if it is not inconsistent with the Zero-Order Theory described below. This constrains the program to consider only those sub-compositions which have some promise of leading to structures compatible with the SPECTRUM.

The Zero-Order Theory assumes that every bond of a structure to which it applies will break (one bond at a time) and that at least one of each pair of substructures obtained from a single break will contribute its mass to the spectrum. The Zero-Order Theory does not apply to double bonds, triple bonds, or bonds leading to certain small structures. That is, in order for a structure to be consistent with the Zero-Order Theory, at least one of the following conditions must be met;

1. The structure contains exactly one atom other than hydrogen.
2. The afferent link is greater than 1.
3. The mass of the structure is less than 30.
4. The mass of the structure is in the SPECTRUM list.
5. The complement mass of the structure is in the SPECTRUM list.

This Zero-Order Theory of Mass Spectrometry is crude but easily implemented. A more elaborate spectral theory is contained in the PREDICTOR (section 4), but obtaining such a spectrum for an arbitrary structure consumes more computer time than would be practical in a program such as the STRUCTURE GENERATOR. The Zero-Order Theory is sufficient to limit the output of the STRUCTURE GENERATOR to a small class of hypotheses, but it will need major revisions before it can be classed as a 'smart' limiting heuristic.

To make use of spectral information in the STRUCTURE GENERATOR it is merely necessary to execute (SETQ SPECTRUM L) where L is a list of integers, corresponding to the desired mass numbers. To terminate use of spectral information, execute (SETQ SPECTRUM NIL).

When structure generation is proceeding in the presence of a SPECTRUM, the work that is remembered for future reference (see section 3.7 describing the dictionary) is independent of the spectral data, so it is permissible to use several different spectra in succession in the same program core image.

3.3. Preventing the generation of forbidden substructures

Some chemical structures are so implausible (unstable) that they would never exist, either alone or imbedded within any larger structure. The STRUCTURE GENERATOR has a list (BADLIST) of these implausible structures; and no output of the STRUCTURE GENERATOR will contain any substructure on BADLIST.¹

The STRUCTURE GENERATOR avoids generating structures containing forbidden substructures by checking rigorously before attaching new atoms to a piece of structure. At every step in generating a radical, the program knows the partially built structure and can determine whether the atom and bond which are about to be attached to it will include one of the forbidden substructures. The following process insures that no forbidden structures will be formed:

1. The partial structure is guaranteed to be plausible because of previous checking.
2. Form the new partial structure by adding the next bond and atom.
3. Consider all elements of BADLIST which have a top atom identical to the atom just added to the partial structure.
4. For each such element of BADLIST, compare the radicals which are attached to the top atom of the new structure with the radicals attached to the top atom of the BADLIST structure.
5. If every radical on the BADLIST structure is found in the list of radicals on the new structure, then the new structure must be rejected.
6. Rejecting a structure means that it is necessary to change either the added bond or the additional atom (or both) in order to generate an allowable structure.

Note that this process prevents the STRUCTURE GENERATOR from creating many implausible molecules, since the addition of each new node causes a check to be made for forbidden substructures including that node. Usually only part of the structure has been generated because unallocated atoms are added only to stable pieces.

Each forbidden substructure appears on BADLIST several times, once for each possible top (apical) node. Structures are added to or removed from BADLIST by the function FIXBADLIST, which first generates all the forms of the forbidden substructure, and then adds to or deletes from BADLIST according to the desire of the user. Naturally if there are no structures on BADLIST then there are no constraints on the output of implausible structures.

¹ As described in section 1, BADLIST may be expanded according to given spectral data. The permanent part of BADLIST belongs to the program's theory of chemical instability. But substructures of both the theoretical and the context dependent parts of BADLIST are treated alike.

The current form of BADLIST has been suggested after several iterations of the following loop:

suggest forbidden substructures.
 generate output.
 inspect output.

The currently forbidden substructures are listed in Table 5. Hydrogen atoms must be specified explicitly, and lists of atoms enclosed in parentheses indicate that any member of the list may be used in that position on the substructure.

1. $C=C-(N,O)-H$
2. $C\equiv C-(N,O)-H$
3. $N=N-(N,O,H)$
4. $H-O-C-(N,O)-H$
5. $H-C-N=O$
6. $N=C-O-H$
7. $O-O$
8. $(N,O)-(N,O)-(N,O)$
9. $O-S$
10. $S-S-S$
11. $H-N-C-N$

$\begin{array}{l} /H \\ /H \\ \backslash ANY \end{array}$
12. $(N,O)-C-O-H$

$\begin{array}{c} || \\ O \end{array}$

Table 5. Forbidden substructures comprising BADLIST

3.4. Partitioning a composition into plausible sub-compositions

The unconstrained structure-generating algorithm produces molecules by first determining the center of the structure (bond or particular atom) and then generating all possible radicals out of the remaining composition and attaching them to the center in all possible combinations. When all possible centers have been considered, the process of structure generation is complete.

The task of generating a set of n radicals from a single composition requires that the composition be divided (partitioned) into n subcompositions. Then a radical is generated from each smaller composition.

All possible partitions are considered in the unconstrained program, regardless of whether the sub-compositions are 'plausible' or compatible with a spectrum. The lowest partition is considered first, where 'lowest' means that it has two sub-compositions, of equal size if possible, and with the lowest ranked atoms all in one of the compositions (where the arbitrary ranking

from carbon to sulfur is: $C < N < O < P < S$). After the lowest partition has been used, it is incremented to the next higher form, used to make radicals, and incremented again until the highest set of compositions has been used.

As each partition is generated it can be checked for plausibility before any attempt is made to generate the corresponding radicals. Each sub-composition is checked against the spectrum. (See section 3.2.) If its weight is not present, the whole partition can be bypassed. Similarly, each sub-composition of a partition can be checked against the dictionary of previous work (see section 3.7) to determine whether any radicals can be made from the sub-composition. If the dictionary indicates an impossible composition, then the whole partition can be bypassed.

The advantages of these constraints are evident in even a simple case. Suppose we wish to partition the composition $((U.1)(C.6)(O.6))$ into two parts. The 'lowest' partition is $[((U.0)(C.6)), ((U.1)(O.6))]$. There are 17 radicals corresponding to $((U.0)(C.6))$ but there are no allowable radicals corresponding to $((U.1)(O.6))$. Thus, the work done in generating all the six-carbon radicals is wasted because there are no six-oxygen radicals to go along with them. If this is determined in advance, then much time will be saved by eliminating this partition. Similarly, this partition might have been eliminated if there were no spectral evidence of a C_6H_{13} fragment (a saturated radical with six carbons). Currently, the only spectral evidence that the program accepts for a composition is a peak at the corresponding mass. This drawback will soon be eliminated, for we are programming the PRELIMINARY INFERENCE MAKER to look for evidence of a more subtle kind. For example, a cluster of peaks may indicate a significant fragmentation although no peak alone indicates it.

Every composition of a partition may satisfy the spectral and dictionary constraints, yet the partition may be implausible when considered as a whole. Plausibility criteria, suggested by chemists, include such considerations as the ratio of carbon to non-carbon atoms in each composition compared to the ratio of carbon and non-carbon atoms in the whole partition. Spectral considerations may be included in the plausibility criteria at a later date. A partition plausibility score is calculated, and if this score does not lie within a given range, the partition is bypassed. The usual lower limit of plausibility scores is $LLIM=4$ (0 is the lowest) and the usual upper limit is $ULIM=10$ (the highest) but these two global variables can be reset by the program user, i.e. (SETQ LLIM 0).

The LISP function MAKELISTCLS (make-a-list-of-composition-lists) is the usual procedure for making a n -part partition out of a composition list. The LISP function MAKEGOODLISTCLS is called instead to insure that the partition satisfies the dictionary, spectrum, and plausibility constraints.

The partitions are usually generated one at a time as needed by the program. But the program can be made to generate all the plausible partitions in

advance and put them on a list (PARTLIST) for future reference by the program. The program assumes that the partitions obtained from PARTLIST are plausible and it makes no further checks. The main advantage in using PARTLIST is that the list can be filtered or re-ordered, either by an arbitrary scheme of the user or on the basis of plausibility scores. Then the 'most interesting' structures will be generated first. At the present time the PARTLIST is only constructed for the top level of molecule-building, although it would be possible to generate partition lists for deeper levels of structure generation.

The partition constraints are activated by the program user as follows:

1. To bypass partitions on the basis of their plausibility scores, either set LLIM>0 or set ULIM<10. If ULIM=10 and LLIM=0, all partitions will be plausible.
2. To activate the partition list, first set the switch NOPARTS=NIL. Then, before generating any molecules, the program will ask the user if he wishes the partition list to be constructed.

3.5. Specifying required substructures

The basic components used by the STRUCTURE GENERATOR are chemical atoms which possess two properties, valence and atomic weight. These atoms are connected by bonds to form radicals (structures with a free bond) which may in turn be connected with other atoms and radicals to form larger radicals and molecules.

The STRUCTURE GENERATOR can also treat complex structural fragments as atoms. Structures so treated have come to be known as 'superatoms'. The STRUCTURE GENERATOR replaces a group of atoms in the given composition by the name of a corresponding superatom, and generates structures with the revised composition (including the superatom name). Only at output time (if then) do the constituent atoms of the superatoms re-appear. Two obvious benefits arise in the use of superatoms:

1. The generation of isomers of a composition is faster because there are fewer atoms in the composition.
2. Structural fragments essential to an explanation of a mass spectrum may be made into superatoms. All isomers will contain the selected substructure, thus the output list is more relevant to the data.

A third benefit was realized as a result of the use of superatoms:

3. Ring structures can now be generated by the previously acyclic STRUCTURE GENERATOR by specifying each different ring as a superatom.

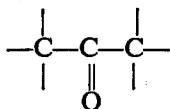
Normal atoms are known to the STRUCTURE GENERATOR because their names are on a global list called ORDERLIST. The usual value of ORDERLIST is '(C N O P S)' and the position on this list defines the 'DENDRAL Order' of the atom: C<N<O<P<S. Each atom on ORDERLIST has the properties

VALENCE and WEIGHT. When superatoms are created, their names are also added to ORDERLIST. Superatoms have the properties VALENCE, WEIGHT, STRUCT and SYM (the last two to be described later).

Any new atom or superatom may be introduced to the STRUCTURE GENERATOR by calling the LISP function (ADDATOM (QUOTE X)) where X is the name¹ of the atom. A common superatom is the keto radical *CO*,

>C=O , with two free bonds on the carbon atom. This superatom has properties VALENCE=2 and WEIGHT=28 and STRUCT=(1 C(2 O)). It is list notation representation² that is stored under the property STRUCT of the superatom name. It is used at output time to put the generated isomer back into canonical DENDRAL notation in terms of ordinary atoms. Note that the free bond leading into the superatom is a 1 rather than a 2, since the most saturated form is always used.

But consider the case of the general ketone substructure, *KET*:



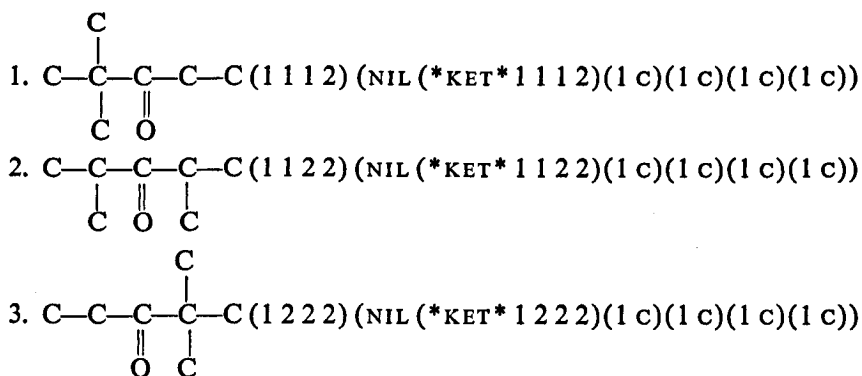
It is desirable to be able to treat this as a superatom, yet there must be some way of stating that the valence of 6 is split up between two different atoms. This is done by stating that VALENCE=(3 3). The property WEIGHT has value 52 and STRUCT=(1 C(1 C(1 C)(2 O))). The numbers in the valence list correspond to atoms with free valences in the list structure, reading from left to right. The first atom which does not have all its valences filled by bonds has an effective valence equal to the first number in the valence list, and so on.

During normal structure generation, the effective valence of this superatom is six, and as many as six radicals may be generated to attach to a ketone superatom. The actual locations to which the radicals are attached are indicated by a locant vector associated with the superatom name in the list notation for a larger structure. The locant vector is a list of numbers, one for each attached radical. Each number specifies the atom to which the radical is attached. The ketone superatom has atoms 1 and 2 which are available for attached radicals. (An atom's number corresponds to its position from left to right in the valence list and the list notation of the structure.) Since each available atom in the ketone structure has three free valences, each atom (1 and 2) may be listed up to three times in the locant vector.

As an example, suppose four methyl groups (radicals of one carbon atom, three hydrogen atoms each) are to be attached to the ketone superatom, represented by '*KET*'. The possibilities, with associated locant vectors and list notation representation, are:

¹ Any atom with a name longer than 1 character should be enclosed in asterisks.

² List notation is analogous to dot notation described in section 7. The difference is that bonds are represented by numbers and radicals are enclosed in parentheses.



Note that 1's in the locant vector are listed before 2's, when the two representations would be equal. For example, in this case

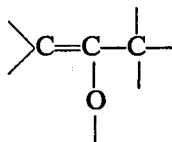
$$(1\ 1\ 1\ 2) = (1\ 1\ 2\ 1) = (1\ 2\ 1\ 1) = (2\ 1\ 1\ 1).$$

A firm rule with locant vectors is that the 'lowest' of equivalent forms is always used.

Further, note that structures 1 and 3 are mirror images, obtained by interchanging nodes 1 and 2. This arises because the ketone superatom is symmetric for these nodes and because the radicals attached to the symmetric nodes are equal. Thus it becomes necessary to check for symmetrical structures to avoid redundancy.

An important property for each superatom with split valences (i.e., the valence property is a list rather than a single number) is the list of symmetries. Each symmetry is a list of numbers, the same length as the valence list, indicating which node is equivalent to the n th node under a given transformation. The identity transformation gives the symmetry (1 2) (node 1 equivalent to node 1, node 2 equivalent to node 2). The identity symmetry is possessed by every split valence superatom and thus is not included in the symmetry list. The reflection transformation applied to the ketone superatom gives the symmetry (2 1) (node 2 equivalent to node 1, node 1 equivalent to node 2). This is the only non-trivial symmetry, so the SYM property for ketone is ((2 1)).

In the case of a superatom such as



the properties would be

STRUCT=(1 c(2 c(1 c)(1 o)))

VALENCE=(2 3 1)

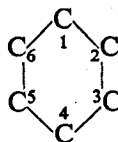
WEIGHT=52

SYM=()

MACHINE LEARNING AND HEURISTIC PROGRAMMING

In the case of the six membered carbon ring shown below the properties would be:

VALENCE=(2 2 2 2 2 2)
 WEIGHT=72
 SYM=((2 3 4 5 6 1)(3 4 5 6 1 2)
 (4 5 6 1 2 3)(5 6 1 2 3 4)
 (6 1 2 3 4 5)(6 5 4 3 2 1)
 (5 4 3 2 1 6)(4 3 2 1 6 5)
 (3 2 1 6 5 4)(2 1 6 5 4 3)
 (1 6 5 4 3 2)).



All possible rotations and reflections produce equivalent structures, so all symmetries are listed.

A ringed structure cannot be written in list notation conveniently, so the composition list is given as the value of STRUCT, for rings, STRUCT=((U. 1)(C. 6)) in this example. (At least one degree of unsaturation is always present in a ring.) Rings are not converted back to basic atoms at output time since there is no convenient way to write them on a single line. The locant vector notation is retained during output of structures containing rings. The first number in the locant vector refers to the attachment position of the afferent link if there is one.

The presence of a superatom name on ORDERLIST defines the superatom for the system but does not force the STRUCTURE GENERATOR to use it. Another global list called GOODLIST, is set by the program user to indicate which superatoms are relevant at a certain time. An element of GOODLIST has the form (NAME MAX MIN), where MAX and MIN specify the upper and lower limits on the number of these superatoms which may be substituted into a given chemical composition. For example, if MAX=0, this superatom is ignored, and if MIN=1, then this superatom is required. If the MAX and MIN are left unspecified, then the assumed values are MAX=100 and MIN=0.

When the STRUCTURE GENERATOR is given a composition from which to generate structures it must first check GOODLIST to see which superatoms can be formed from the given composition. If any superatom has a specified minimum greater than 0, its atoms are removed from the composition and the corresponding number of superatoms are inserted instead. If any superatom with a minimum greater than 0 cannot be made from the atoms of the composition, then all structure generation terminates for that composition. If the Zero-Order Theory is operating, structure generation also terminates if the mass of the superatom is not in the spectrum.

When the required superatoms have been inserted into the composition, the STRUCTURE GENERATOR then places all possible combinations of remaining superatoms into the composition and generates all possible structures from the new set of corresponding compositions.

Consider an example:

GOODLIST=((*COOH*)(*CO*)(*NOH*)(*KET*))

COOH: STRUCT=(1 C(1 O)(2 O))

VALENCE=1

WEIGHT=45

CO: STRUCT=(1 C(2 O))

VALENCE=2

WEIGHT=28

NOH: STRUCT=(1 N(1 O))

VALENCE=2

WEIGHT=31

KET: STRUCT=(1 C(1 C(1 C)(2 O)))

VALENCE=(3 3)

WEIGHT=52

SYM=((2 1))

The composition $C_3H_7NO_2$ has the composition list ((U.1)(C.3)(N.1)(O.2)). The composition lists for the four superatoms are:

((U.1)(C.1)(O.2))

((U.1)(C.1)(O.1))

((U.0)(N.1)(O.1))

((U.1)(C.3)(O.1))

The following revised composition lists are all possible ways of generating isomers of $C_3H_7NO_2$.

1. ((U.0)(C.2)(N.1)(*COOH*.1))
2. ((U.0)(C.2)(*CO*.1)(*NOH*.1))
3. ((U.0)(C.2)(N.1)(O.1)(*CO*.1))
4. ((U.1)(C.3)(O.1)(*NOH*.1))
5. ((U.0)(*NOH*.1)(*KET*.1))
6. ((U.0)(N.1)(O.1)(*KET*.1))
7. ((U.1)(C.3)(N.1)(O.2))

There are two global variables which may be set to limit the total number of superatoms in any single composition. These are MXSAT and MNSAT, representing the maximum and minimum number, respectively. In the example above, if MXSAT=1 then #2 and #5 would not be permitted; or if MNSAT=1 then #7 would not be permitted. If GOODLIST had the form

((*COOH* 2 1)(*CO*)(*NOH*)(*KET*))

then only #1 would be permitted.

When GOODLIST is present during structure generation, the program does not allow implicit superatoms to be formed from the remaining normal atoms. The structure (1 C(1 O)(2 O)), for example, is a forbidden substructure when the explicit superatom *COOH* is on GOODLIST. Thus, if isomers

were generated for all the seven composition lists given above, the total number of isomers would be exactly the number generated by the program for the composition $C_3H_7NO_2$ with no GOODLIST present. For that reason, the variables specifying the maximum and minimum for total and individual superatoms are nearly always specified to prevent uninteresting isomers from being generated. If a GOODLIST superatom happens to contain one of the forbidden substructures on BADLIST, BADLIST prevails. That is, the BADLIST theory of chemical instability must be changed before conflicting superatoms will be generated.

The superatom type of structure generation is activated whenever GOODLIST is not NIL. A general form of GOODLIST is stored in the program and can be used by executing (SETQ GOODLIST SAVEGOODLIST). A function called STRUCTURES will ask the user to specify the maximum and minimum number for each superatom on GOODLIST. As described earlier, the PRELIMINARY INFERENCE MAKER gives superatoms to the STRUCTURE GENERATOR automatically so that generated isomers will correspond as closely as possible to the information contained in the real spectrum.

3.6. Observing the process of structure generation

The basic structure-generating function is a LISP function called GENRAD which generates a single radical from a composition list. GENRAD operates recursively by calling itself for successively smaller subsets of the composition list.

It sometimes happens that the work done in generating radicals from a composition subset is wasted because the partial structure to which these radicals will be attached is implausible. Or else, a user of the program may be uninterested in certain classes of structures, but has to watch impatiently while the program works its way down to the interesting structures.

A dialog option allows the on-line user of the program to inspect GENRAD at each level and to decide whether GENRAD should be allowed to proceed to a deeper level of recursion. At each dialog point, the current partial structure and remaining composition are printed. This gives the user the opportunity to decide if the structure-generating algorithm is proceeding along a fruitful path. If not, a sort of user-implemented tree pruning can be evoked just by answering 'N' to the program's query about whether to continue. It is hoped that the program can learn to make use of the reasons for altering its operation in this way.

The dialog option is initiated by executing (SETQ DIALOG (QUOTE ON)) and it is terminated by executing (SETQ DIALOG (QUOTE OFF)). The dialog may be stopped during structure generation by typing a left arrow instead of the usual answer to questions typed by the program. Then type (SETQ DIALOG (QUOTE OFF)) and the dialog will no longer appear.

The user has another device available to cause the program to terminate structure generation even though the output list is incomplete. To control

output in this way, execute (SETQ OUTCONTROL (QUOTE ON)). The program will pause after a specified number of outputs have been printed and ask permission to continue. The interval for pausing is determined by the number stored as the value of the global variable called AMT.

Both these devices for controlling output are usually ignored, but have been very useful during program debugging and demonstrations.

3.7. Rote memory

The normal operation of the STRUCTURE GENERATOR causes a dictionary to be built which contains all the structural isomers of every composition (and every subset of every composition) which has been encountered. This dictionary contains lists of radicals, saved under names which can be reconstructed from the compositions. Whenever structure generation is under way, the program first searches the dictionary to see if the current composition has been encountered previously. If a dictionary entry exists for a composition, it is assumed to be an exhaustive list of all radicals which can be made from the composition. No further structure generation is performed; the dictionary list provides the output.

Dictionaries which are built during a run of the program may be saved on tape for further use. The function SAVDICT writes a dictionary on tape. The dictionary is recalled for further use by the function GETDICT. The global variable called DICTLIST has as its value a list of all names of entries in the current dictionary.

Because every dictionary entry is assumed by the program to be an exhaustive list of radicals corresponding to the named composition, the dictionary may be used as a program constraint. An existing dictionary (either in core or on tape) may be edited manually (or by computer, using techniques which will be described in a later report). The editing may either delete or add radicals, and subsequent structure generation with the edited dictionary present will result in reduced or expanded output lists.

Sometimes a previously edited dictionary is used unintentionally. For example, BADLIST prevents certain structures from entering the dictionary at all. If BADLIST is later changed, but a previously built dictionary is left in the program, the output will appear as though the old form of BADLIST were still present; or worse, it may appear that the old BADLIST is present for small structures (ones that were previously in the dictionary) but the new BADLIST is present for large structures (ones that are being built for the first time). A certain amount of caution in changing BADLIST will prevent this type of erroneous structure generation.

The dictionary-building process may be turned off so that previous work is not remembered. The command for this is (SETQ DICTSWITCH (QUOTE OFF)), but seldom is it advisable to do this. It speeds initial work, but later work that depends on previous work is slower. A core dictionary may be deleted by executing (UNDICT DICTLIST) and (SETQ DICTLIST NIL). To

restart the dictionary building process after it has been turned off, execute (SETQ DICTSWITCH (QUOTE ON)).

4. THE PREDICTOR: FIRST STEPS TOWARD A COMPUTER THEORY OF MASS SPECTROMETRY

Introduction

Part of HEURISTIC DENDRAL is a computer program which predicts major features of mass spectra of acyclic organic molecules. The program contains a rough theory of mass spectrometry which is still in its formative stages.

In the course of designing the HEURISTIC DENDRAL program for formulating hypotheses to explain mass spectral data, it became apparent that the program needed a detailed theory of mass spectral fragmentation processes. This is because the STRUCTURE GENERATOR suggests plausible candidate structures for explaining the data, but has no way of testing its candidates. Thus, a theory by which the computer could make some verifiable predictions about each candidate would help to reduce the set of likely candidates. Through the program described here, the prediction now takes the form of a suggested mass spectrum for each candidate structure. The EVALUATION FUNCTION described in section 5 then compares the predicted spectrum and original spectrum to determine which structural candidate is the most satisfactory choice.

A mass spectrometer is, briefly, an instrument into which is put a small sample of some chemical compound and out of which comes data representable as a bar graph. The instrument itself bombards molecules of the compound with electrons, producing ions of different masses in varying amounts. The *x*-points of the bar graph represent the masses of ions produced,¹ and the *y*-points represent the relative abundances of ions of these masses. The spectrum predictor program is an attempt to codify the numerous descriptions of what happens inside the instrument, and thus to generate mass spectra in the absence of both the instrument and the actual sample of the substance. The input to the program is a string of characters representing the graph structure of a molecule. The output is a bar graph representing the predicted mass spectrum for this molecule.

In broadest outline, the mass spectrum predictor calculates a spectrum (list of mass-intensity pairs) for a molecule in the following series of steps:

1. Calculate the mass of the molecular ion and an associated intensity, depending on the degree of unsaturation.
2. Determine the nature and extent of eliminations and rearrangements of the molecular ion.
3. Break a bond between a pair of adjacent atoms in the molecule, looking at each bond only once.

¹ More accurately, the *x*-points of a mass spectrum represent the mass to charge ratio (m/e), where most, but not all, recorded fragments are singly charged.

4. Calculate the masses of the two resulting fragments. Then calculate an intensity (on an absolute scale) for each fragment ion by estimating the probability that this bond will break and the probability associated with ionization of each fragment.
5. Determine the nature and extent of eliminations and rearrangements of each fragment ion.
6. Add isotope peaks and peaks at $m + 1$, $m + 2$, to account for hydrogen addition to some fragment ions. (Optional.)
7. Recycle through 3–6 until every bond in the molecule has been considered once.
8. Eliminate low mass peaks and adjust the intensities to percent of the highest peak.

The following discussion elucidates the theory by explaining in detail what the program does in each of the above steps. Justification for some decisions exists in print. Many decisions, however, have been made out of considerations of simplicity, elegance, deference to the intuitions of professionals, or out of ignorance.

4.1. The Program

The molecule is represented in a slight variant of Lederberg's DENDRAL notation described in section 4.4. This notation omits explicit mention of hydrogen atoms but shows all the other connections of a chemical graph. The program attaches a unique name to each atom, and keeps track of each atom's place in the graph by putting names of adjacent atoms on the property list of each atom under indicators FROM and TO.

The molecular ion's mass is the sum of the masses of all atoms plus the masses of all implicit hydrogen atoms. The intensity associated with this mass is a function of the degree of unsaturation of the molecule. This function is easily changed, but it is currently x times the product of all bond orders (on an absolute scale) where $x = 2$ [LISP function called MOLVAL].¹ For example, in a molecule with all single bonds except for one double bond and one triple bond, the intensity of the molecular ion would be

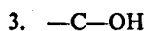
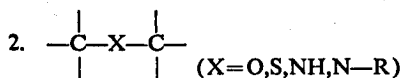
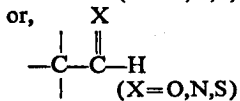
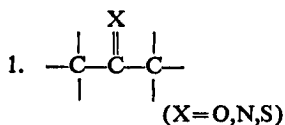
$$2(1 \times 1 \times \dots \times 1 \times 2 \times 3) = 12.$$

Peak heights on this absolute scale are translated onto a 0–100 scale at the end.

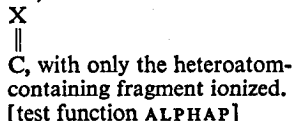
Since any ion, including the molecular ion, tends to a more stable form if possible, the program must take account of the relative stability of each ion, as compared to the stability of possible products it may form. The program looks for ways of losing certain neutral molecules or rearranging the atoms already present. It has a list of very stable ion products, which are preferred structures, and must determine whether, and to what extent, the given molecular ion can form one of these products. To do this, it looks for any occurrence

¹ The most important LISP functions and parameters will be bracketed throughout this section.

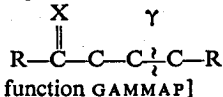
MACHINE LEARNING AND HEURISTIC PROGRAMMING



- a. Only two types of bonds break:
 (i) Bonds alpha to the heteroatom;
 i.e., the bonds between C and



- (ii) Bonds gamma to the heteroatom;
 i.e.,



- b. The McLafferty rearrangement, if possible, is invoked for the molecular ion.
 c. Carbon monoxide is lost from each α -cleavage fragment containing the carbonyl radical.

- a. Only two types of bonds break:
 (i) α -bonds, as above. Preference is given to loss of the most highly substituted fragment or to loss of the longest carbon chain (when degrees of substitution are equal).

(ii) γ -bonds, as above.

- b. In each resulting fragment, subsequent rearrangements favor loss of highly substituted carbons and loss of long carbon chains.

- a. Reduce the intensity of the molecular ion to zero.
 b. Add peaks at the following x, y points (intensities on absolute scale, $a=2$)

<i>mass</i>	<i>intensity</i>
$M-18$	$2a (=4)$
$M-18-15$	$4a (=8)$
$M-18-28$	$8a (=16)$
$M-18-29$	$3/4 \cdot 8a (=12)$
$M-18-42$	$8a/2 (=8)$
$M-18-43$	$3/4 \cdot 8a/2 (=6)$
$M-18-56$	$8a/4 (=4)$
$M-18-57$	$3/4 \cdot 8a/4 (=3)$

until $(M-18-X) < 29$.

[The value of a is controlled by the parameter AFACT.]

- c. Add α -cleavage peaks (with only the hydroxyl fragment ionized). Preference is given to loss of the most highly substituted fragment or to loss of the longest carbon chain.

Table 6. Significant radicals and their effects

type	characteristic subgraph	allocation of intensity units of the molecular ion and [parameter names]*		skeleton of daughter ion
		K1% for parent	K2% for daughter	
McLafferty Rearrangement	$ \begin{array}{c} \text{X} \\ \\ \text{C} \\ / \quad \backslash \\ \text{Z} \quad \text{X-X-X} \\ \text{H} \\ \\ \text{H} \end{array} $ <p>(X=C, N, O, P, S) (Z=C, H, N, O, P, S)</p>	40 [KMCOLD]	60 [KMCNEW]	$ \begin{array}{c} \text{XH} \\ \\ \text{C} \\ / \quad \backslash \\ \text{Z} \quad \text{X} \end{array} $
1, 2† elimination of H ₂ O (thermal)	$ \begin{array}{c} \text{H} \quad \text{OH} \\ \quad \\ -\text{C}-\text{C}- \\ \quad \end{array} $	60 [KWOLD]	40 [KWNEW]	$ \begin{array}{c} -\text{C}=\text{C}- \\ \quad \end{array} $
1, 3† elimination of H ₂ S	$ \begin{array}{c} \text{H} \\ \\ -\text{C}-\text{C}-\text{C}- \\ \quad \quad \\ \quad \quad \quad \text{SH} \end{array} $	0 [KS3OLD]	200 [KS3NEW]	‡
1, 4† elimination of H ₂ S	$ \begin{array}{c} \text{H} \\ \\ -\text{C}-\text{C}-\text{C}-\text{C}- \\ \quad \quad \quad \\ \quad \quad \quad \quad \quad \text{SH} \end{array} $	0 [KS4OLD]	300 [KS4NEW]	‡
1, 3† elimination of HCl	$ \begin{array}{c} \text{H} \\ \\ -\text{C}-\text{C}-\text{C}- \\ \quad \quad \\ \quad \quad \quad \text{Cl} \end{array} $	0 [KCL3OLD]	200 [KCL3NEW]	‡
1, 4† elimination of HCl	$ \begin{array}{c} \text{H} \\ \\ -\text{C}-\text{C}-\text{C}-\text{C}- \\ \quad \quad \quad \\ \quad \quad \quad \quad \quad \text{Cl} \end{array} $	0 [KCL4OLD]	300 [KCL4NEW]	‡

* Global parameters' current values are listed; parameters may be reset by the user.

† The number notation *l, n* refers to the relative positions of the combining atoms or radicals.

‡ The program now calculates only a mass-intensity pair of these daughter ions since we are uncertain about their structures.

Table 7. Rearrangements and eliminations in molecular ions

of the significant radicals listed in Table 6, for example the carbonyl radical >C=O . Associated with each of these significant radicals is a set of rules for restructuring the ion to make it more stable and a set of parameters for determining the extent to which this restructuring should (or does) occur. Heuristic programmers recognize such a plan as a list of situation-action rules of the form: in situation *X* perform action *Y*. A very desirable feature of this is that the list can be extended or amended very easily.

Table 7 is a summary of the program's rules for eliminating neutral molecules or rearranging atoms in molecular ions. The characteristic subgraph is the part of the ion which the program looks for. The two parameters determine the percent of the abundance of the molecular ion (as originally determined) which should be allocated to the original molecular ion ($K1\%$) and to the daughter ion, that is, to the ion after restructuring ($K2\%$). Resetting the parameter whose name is bracketed changes the corresponding allocation. The skeleton of the daughter ion is indicated. The program has rules for removing atoms, changing the order of bonds, and moving atoms from place to place, so that the structure of rearrangement products will be printed, if requested.

After the program has considered the molecular ion [function PARENT] it considers the likelihood that the molecular ion will fragment at each of the bonds between atoms. Its theory says that only single bonds will break apart, thus it skips over double and triple bonds in the molecule. Of the single bonds, it distinguishes bonds between carbon atoms from bonds between a carbon and a non-carbon atom. The probability that the ion fragments at a given bond depends upon the environment of the bond and the functional groups present in the molecule. The probability associated with the ionization of one or the other of the resulting fragments also depends on these features. This part of the program is also organized as a set of conditional sentences: if the molecule contains functional group X and this bond environment has feature b then include the factor resulting from calculation $f(b)$ in figuring the probability that the molecule fragments at this bond. The features which the program considers are explained in detail in section 4.2, together with the associated functions. Briefly, however, the program first checks for the presence of the significant radicals listed in Table 6 and then looks at some or all of the following eleven features to determine both the probability that any particular bond will break and the probability of ionization for each resulting fragment:

1. The order of the bond itself [HTVAL].¹
2. The types of atoms joined by the bond [HTVALCC, HTVALCX, INDUCTIVE].
3. The orders of the bonds which are one atom removed from this bond [VINYLIC].
4. The number of non-hydrogen atoms which are one atom removed from this bond, i.e., the degree of substitution on the first atom in each fragment [LCALC, METHYLP].
5. The number of heteroatoms which are one atom removed from this bond [CONTIGHET].
6. The types of heteroatoms which are one atom removed from this bond [HETERO].

¹ The bracketed names of the LISP functions responsible for these features are given for later reference.

7. The types of radicals that are one atom removed from this bond [CARBONYL].
8. The orders of the bonds which are two atoms removed from this bond [ALLYLIC].
9. The length of the carbon chain in each fragment [CHAINLTH].
10. The total number of heteroatoms in each fragment [NHET].
11. The total number of carbon atoms in each fragment [NUMCARBS].

The result of calculating the probability of ionization of a fragment is a number pair, (x,y) . The first component is the mass of the fragment. The second is the relative abundance of these fragment ions ($y \geq 0$).

Since fragment ions, as well as the molecular ion, may eliminate neutral molecules or rearrange atoms to form more stable ions, the program must be able to predict the most significant occurrences. After the program calculates the mass of a fragment and the relative frequency of its ionization, it checks the fragment for elimination and rearrangement possibilities. Exactly the same procedure is used as for the molecular ion, but the list of characteristic subgraphs may be different depending on the functional groups present in the molecule. Table 8 lists the different possibilities now in the program.

Thus the program examines each bond to calculate the probability of cleavage and each fragment to calculate both the probability of ionization and the possibility of rearrangements. In addition, it has already calculated a molecular ion peak and has looked for the possibility of eliminations and rearrangements in the molecular ion. By the time it has finished, it should have calculated a list of mass-intensity pairs corresponding to the most significant peaks in the actual mass spectrum for the same molecule. To conform to common practice, mass units below 29 are deleted and the intensities are converted to percent of the highest peak (base peak).

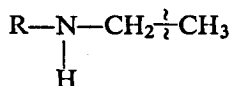
Some annotated examples of predicted spectra appear in section 4.3 together with the actual mass spectra for the same molecules. Section 4.4 explains how to use the program; and section 4.5 explains the options that are available from the console.

4.2. Rules for calculating relative intensities of primary fragments

Cleavage of Single Bonds between Carbon Atoms. Under special conditions the program bypasses the general rule for calculating intensities of fragments given below. Thus, before stating the general rule, which is relatively complex, the exceptions will be noted.

A. *Exceptions*

1. Assign zero as the intensity of the two fragments when considering the bond between CH_2 and CH_3 ; that is, do not break off a methyl which is part of an ethyl. This rule is preempted by the special rules of Table 6 for significant radicals. For example, the methyl radical *will* be lost in this molecule



because this is α -cleavage in an amine.

[test function METHYLP].

2. Assign zero as the intensity of both fragments when considering a vinylic bond; that is, do not break bonds which are adjacent to double bonds. The rules of Table 6 preempt this rule also.

[function VINYLIC].

3. Assign zero as the intensity of all primary fragments of ketones, aldehydes, amines, ethers, thioethers, and alcohols except for the fragments resulting from the rules of Table 6.

4. For α -cleavage in amines, ethers, and thioethers, calculate the intensity of the heteroatom-containing fragment as a function of

(a) the degree of substitution of the first carbon atom of that fragment, and

(b) the number of carbon atoms lost.

Specifically, the intensity is the sum of two factors X_1 and X_2 where $X_1=0$ if 3 or 2 hydrogens are attached to the first carbon,

30 if 1 hydrogen is attached, or

45 if 0 hydrogens are attached.

[function AMINESUBST]

$X_2=3$ if 2 or 1 carbon atoms are lost,

10 if 3 are lost, or

15 if 4 or more are lost.

[function AMINECARBS].

B. The General Rule

The general rule for calculating the intensity of each fragment resulting from dissolution of the bond between two carbon atoms is

$$I = (Z_1 + Z_2 + Z_3 + W_1) \times W_2 \times W_3 \times W_4$$

The Z -factors are context-dependent factors. That is, it is necessary to look at features of both fragments (the total context) in order to calculate each Z -factor. The W -factors are context independent, which is to say that each one can be calculated by looking only at the fragment under immediate consideration.

1. Z_1 is calculated in two steps according to the number of non-hydrogen atoms alpha to the bond under consideration:

(a) Compute a factor (T_1) which is equal to the sum of intensities of both fragments (estimated probability that this bond will break given this information):

$T_1 = 10$ if there are 0, 1, or 2 branches to non-hydrogen atoms,

12 if . . . 3 branches . . .,

16 if . . . 4 branches . . .,

18 if . . . 5 branches . . .,

20 if . . . 6 branches . . . [parameter list TINLTH]

(b) Compute a ratio for splitting T_1 between the two fragments (relative probability that each fragment will be ionized as a result of this break given this information):

ratio = 1:1 if the difference between the number of non-hydrogen branches is 0,

5:7 if the difference . . . is 1,

1:3 if the difference . . . is 2,

1:5 if the difference . . . is 3. [parameter list RINLTH]

The ratio is weighed in favor of the fragment with more alpha branches. Z1 for each fragment is then the result of applying this ratio to T1.

[function LCALC]

2. Z2 is calculated in a similar two-step manner, this time taking into account the number of heteroatoms (non-hydrogen, non-carbon atoms) alpha to the bond under consideration:

(a) Compute a factor (T2) which is equal to the sum of intensities of both fragments (estimated probability that this bond will break given this information):

T2 = 0 if there are 0 branches to non-hydrogen, non-carbon atoms

(heteroatoms) from both of the carbon atoms,

3 if . . . 1 branch . . .,

10 if . . . 2 branches . . .,

20 if . . . 3 branches . . .,

30 if . . . 4 branches . . .,

40 if . . . 5 branches . . .,

50 if . . . 6 branches . . . [parameter list TINCON]

(b) Compute a ratio for splitting T2 between the two fragments (relative probability that each fragment will be ionized as a result of this break given this information):

ratio = 1:1 if the difference between the number of branches to

heteroatoms is 0,

3:10 if the difference . . . is 1,

1:9 if the difference . . . is 2,

1:19 if the difference . . . is 3. [parameter list RINCON]

(again, weighted in favor of the fragment with the more branches).

Z2 for each fragment is then the result of applying this ratio to T2.

[function CONTIGHET]

3. Z3 is an attempt to integrate the principle that longer carbon chains are lost preferentially to smaller ones. The longer a carbon chain in a fragment, the higher the probability that the molecule will split apart at that bond. Also, though, the long-chain fragment is less likely to be ionized than the other fragment at this break-point. So Z3 is calculated for fragment #1 at a break-point as a function of the chain length of fragment #2.

[function CHAINLTH]

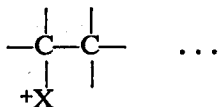
Currently the function just multiplies the chain length by two [the value of CHFACT] although this parameter, like every other in the program, can be easily changed.

The next factors in the intensity calculation for any fragment are context-independent. The program considers only features within the fragment, first on one side of the bond under consideration, then on the other.

4. $W1$ is equal to the number of heteroatoms in the fragment. The program now simply counts the number of occurrences of non-hydrogen, non-carbon atoms, although it could return some function of the count.

[function NHET]

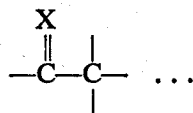
5. $W2$ attempts to capture the principle that in a fragment the types of heteroatoms alpha to the bond under consideration greatly influence the probability that the fragment retains the charge (is ionized) when this bond is broken. That is, the program looks at atoms in the place occupied by X in the following schema and assigns $W2$ by the accompanying rule:



$W2 = 5$ [FHETN] if X is Nitrogen,
 4 [FHETS] if X is Sulfur,
 3 [FHETO] if X is Oxygen,
 2 [FHETCL] if X is Chlorine,
 1 otherwise.

[function HETERO]

6. $W3$ is a similar factor taking account of certain heteroatoms doubly bonded to the carbon at the break-point. The program looks at atoms in the X -place in the schema and assigns $W3$ by the following rule:

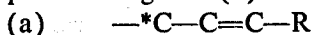


$W3 = 4$ [FCARBN] if X is Nitrogen
 3 [FCARBO] if X is Oxygen
 2 [FCARBS] if X is Sulfur
 1 otherwise.

[function CARBONYL]

Thus for a bond connecting two carbon atoms in the molecule, the intensities of the two fragments depend upon the context-dependent and context-independent factors (the Z s and W s) as just described. The atoms closest to the bond have the greatest effect, but two of the factors ($Z3$ and $W1$) depend upon atoms farther away from this bond.

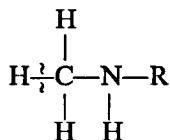
7. $W4$ is a factor which attempts to capture the favorable influence of allylic bonds on the fragmentation process. For example, in fragment (a) below the bond marked with an asterisk is an allylic bond (relative to the double bond of the fragment) and thus increases the probability of fragmentation to produce fragment (a).



[function ALLYLIC,
parameter KALLYLIC]

Cleavage of Carbon-Heteroatom Bonds. For a single bond between a carbon atom and a heteroatom, several of the same calculations are made as for carbon-carbon bonds. In accordance with existing theory, the carbon-containing fragment is much more likely to be ionized as a result of this cleavage than the other one. (In rearrangement products, however, the heteroatom-containing fragment often retains the charge; see the amine entries of Table 8 for example.) But this bond is less likely to be broken than a single bond between two similar carbon-containing fragments.

A. *Exception.* A bond between a carbon and hydrogen atom breaks if this is an α -cleavage in an amine. For example,



B. *The General Rule.* The equation for calculating the intensity of the carbon-containing fragment at a C—X break is:

$$I_c = Z4 \times Z5 \times W5.$$

As before the Z s are context-dependent factors and the W is context-independent.

For the heteroatom-containing fragment the intensity is merely:

$$I_x = Z4.$$

1. $Z4$ is directly analogous to the factor $Z1$ for carbon-carbon bonds. The carbon-containing fragment resulting from such cleavage ordinarily should have a smaller intensity than the corresponding fragment in cleavage of a carbon-carbon bond. The program accounts for this in the first two steps for calculating $Z4$.

(a) Compute a factor ($T4$) which is to be equal to the sum of the intensities of both fragments:

$T4$ = the intensity which would be assigned to the carbon-containing fragment in a similar carbon-carbon bond environment. (The program 'pretends' that the heteroatom is a carbon atom and computes $T4 = ((Z1 + Z2 + Z3 + W1) \times W2 \times W3)$ as above for the intensity to be divided between the two fragments.)

(b) Compute a ratio for splitting $T4$ between the two fragments according to the number of heteroatoms in the carbon-containing fragment which are alpha to the bond under consideration:

ratio = 10:1 if the number of heteroatoms attached to this carbon atom is 0

20:1 if the number of heteroatoms . . . is 1

30:1 if the number of heteroatoms . . . is 2

40:1 if the number of heteroatoms . . . is 3

MACHINE LEARNING AND HEURISTIC PROGRAMMING

(weighted in favor of the carbon-containing fragment). Z4 for each fragment is then the result of applying this ratio to T4. The smaller intensity is returned for the non-carbon fragment.

[function LCALCX]

For the carbon-containing fragment two additional context-dependent factors Z5 and W5 are calculated. The intensity for this fragment is the product

$$Z4 \times Z5 \times W5.$$

2. W5 = 5[FHETN] if nitrogen is singly bonded to the carbon at the break point

4[FHETS] if sulphur . . .

3[FHETO] if oxygen . . .

2[FHETCL] if chlorine . . .

1 otherwise.

[function HETERO]

type	characteristic subgraph	allocation of intensity units of the parent ion [and parameter names]		
		K1% of parent's intensity (for parent)	K2% of parent's intensity (for daughter)	skeleton of daughter ion
Rearrangement of Amines	$\begin{array}{c} + \\ -\text{H}_2\text{C}-\text{N}-\text{R}_1 \\ \\ \text{R}_2 \end{array}$	100 [KAM3OLD]	80 [KAM3NEW]	$\begin{array}{c} + \\ \text{H}_2\text{C}=\text{N}-\text{R} \\ \\ \text{H} \end{array}$
	(Check degree of substitution and number of carbon atoms in R1 and R2 to see which drops away)			
Rearrangement of Ethers and Thioethers	$\begin{array}{c} + \\ -\text{H}_2\text{C}-\text{X}-\text{R} \\ (\text{X}=\text{O},\text{S}) \end{array}$	100 [KAM2OLD]	80 [KAM2NEW]	$\begin{array}{c} + \\ \text{H}_2\text{C}=\text{NH}_2 \\ + \\ \text{H}_2\text{C}=\text{XH} \end{array}$
McLafferty Rearrangement	$\begin{array}{c} \text{X}^+ \\ \\ \text{C} \\ / \quad \backslash \\ \text{Z} \quad \text{X}-\text{X}-\text{X} \\ (\text{X}=\text{C},\text{N},\text{O},\text{P},\text{S}) \\ (\text{Z}=\text{C},\text{H},\text{N},\text{O},\text{P},\text{S}) \end{array}$	40 [KMCOLD]	60 [KMCNEW]	$\begin{array}{c} + \\ \text{XH} \\ \\ \text{C} \\ / \quad \backslash \\ \text{Z} \quad \text{X} \end{array}$
Type F (Biemann)	$\begin{array}{c} +\text{X} \quad \text{H} \\ \quad \\ \text{C}-\text{C}-\text{C} \\ (\text{X}=\text{C},\text{O},\text{N},\text{S}) \end{array}$	80 [KFOLD]	20 [KFNEW]	$\begin{array}{c} +\text{X} \\ \\ \text{HC} \end{array}$
Type G (Biemann)	$\begin{array}{c} \text{H} \\ \\ \text{C}-\text{C}-\text{N}=\text{C} \\ \\ \text{H} \end{array}$	20 [KGOLD]	80 [KGNEW]	$\begin{array}{c} +\text{N}=\text{C} \\ \\ \text{H} \end{array}$

Table 8. Rearrangements for fragment ions

3. Z5=5[KINDCL] if the heteroatom *X* at this C—X breaks is chlorine
 4[KINDBR] if the heteroatom . . . is bromine, oxygen or sulphur
 3[KINDI] if the heteroatom . . . is iodine
 1 otherwise [function INDUCTIVE]

The next section shows some examples with brief explanations of the PREDICTOR's work. Whenever a chemist finds major discrepancies between predicted and actual spectra, we try to localize the contributing functions or parameters and change them. The specialized rules of Table 6 and Table 8 in particular, directly resulted from finding major errors in predictions for ketones, amines, ethers, and alcohols. Instead of adjusting the core of the theory in these cases, however, special tests and branches were added. At a later date, we hope to be able to reunify the PREDICTOR's theory.

4.3. Examples

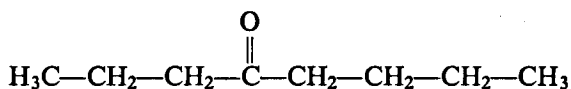
The command DRAW in each case started the predictor's work on the indicated structures. The list of number pairs following the command is the output from the program: the mass-intensity pairs of the most significant ionized fragments.

Example A

(DRAW (QUOTE C2110ClClCClClClC\$))

((43 . 100) (57 . 88) (58 . 22) (71 . 100) (85 . 88) (86 . 22) (128 . 14))

The graphical representation for this molecule, 4-octanone, is



The mass spectrum for this compound from the Stanford University Mass Spectrometry Laboratory is

((41 . 48) (42 . 8) (43 . 100) (44 . 3) (53 . 2) (55 . 8) (56 . 2) (57 . 92)
 (58 . 56) (59 . 2) (64 . 1) (67 . 1) (69 . 3) (70 . 1) (71 . 91) (72 . 4) (81 . 1)
 (83 . 1) (84 . 1) (85 . 60) (86 . 23) (87 . 2) (99 . 3) (113 . 2) (128 . 13)
 (129 . 1))

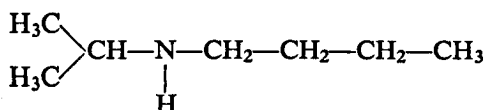
The molecular ion has mass 128. The two other even numbered peaks of high intensity, 86 and 58, are the results of the McLafferty rearrangement of the molecular ion (twice). The peaks at 85 and 71 result from alpha-cleavages, in each case with only the heteroatom-containing fragment retaining the charge. The peaks at 57 and 43 come from loss of carbon monoxide (mass 28) from each of the alpha cleavage fragments. The remaining peaks in the actual spectrum are of little informative value to chemists, thus they remain unpredicted. Several of these could be regarded as isotope peaks and thus could have been predicted (by setting IPEAKS = T).

Example B

(DRAW (QUOTE Cl11CCNlClClClC\$))

((30 . 17) (44 . 80) (72 . 21) (100 . 100) (115 . 2))

This molecule is graphically represented as



and its actual mass spectrum (from the Stanford Mass Spectrometry Laboratory) is

((41 . 38) (42 . 21) (43 . 25) (44 . 88) (45 . 2) (54 . 1) (55 . 3) (56 . 8)
 (57 . 13) (58 . 16) (70 . 11) (71 . 4) (72 . 100) (73 . 4) (84 . 2) (85 . 2)
 (98 . 3) (100 . 61) (101 . 4) (114 . 2) (115 . 5))

The molecular ion is of mass 115. Alpha-cleavage accounts for the peaks at 100 and 72, in each case with only the nitrogen-containing fragment retaining the charge. The amine rearrangement shown in Table 8 affects each of the alpha-cleavage fragment ions resulting in the peaks at masses 44 and 30. In the actual spectrum, peaks below mass 41 were not recorded, but it is not unreasonable to believe that the peak at mass 30 would be a strong peak. Some of the other discrepancies may be due to isotope peaks; many of the rest from lack of rules for amine fragmentation processes.

4.4. Using the mass spectrum PREDICTOR

To run the program, once it is in core, call the top level function DRAW. This function requires one argument, a name of the quasi-DENDRAL symbol string which represents the molecule whose spectrum is to be predicted. For example, either (a), or (b) below would serve for predicting a spectrum for glycine: HO—C—CH₂—NH₂:



- (a) (DRAW (QUOTE C121OOC1N\$))
 (b) (SETQ GLYCINE (QUOTE C121OOC1N\$))
 (DRAW GLYCINE)

Quasi-DENDRAL notation is just DENDRAL dot notation (not necessarily canonical) with four changes:

1. dots are replaced by numerals to indicate bonds,
2. the symbol string is terminated with a dollar sign,
3. atom names longer than one character are surrounded by asterisks, for example *CL* for a chlorine atom, and
4. central bond molecules are prefixed with a special character – the value of the variable CENTRALBOND (currently an asterisk).

The output of the PREDICTOR is a list of number pairs, representing the mass-intensity pairs of the predicted spectrum. Peaks below mass 29 are omitted and intensities are adjusted to percent of the base peak (highest peak). Several options are available to the user to help him interpret the program's work. Section 4.5 lists those currently available.

Also, users familiar with the program can change its theory substantially by resetting parameters before calling DRAW. Section 4.2 indicates many of the parameter names and current values as well as brief descriptions of their effects. Since the functions using these global variables are so intertwined, it is impossible to describe the effects in all contexts. Thus it is generally helpful to look at several examples before and after changing parameters.

4.5. Options and how to use them

1. Print the spectrum as a bar graph instead of as a list of number pairs. Give a name of the spectrum to the function PSPEC, e.g.,

```
(PSPEC (DRAW (QUOTE C1N1C1C1C$)))
```

```
or (PSPEC (QUOTE ((15 . 20) (29 . 40) (30 . 22) . . . )))
```

```
or (PSPEC SPECNAME), where 'SPECNAME' is the name of a list of number pairs.
```

2. Print an analysis of the last predicted spectrum. This function prints all the number pairs of the spectrum, in order of descending mass units, with a short note explaining the source of the peak.

The synopsis printed by the function SCAN first indicates the structure of the molecule in DENDRAL dot notation, except with numbered atoms replacing the atom-types. For example C1=.01 02 C2.N1 for glycine, as given in the quasi-DENDRAL notation above. The mass-intensity pairs for the ions are indicated in order of decreasing masses. An indication of the source of the pair follows each pair:

- (a) (MOL ION) following a mass-intensity pair indicates that this is the pair resulting from the unfragmented molecule.
- (b) (*RR MOL) indicates that the pair resulted from some rearrangement of the molecular ion.
- (c) (C4 1 C3) indicates that the pair resulted from breaking the single bond between atoms C4 and C3.
- (d) (C4 1 C3:*RR C4) indicates that after the bond between C4 and C3 was broken, the fragment containing C4 underwent some rearrangement which resulted in the mass-intensity pair on this line.

To obtain this analysis, call the function SCAN after the spectrum has been calculated: (SCAN).

3. Calculate isotope peaks. After the mass-intensity pair calculations have been made for a fragment, the program can generate a cluster of peaks around the original one to account for isotopic variations of the fragment and addition of extra protons to the fragments. This feature is optional since the most significant peaks in spectra often do not include isotope peaks. Set the global variable IPEAKS to T (true) before calling DRAW: (SETQ IPEAKS T).

4. Print an on-line report of progress, including:

- (a) the bond under consideration
- (b) the rearrangement products

- (c) the features considered and the numerical values associated with them during intensity calculation.

To monitor progress in this way, set the global variable SPEAK to T: (SETQ SPEAK T).

5. Include mass units below 29 in the spectrum. Use the function DRAW 100 instead of DRAW as the top-level function.

5. THE EVALUATION FUNCTION

After candidate structures have been generated by the STRUCTURE GENERATOR, the program needs some way to attach a degree of plausibility to each one. The PREDICTOR makes predictions for each one; the EVALUATION FUNCTION must now reflect the degree to which the predictions confirm or disconfirm each candidate hypothesis. Strictly numerical evaluation functions score predicted spectra on the basis of how much they 'cover' the peaks in the original spectrum without adding spurious peaks – perhaps weighting various kinds of failures. But all of these fail to account for the higher theoretical significance of some peaks over others, regardless of the numbers involved. After experimenting with such numerical evaluation functions, their inadequacies became obvious.

The current evaluation function is relatively untried, but its theoretical base is much sounder than that of previous functions. The PREDICTOR now marks various kinds of cleavages and rearrangements as being very significant from a theoretical point of view. For example, the results of alpha-cleavage in ketones, amines and ethers are put on a global list named SIGNIFICANT, together with the results of other theoretically significant peaks in the predicted spectrum. At the end of the PREDICTOR's run this global list remains set for use by the EVALUATION FUNCTION. Evaluation is a two-step process here: (A) reject any candidate whose predictions are inconsistent with the original data, and (B) rank the remaining candidates. (A) For each candidate molecule the EVALUATION FUNCTION looks in the original spectrum for each member of this list of significant peaks. Either a significant predicted mass point is represented in the original spectrum or it is not. If there is a peak at this mass point, x , and its intensity level is higher than the expected intensity level from an isotope peak (1% of the intensity of the $x-1$ peak times the estimated maximum number of carbon atoms in the $x-1$ peak), then the next significant predicted peak is considered. When the evaluation routine decides that the original spectrum shows a significant peak only because this is an isotope peak, the candidate is rejected. Rejection of a candidate is accompanied by a message explaining which significant peaks were missing from the original spectrum or were present only in amounts expected from isotopic variations, as shown in the examples in the following section. If the significant peak is not present in the original spectrum and other masses in this region were recorded in the original spectrum, then

this candidate is rejected entirely. For example, if the predicted spectrum shows rearrangement peaks at the wrong mass points, it should not warrant further consideration since the theory is strongly violated by that candidate.

When only high mass peaks have been recorded in the original spectrum, as is frequently the case, and one of the significant peaks thus fails to appear in the spectrum, the EVALUATION FUNCTION notes this fact on the list LOWPKS. For example, a significant peak at mass 15 will not be found in a spectrum which starts at mass 40. No candidate is ruled out by the failure to match unrecorded peaks since a more complete spectrum may well include them. On the other hand, there is no assurance that these significant low mass peaks would, in fact, appear if low masses had been recorded.

(B) For each candidate, the list SIGNIFICANT is matched against the original spectrum. If the candidate is rejected, all of the missing significant peaks are printed as justification for rejecting it. The second step of this routine is to rank the remaining candidates, each of which accounts for some of the non-isotopic peaks in the recorded spectrum, but not necessarily all. The best candidate is taken to be the one which accounts for the most peaks, as one should expect. In case of ties, the preferred molecule is the one with the lower number of unrecorded low mass peaks in doubt (as saved on the list named LOWPKS). The rest of the peaks in the predicted and actual spectra are not used at all presently. However, we may want to resolve ties by a numerical scoring of the remaining (non-significant) peaks in the spectra.

Examples of the results of this EVALUATION FUNCTION are shown in the next section.

6. EXAMPLES, SUMMARY, AND CONCLUSIONS

The preceding sections have promised that section 6 would include examples showing the entire operation of the HEURISTIC DENDRAL program. Two simple examples are shown in Tables 9 and 10. These examples seem trivial until one considers the possible list of answers which could have been generated. The total number of structures for the composition $C_8H_{16}O$ is about seven hundred 'chemically stable' structures. (Several thousand others were eliminated by BADLIST.) Forty of these are ketone structures, yet the particular spectra (S:09320 and S:09046) enable the program to reduce the output to one and three structures, respectively.

We have been using the program in another mode, namely to direct a search of chemical literature to determine which structural isomers of a given composition have previously been synthesized by chemists. In the case of threonine (composition $C_4H_9NO_3$), we estimate that there are several thousand isomers (unrestricted by BADLIST). Approximately 750 of these are considered 'chemically stable' but only about sixty of these have been reported in standard chemical references. This disparity has great significance for chemists because of the number of potentially useful compounds that may be found among the 890 'new' structures.

MACHINE LEARNING AND HEURISTIC PROGRAMMING

(EXPLAIN (QUOTE C8H16O) s:09320 (QUOTE TEST2) (QUOTE JULY8))
 *GOODLIST=(*N-PROPYL-KETONE3*)
 *BADLIST=(*C-2-ALCOHOL* *PRIMARY-ALCOHOL* *ETHYL-ETHER2*
 METHYL-ETHER2 *ETHER2* *ALDEHYDE* *ALCOHOL* *ISO-PROPYL-
 KETONE3* *ETHYL-KETONE3* *METHYL-KETONE3*)

 (JULY-4-1968 VERSION)
 C*N-PROPYL-KETONE3*H8
 MOLECULES NO DOUBLE BOND EQUIVS
 1. C=.. O C3H7 CH2.C3H7,
 2. C=.. O C3H7 CH2.CH..CH3 CH3.
 DONE

 (SCORE (QUOTE TEST2) s:09320)
 JULY-8-1968
 1.) c211oc1c1ccl1c1c1c\$
 ((43.100)(57.88)(58.22)(71.100)(85.88)(86.22)
 (128.14))
 2.) c211oc1c1ccl1c1cc\$
 ((43.87)(57.100)(58.8)(71.87)(85.100)(86.4)
 (100.4)(128.16))
 *THIS CANDIDATE IS REJECTED BECAUSE OF (100).

 *LIST OF RANKED MOLECULES:

1. # 1.
 s = 6.
 P = (57 71 43 85 86 58)
 U = NIL

*1. # N MEANS THE FIRST RANKED MOLECULE IS THE NTH IN THE ORIGINAL NUMBERED LIST ABOVE. S = THE SCORE (HIGHEST = BEST) BASED ON THE NUMBER OF SIGNIFICANT PREDICTED PEAKS IN THE ORIGINAL SPECTRUM. P = THE LIST OF SIGNIFICANT PREDICTED PEAKS. U = THE LIST OF POSSIBLY SIGNIFICANT UNRECORDED PEAKS USED IN RESOLVING SCORING TIES (THE FEWER IN DOUBT THE BETTER).
 DONE

Table 9. An example of HEURISTIC DENDRAL output: 4-Octanone

(EXPLAIN (QUOTE C8H16O) s:09046 (QUOTE TEST1) (QUOTE JULY8))
 *GOODLIST=(*ETHYL-KETONE3*)
 *BADLIST=(*C-2-ALCOHOL* *PRIMARY-ALCOHOL* *ETHYL-ETHER2*
 METHYL-ETHER2 *ETHER2* *ALDEHYDE* *ALCOHOL* *ISO-PROPYL-
 KETONE3* *N-PROPYL-KETONE3* *METHYL-KETONE3*)

 (JULY-4-1968 VERSION)
 C2*ETHYL-KETONE3*H8
 MOLECULES NO DOUBLE BOND EQUIVS
 1. CH2..CH2.C3H7 C=.O C2H5,
 2. CH2..CH..CH3 C2H5 C=.O C2H5,
 3. CH2..CH2.CH..CH3 CH3 C=.O C2H5.
 DONE

Table 10. An example of HEURISTIC DENDRAL output: 3-Octanone

(SCORE (QUOTE TEST1) S:09046)

JULY-8-1968

1.) c11c1c1c1cc21oc1c\$

((29 . 100)(57 . 100) (71 . 70) (85 . 40) (99 . 70) (128 . 13))

2.) c11c11cc1cc21oc1c\$

((29 . 100) (57 . 100) (71 . 100) (72 . 4) (99 . 100) (128 . 19))

3.) c11c1c11ccc21oc1c\$

((29 . 100) (57 . 100) (71 . 87) (99 . 87) (128 . 16))

*LIST OF RANKED MOLECULES:

1. # 2

S = 5.

P = (29 99 57 71 72)

U = (29)

2. # 1

S = 4.

P = (29 99 57 71)

U = (29)

3. # 3

S = 4.

P = (29 99 57 71)

U = (29)

*1. # N MEANS THE FIRST . . .
(see Table 9)

Table 10 (contd.). An example of HEURISTIC DENDRAL output: 3-Octanone

We realize that the internal structure of HEURISTIC DENDRAL has not been presented in much detail. No very unusual programming has been employed, however; but we have taken full advantage of the facilities of LISP 1.5. What we have tried to present in this paper is the global strategy of the program. Between the global strategy of a program and its coded functions there are many levels of complexity. We have tried to keep an eye on both extremes and to stay roughly mid-way between them in order to show how some of the heuristics of the program work, how the various subroutines are tied together, and how we plan to expand the program to cover cyclic structures and more classes of acyclic structures.

Recently we have had some ideas of how to rewrite HEURISTIC DENDRAL to separate more completely the model of chemistry from the graph manipulating processes. This will be our next big programming effort. Hopefully the revised program will handle rings without making them special cases. The program's poor handling of ringed structures is now its major deficiency.

In limited areas, the current program performs its two major tasks with a fair measure of success.

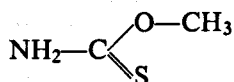
- (1) Using Lederberg's DENDRAL algorithm plus a theory of chemical stability, the STRUCTURE GENERATOR can construct all acyclic isomers (structural variants) of a given composition, either including or rejecting unstable structures.
- (2) With the more interesting task of explaining the data from a mass spectrometer by finding molecular structures which best account for the data, the program approaches the chemists' level of sophistication only for a few select classes of molecules. Expanding the program to cover more classes depends upon much interaction with chemists, but no new programming strategies are anticipated.

7. A SUMMARY OF THE DENDRAL ALGORITHM

DENDRAL is a system of topological ordering of organic molecules as tree structures. Proper DENDRAL includes precise rules to maintain the uniqueness and the non-ambiguity of its representations of chemical structures. Each structure has an ordered place, regardless of its notation; the emphasis is upon topological uniqueness and efficient representation of molecular structures. The principal distinction of DENDRAL is its algorithmic character. DENDRAL aims (1) to establish a unique (i.e., canonical) description of a given structure; (2) to arrive at the canonical form through mechanistic rules, minimizing repetitive searches and geometric intuition; and (3) to facilitate the ordering of the isomers at any point in the scan, thus also facilitating the enumeration of all of the isomers.

The DENDRAL representation of a structure is made up of operators and operands. The operators are valence bonds issuing from an atom. Each bond looks for a single complete operand. An operand is (recursively) defined as an unbonded atom, or an atom whose following bonds are all satisfied in turn by operands. Hydrogen atoms are usually omitted, but are assumed to complete the valence requirements of each atom in the structure. If the structure has unsaturations (one unsaturation for each pair of hydrogen atoms by which the structure falls short of saturation), these are indicated by locations of double and triple bond operators. Single, double, and triple bonds are represented by . , : , and ; respectively. The operator : may be represented by = and the operator ; by \$.

As an example, the molecule



has one unsaturation and may be written in many ways, including:

- (1) C.O.C.:NS
- (2) C.O.C.:SN
- (3) O..CC.:NS
- (4) O..C.:NSC
- (5) C..:O.CNS
- (6) C..:O.CSN
- (7) C..:NSO.C (canonical)
- (8) C..:SNO.C
- (9) S:C..O.CN
- (10) N.C.:O.CS

Each of these ten notations is a non-ambiguous representation of the molecule. However, proper DENDRAL also specifies that the representation be unique. The key to obtaining the unique or 'canonical' representation is the

recognition of the unique center of any tree structure and the subsequent ordering of successive branches of the tree.

The centroid of a tree-type chemical structure is the bond or atom that most evenly divides the tree. A molecule will fall into just *one* of the following categories, tested in sequence. Let V be the count of non-hydrogen atoms in the molecule. Then either

- A. *Two central radicals* of equal count are either (1) united by a leading bond (V is even) or (2) sister branches from an apical node (V is odd);
or
- B. *Three or more central radicals*, each counting less than $V/2$, stem from a single apical node.

In the first case, the centroid is a bond, and the canonical representation is an operator followed by two operands. In the other two cases the centroid is an atom, and the canonical representation is an operand in the form of an atom followed by two or more bonds and operands. In every case where two or more bonds follow an atom, the operands must be listed in ascending DENDRAL order.

DENDRAL order (or simply 'weight') is a function of the composition and arrangement of a structure and finds its primary use when comparing two operands (radicals). The weight of a radical is evaluated by the following criteria (in descending significance): count, composition, unsaturation, next node, attached substructures.

Count is the number of skeletal (non-hydrogen) atoms. Of two radicals, the one with the higher count is of higher weight.

Composition refers to the atoms contained in the radical. An arbitrary ordering of the atoms makes carbon less than nitrogen less than oxygen less than phosphorus less than sulfur, $C < N < O < P < S$. (This ordering is alphabetical as well as by atomic number.) When comparing two radicals of the same count, the one with the fewer number of carbons has lesser weight. If carbons are equal, the one with the fewer nitrogens is of lesser weight. And so forth.

Unsaturation counts the number of extra bonds (1 for a double bond, 2 for a triple bond) in the radical, including those (if any) in the afferent link (the bond leading into the radical). Of two radicals, the one with the greater number of unsaturations has the greater weight.

The *next node* or *apical node* refers to the first atom in the radical (the one connected to the afferent link). When comparing two apical nodes, the following three criteria are evaluated (in order of decreasing significance):

Degree is the number of afferent (attached) radicals. The apical node with the most radicals attached to it has the greater weight.

Composition refers to the type of atom. A carbon atom is the lowest apical node, while a sulfur atom is the highest.

Afferent link refers to the bond leading to the apical node. A single bond afferent link is the lowest, a triple bond is the highest.

If the above criteria fail to determine which of two radicals has the greater weight, then the radicals appendant on the two apical nodes must be arranged in increasing order and compared in pairs. The first inequality in weight of appendant radicals determines the relative weight of the original radicals.

The canonical representation for the molecule in the example given earlier is notation #7. It must be a central atom molecule since its count (ignoring hydrogen atoms) is 5; and the non-terminal carbon atom is the only atom which has all its appendant radicals with counts less than $5/2$. Of the three appendant radicals, the one containing two atoms has the highest count and thus is the heaviest. Of the two radicals containing a single atom each, the one with the double bond is the heavier because it has more unsaturations.

Even-count molecules may have a bond for center, if the count of the molecule is evenly divided by cutting that bond. Thus, the canonical form for $\text{NH}_2\text{—CH}_2\text{—CH}_2\text{—OH}$ is .C.NC.O, a leading bond, the first dot, calling for two operands.

The collection of rules and conventions described above provides a unique and non-ambiguous representation for any non-ringed chemical structure. In addition, the rules also allow us to construct the 'lowest' structure which can be made from a composition (collection of atoms). Once this lowest structure has been made, it may be transformed by a process of rearranging its atoms and unsaturations into the 'next to lowest' structure. This 'incrementing' process may be continued until the 'highest' structure has been made.

REFERENCES AND READING

- Lederberg, J. (1964) DENDRAL-64, *A System for Computer Construction, Enumeration and Notation of Organic Molecules as Tree Structures and Cyclic Graphs*, Parts I-V. Interim Report to the National Aeronautics and Space Administration.
- Lederberg, J. & Feigenbaum, E. A. (1967) *Mechanization of Inductive Inference in Organic Chemistry*. Stanford Artificial Intelligence Project, Memo No. 54.
- Sutherland, G. (1967) DENDRAL-A Computer Program for Generating and Filtering Chemical Structures. Stanford Artificial Intelligence Project, Memo No. 49.

A Chess-Playing Program

J. J. Scott

The Computer Laboratory
University of Lancaster

INTRODUCTION

Aims

The aim of the project is to produce a chess-playing program which is capable of playing good human chess with reasonable move times. I also hope that some general principles of machine intelligence may emerge from the work.

Project and program segmentation

The program, mirroring the aspects of the project, falls easily into three major segments:

1. a 'communication' segment, which can describe the computer's moves in an acceptable format, can decode its opponent's moves from a standard chess notation and check their validity and non-ambiguity, and can provide other necessary facilities;
2. a 'thinking' segment, which, when given a board position, can decide which move to make;
3. a 'learning' segment, which can
 - (i) adjust the parameters of the 'thinking' segment so as to optimise it,
 - (ii) record opening sequences and their relative merits, and use these to provide alternatives to the moves generated by the 'thinking' segment.

At the time of writing the 'communication' segment is fairly complete; the 'thinking' segment is being continually improved, and at the moment is being upgraded to play 'full chess' as opposed to a 'restricted chess', which does not allow pawn-promotion, castling or pawn-taking *en passant* (previously omitted so as to get a clearer picture of the problems associated directly with choice-making); the 'learning' segment is virtually non-existent, the function of parameter adjustment being performed manually by myself.

Implementation

The latest version is written in PLAN 3 for an ICT 1909/5 machine with a 2 micro-second store.

The program requires between 8 and 8.5 K of store, a short overlay file on disc, and an interrogating typewriter; it also optionally uses one paper tape reader and one paper tape punch.

Development of the program has been fairly rapid. As the program requires only a small virtual machine, it can be time-shared with a reasonable proportion of the Laboratory's job load, including jobs run under the operating system, and in this way much testing has been possible.

THE COMMUNICATION SEGMENT

Introduction

A communication segment is a vital though perhaps not a very interesting part of any program involving human interaction. Especially when the program is to hold a conversation, or the like, with someone who knows little about computers, it is necessary to make the computer as human-like as possible to get an accurate performance from the person.

All communication is via an interrogating typewriter. For examples of dialogue see Appendix 1.

Starting or re-starting a game

To start or re-start a game a short series of questions are asked by the program (most requiring just a 'yes' or 'no' answer) to set the game up as required.

Output: move description

The format used in the presentation of the computer's moves is an extended form of the Standard English Notation, for example:

(a) N(QN1) - QB3

to be read: 'the knight on queen's knight one moves to queen's bishop three'.

(b) B(KN5) x N(KB6)

(All positions here refer to the computer's side).

Input: move description

The program will accept most variations of the Standard English Notation (see Appendix 1). Checks are made on (a) format, (b) legality of move, (c) non-ambiguity of move.

Other facilities

These include directives to obtain a picture of the board for checking purposes, temporarily suspending games, terminating games by resigning, etc.

THE THINKING SEGMENT

Introduction

This is the most interesting part of the program, for it involves the application of many principles of machine intelligence. As such, I have devoted a large proportion of my time and this paper to it. Many of the methods and principles in this segment have been derived by considering the methods and principles of human thought (especially my own).

The basic method of choosing a move

A fairly good definition of the best move in any given position is 'that move which gives most chance of winning as soon as possible'. To determine the best move using this definition would involve constructing an exhaustive move tree (i.e., a graph of all possible combinations of moves), each branch of the tree being terminated only by a winning, drawing or losing position. Unfortunately, in the majority of positions (the exceptions being end game positions near to a mate, etc.) the amount of calculation required to do this is prohibitive.

Hence, some compromise has to be made: in this chess program, as in others, the method of finding the best move is to construct a move tree of all possible combinations of moves to a depth of n moves (i.e., $2n$ 'half-moves' or 'plys'), where n is fixed for any particular search, then 'mini-maxing' to choose the move (*see* Michie, 1966, for a general review).

A prototype: the MK1 version

The first version of this program was written in FORTRAN using a very simple-minded approach.

The best move was determined by using a simple evaluation function applied at each terminal node of a 2-ply move tree.

The evaluation function consisted of the following:

- (i) points were given to each piece for its existence on the board in the ratio
 - P = 1,
 - N = 3,
 - B = 3.5,
 - R = 5,
 - Q = 9,
 - K = 1000;
- (ii) additional points were given for the degree of advancement of each pawn;
- (iii) points were given for the mobility of each piece (i.e., the number of free squares to which each can move), the number depending on the type of piece under consideration;

- (iv) points were given for any attacks, the amount being determined by the piece under attack (but not the attacking piece).

This version played fairly poor chess, but the writing and developing of it was a very instructive experiment, and it was decided that if the play was to be improved, either the evaluation function should be made more powerful or the program should look further ahead.

Thinking time

The MK1 version showed that a very important factor in programs performing the type of machine intelligence involved in playing chess, is that of 'thinking time'. It seems reasonable to suppose that, *given long enough*, a program can draw all the conclusions from a set of data that it is possible to draw, even though this may be millions of years; the intelligence in such programs is the ability to draw the necessary conclusions while doing the minimal amount of work. To stress this point I cite the following example:

The MK1 version took, on average, about 30 seconds to find the best move looking at a depth of 2 half-moves. If the program had been altered to look 4 half-moves ahead without devising any time-saving methods, the amount of calculation required would have been multiplied by a factor of 1,000, and the average move time would have been of the order of 8 hours. In the latest version, using the time-saving methods described in some of the following sections, the average time for such a search is approximately 45 seconds—faster by a factor of 700!

Hence, methods were devised to speed up the program and these fall into three main categories:

1. optimisation of the code of the program;
2. reduction of the effective size of the search tree by applying the alpha-beta heuristic, etc;
3. reduction of the time taken for position evaluation by using methods of value alteration rather than complete re-evaluation.

More details of these methods are given in some of the following sections.

THE LATEST VERSION OF THE THINKING SEGMENT: MK3

The evaluation function

Like the MK1 version the method of finding the best move consists of looking ahead an even number of half-moves then applying a simple evaluation function at the terminal nodes of the move tree, but in this case the search depth is not limited to 2 half-moves but is permitted to vary under the control of various parameters, a typical depth being 4 half-moves.

Additional features were added to the evaluation function which now consists of:

- (i) points given to each piece for its existence on the board in the ratio
 - P=1,
 - N=3,
 - B=3.5,
 - R=5,
 - Q=9,
 - K=a relatively infinite value;
- (ii) additional points given for the position of pawns on the board, in general the number becoming greater the more advanced the pawn and the more central the file on which the pawn lies;
- (iii) points given for the mobility of (i.e., the number of free squares available to) each piece, the number depending on the type of the piece;
- (iv) points given for both attacks and defences, the number being dependent on both the attacked / defended piece, and the attacking / defending piece.

Trial-and-error experiments showed that the magnitude of all the points described above should be symmetric with respect to possession (for example, that the value of a knight attack on a queen should be the same (but of opposite sign) whether the queen belongs to the computer or its opponent).

To counteract the unbalancing effect of the fact that some of the attacks of a side are much more potent if it is that side's turn to move, the following principle is adopted: if the last half-move before the evaluation function is applied implies an advantage for the side making the move, the magnitude of the improvement is reduced and an upper limit is placed on it.

Various other tricks are used to give the program a clear understanding of mate and stalemate, etc.

Adjustment of the value obtained from the evaluation function as opposed to total re-calculation

To describe the methods used to adjust the value of the evaluation function it is first necessary to describe the exact way in which the program values a position; the procedure used is as follows:

A contributory value is attributed to each piece on the board and these values are summed to give the value of the position.

To calculate the contributory value of, for example, a bishop:

- (i) give points for its existence on the board;
- (ii) for each of its (four) paths of movement consider, sequentially, longer and longer moves down that path until either another piece or the edge of the board is encountered. For each of the moves to an empty square give mobility points, and, if the path is terminated by a piece, give points for the attack or the defence so implied.

Almost identical methods can be used for the rook and the queen. Similar methods can be used for the king and the knight, but in these cases each path is at most only one move long. Again, the pawn, although slightly more complex can be treated in a similar way.

Let us suppose that the position in figure 1 has been evaluated by the method described above. If White now moves his rook to the opposite R5 then the only pieces which have their contributory values changed are the moved rook, the white king, and the white pawn on N4; the values associated with the other pieces are unchanged. Thus, if the original position is fully evaluated and the necessary information is stored about the interrelations between the pieces on the board, only three pieces need to be revalued.

			Black					
k	r	
b	.	n	
.	
R	
K	P	
P	.	B	
.	
.	
			White					

Figure 1 (lower case letters represent Black)

In a typical game the average number of pieces affected by a move is between 3 and 4. Thus, at the most critical part of the program, with respect to time, this method of re-valuing the position as opposed to evaluating it fully can reduce the amount of calculation necessary by a factor of some 6 or 7 on average.

This method can be extended and improved by subdividing each piece into 'piece-paths', a piece-path being a possible line of movement of a piece; a king consists of 8 piece-paths, a rook of 4, etc.

If evaluation of a position is performed by attributing values to piece-paths, instead of pieces, in an exactly similar way to that described above, value adjustment involves about 15 times less calculation than a complete re-evaluation.

Using this latter method in the MK3 version has reduced the average position evaluation time by a factor of about 10.

Data structure

To hold the necessary information about the relations between piece-paths on the board I have used a data structure that I call a 'three-dimensional list structure' (a term which suggests its physical counterpart: that of a stack of stacks of chess-boards), which is, in fact, a plex, the cells of which are interwoven to form lists holding three different types of information.

Each cell of this structure represents a possible move or a defence. When the 'thinking' segment is entered, this structure is initialised for the initial board position it is given. During the tree search the plex is continually updated (except for the terminal moves) so as to refer to the board position existing at that instant in the search.

Each cell contains pointers for the lists described below and also information to indicate which move it represents, to which piece-path it belongs, etc. The cells are joined up into three lists:

- (a) ('Piece-path lists'). A list for each piece-path linking all cells of that piece-path.
- (b) ('Position lists'). A list for each square on the board linking cells which represent either moves to that square or defences of a piece on that square.
- (c) ('Move lists'). A list for each side of all cells which represent possible moves for that side. For the majority of the time these lists hold the moves approximately in descending order of merit; this enhances the effect of the alpha-beta cut-off (see next section).

To determine which piece-paths are affected by a move (beside the moved piece and any taken piece, etc.) all that is necessary is to look down the 'position lists' of any squares whose contents have been altered by the move: each cell of these lists indicates an affected piece-path.

For all moves except terminal moves (i.e., moves to a terminal node of the move tree) new cells are constructed for all the affected piece-paths and the old cells are removed by making use of the 'piece-path lists'. For the terminal moves of the search, the lists are used to determine (as described earlier) which piece-paths require re-evaluation, the amount of calculation being reduced to a minimum.

The time taken in setting up and maintaining the plex is almost negligible compared with the time saved in the innermost routines which have to be executed much more frequently.

Use of the alpha-beta heuristic

I do not describe this heuristic here because it is amply discussed in other texts. It is the most powerful time-saving tool in the program. The effect of applying it is to reduce the 'branching factor' at each node of the move tree. (The 'branching factor at a node' is the number of alternative moves considered from the position which the node represents.)

Under optimal circumstances, if the branching factor without the application of the heuristic is n , it can be reduced to \sqrt{n} when the heuristic is applied. In the MK3 version, where the moves are held in lists in which the good moves are frequently near the front of the lists, the alpha-beta heuristic is fairly well optimised and the average branching factor has been reduced to about $1.6\sqrt{n}$, where n is of the order of 35 on average. Hence, for a depth 4

half-move search in the middle game, which may involve taking 1,000,000 positions into consideration, only about 7,000 will actually be evaluated.

The efficiency of the heuristic, as applied in this chess program, varies quite significantly with the details of the board position: in general, the more obvious the best move in a position is to a human, the more forced is the position, the more efficient the alpha-beta cut-off, and consequently the computer replies faster.

Note. To assist the ordering of the move lists, for each search of h half-moves, if $h > 3$ a depth $h-2$ half-move search is performed before the bigger search: this technique is applied recursively, thus, for example, a depth 5 half-move search would first call for a depth 3 half-move search and that for a depth 1 half-move search.

Move-time control

The program is given two parameters for any game:

- (i) a limit to the depth of search it may perform,
- (ii) a limit to the average move time it must maintain.

For all moves of a game the program initially performs a depth 2 half-move search, then may progress to consecutively deeper and deeper searches of 4 half-moves, 6 half-moves, etc. After each such search the program first checks whether the depth limit has been reached; if not, then it estimates the time it would take to perform the next deeper search and uses this, together with the degree to which it is above or below the average time limit and various other parameters, to decide whether or not it is desirable to perform the deeper search.

The 'standard restrictions' placed on the program are to have no limit on the depth of search but to have an average move-time limit of $2\frac{1}{2}$ minutes. The result is as follows:

In the opening and middle games most moves are determined by a depth 4 half-move search taking an average of about 45 seconds each, thereby storing up a large time 'reserve'. Near and during the end game the board becomes more empty of pieces, move analysis becomes easier and depth 6 half-move searches become frequent, thus improving the standard of play. Consequently, the program's standard of play, in comparison with human play, is maintained fairly constant throughout the whole game.

THE FUTURE

Several modifications and extensions need to be made to the present version:

- (i) extend to play 'full chess' recognising all the various draws: third repeated move draw, 50 move draw;
- (ii) optimise the code at critical points in the program in order to speed it up and enable deeper searches to be made;
- (iii) write the learning segment: this should generally improve the play of the program;

- (iv) get the program to utilise the opponent's reply time by guessing his move and analysing the resulting position.

After these improvements have been made experiments can be performed on the applications of various methods, such as 'forward pruning' of the move tree, localised deepening of the tree, etc.

CONCLUDING REMARKS

Although several good chess-playing programs are already in existence, this one contains, as far as I know, some unique features:

- (i) use of a very simple evaluation function;
- (ii) adjustment of the value given by the evaluation function, rather than complete re-evaluation, during the tree search;
- (iii) dynamic alteration of the depth of analysis, often enabling the program to play reasonably well right through to the natural end of a game.

Acknowledgements

I am indebted to the Lancaster University Computer Laboratory for supplying the necessary computer time for this work and also to Andrew Colin and Malcolm Atkinson for helpful discussion and criticism.

REFERENCE

Michie, D. (1966), Game-playing and game-learning automata. *Advances in programming and non-numerical computation*, pp. 183-200 (ed. Fox, L.). Oxford: Pergamon Press.

APPENDIX 1

Notation: responses of program's opponent in lower case.

A short game showing complete dialogue

```
#XESS MK 3B   DATE 26/07/68   TIME 10/32/00
IS THAT ALL? no
NAME PLEASE? opponent.1
SHALL WE START A NEW GAME? yes
SHALL I CHOOSE SIDES AT RANDOM? yes
AM I TO PLAY WITH STANDARD RESTRICTIONS? yes
YOU ARE WHITE.
```

TYPE YOUR MOVES BELOW:

```
OPPONENT.1   V   #XESS(BLACK)
1. p-q4      P(Q2)-Q4
2. p-qb4     N(QN1)-QB3
3. b-b4      P(K2)-K4
4. qp x p    B(KB1)-QN5
5. n-qb3     P(Q4)-Q5
6. n-kb3     P(Q5) x N(QB6)
```

MACHINE LEARNING AND HEURISTIC PROGRAMMING

OPPONENT.1 V #XESS(BLACK)
 7. $q \times q$ N(QB3) \times Q(Q1)
 8. $r-n1$ AMBIGUOUS
 8. $r-qn1$ P(QB6)-QB7
 9. $b-q2$ P(QB7) \times R(QN8)
 10. resign

Note: this differs from the actual dialogue only in the change of the opponent's name, a slight tidying up of some typing errors, and a minor re-arrangement of a heading.

APPENDIX 2 : SOME SPECIMEN GAMES

Note: the following games illustrate some of the program's better play

(1) #XESS MK 3A	DATE 20/07/68
<i>White</i>	<i>Black (Program)</i>
1. P-Q4	N(KN1)-KB3
2. N(QN1)-QB3	P(Q2)-Q3
3. P-K4	B(QB1)-K3?
4. N(KN1)-KB3	P(KR2)-KR4
5. P-KR3	P(QR2)-QR4
6. P-QN3	P(KR4)-KR5
7. B-QB4	B(K3) \times B(QB5)
8. P \times B	N(QN1)-QB3
9. B-KN5	P(QN2)-QN3
10. B \times P	R(KR1)-KR4
11. B-KN5	K(K1)-Q2
12. K-K2	Q(Q1)-K1
13. R-K1	R(QR1)-QN1
14. R-QN1	N(QB3)-QN5
15. P-QR3	N(QN5)-QB3
16. N-QN5?	N(KB3) \times P(K5)
17. Q-Q3?	N(K5) \times B(KN4)
18. Q-KB5	N(KN4)-K3!
19. Q \times R?	N(K3)-KB5
20. K-K3	N(KB5) \times Q(KR4)
21. P-N4	N(KR4)-KB3
22. QR-Q1	P(K2)-K3
23. P-QB5	P(QN3) \times P(QB4)
24. P \times P?	R(QN1) \times N(QN4)
25. N-KN5	R(QN4) \times P(QB4)
26. K-KB3?	R(QB4) \times N(KN4)
27. R-Q3	R(KN4)-QB4
28. P-B3	N(QB3)-K4
29. K-K3	N(K4) \times R(Q6)

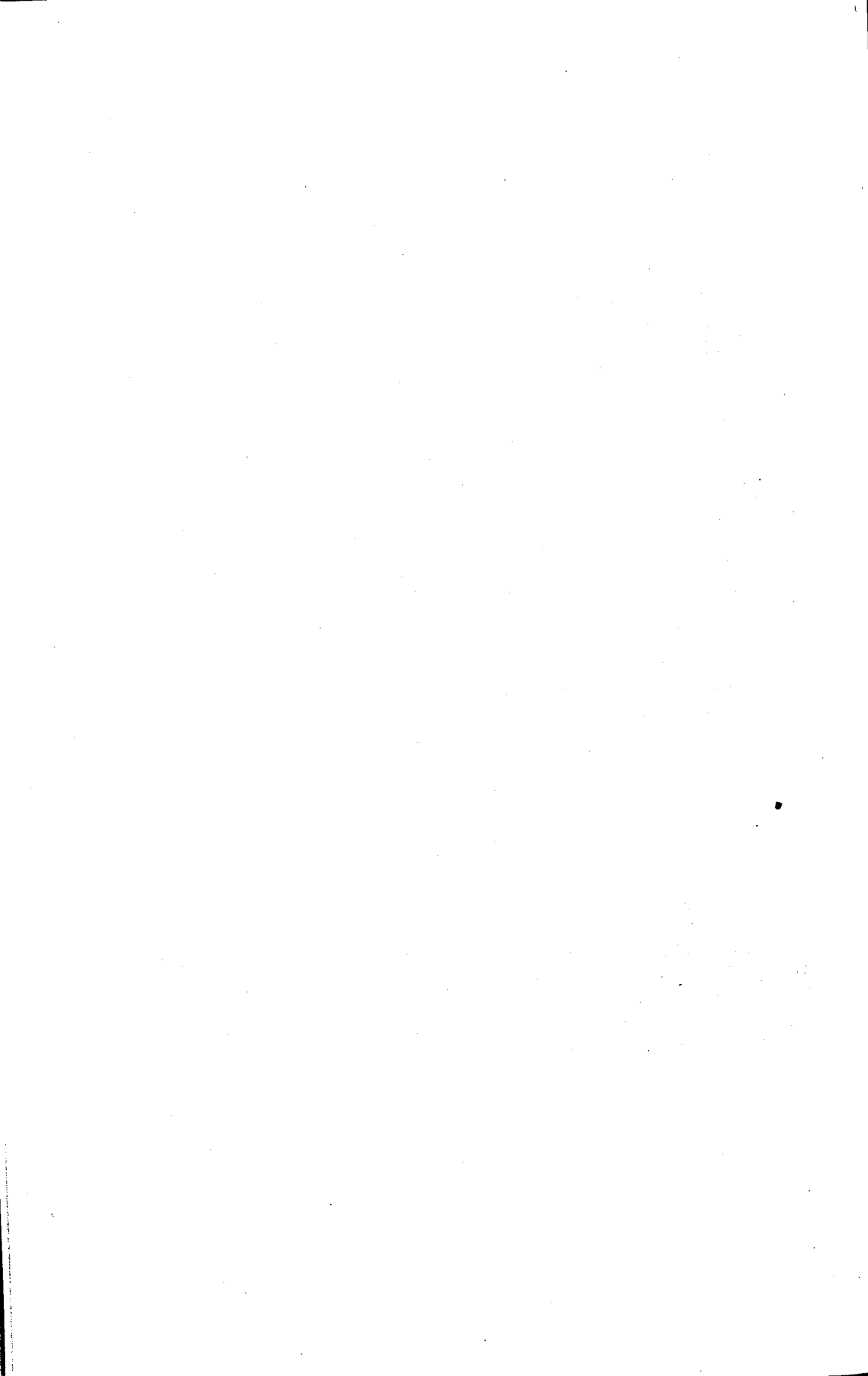
(GAME STOPPED)

(2) #XESS MK 3C

DATE 20/07/68

*White (Program)**Black*

- | | |
|-----------------------|------------|
| 1. P(Q2)-Q4 | N-KB3 |
| 2. N(KN1)-KB3 | P-Q4 |
| 3. N(QN1)-QB3 | P-K3 |
| 4. P(K2)-K3 | B-QN5 |
| 5. P(QR2)-QR3 | B × N |
| 6. P(QN2) × B(QB3) | N-K5 |
| 7. P(QB3)-QB4 | N-QB6 |
| 8. Q(Q1)-Q3 | N-R5 |
| 9. Q(Q3)-QN3 | Q-Q2 ? |
| 10. N(KB3)-K5 | P × P |
| 11. B(KB1) × P(QB4) | Q-K2 |
| 12. Q(QN3) × N(QR4) | N-Q2 |
| 13. P(K3)-K4 | 0-0 |
| 14. Q(QR4)-QN3 | P-QB4 |
| 15. N(K5) × N(Q7) | B × N |
| 16. Q(QN3) × P(QN7) | P × P |
| 17. Q(QN7)-QN4 | Q-KR5 |
| 18. Q(QN4)-QN7 | R(KB1)-Q1 |
| 19. P(KN2)-KN3 | Q-KB3 |
| 20. P(KB2)-KB4 | R(R1)-QN1 |
| 21. Q(QN7) × P(QR7) | P-Q6 |
| 22. P(K4)-K5 | Q-KB4 |
| 23. B(QB4) × P(Q3) | Q-KN5 |
| 24. B(Q3)-K2 | Q-KR6 |
| 25. B(K2)-KB3 | Q-KB4 |
| 26. B(QB1)-K3! | Q × P(QB7) |
| 27. R(QR1)-Q1 | Q-QB6 ch. |
| 28. K(K1)-KB2 | R-QN7 ch. |
| 29. B(KB3)-K2 | Q-QB1 |
| 30. R(Q1)-Q2 | R-QB7 |
| 31. R(KR1)-Q1 | R × R |
| 32. R(Q1) × R(Q2) | P-KB3 |
| 33. B(K3)-QN6! | Q-QR1 |
| 34. R(Q2) × B(Q7) | Q × Q |
| 35. R(Q7) × R(Q8) ch. | K-KB2 |
| 36. B(QN6) × Q(QR7) | K-K2 |
| 37. R(Q8)-QN8 | resigns |



Analysis of the Machine Chess Game,
 J.Scott (White), ICL-1900 *versus*
 R. D. Greenblatt, PDP-10

I. J. Good

Virginia Polytechnic Institute

It is no disgrace for Scott's program to have lost to Greenblatt's which seems to be the best chess program so far written: it finished one of its games with a brilliant five-move combination.* Judging by the present game Greenblatt's program could play about board 2000 for England. Neither program seems able to form a plan that is naturally expressed by a *description* rather than by evaluation functions plus analysis. In the following game the first four moves on each side were played before the machines took over the play, because the ICL program cannot castle. The move time limits originally agreed were 90 seconds for #XESS and 'blitz speed' (5 or 10 seconds per move) for the Greenblatt program, as it was considered that the PDP10 is about 10 times faster than the ICL1909/5. After a few moves the PDP10 team increased its move time limit to about 25 seconds per move, while #XESS remained at 90 seconds per move.

1. P-KN3, P-KN3
2. N-KB3, N-KB3
3. B-N2, B-N2
4. O-O, O-O
5. N-QB3, N-QB3 (Both players weakly block their QBPs)
6. P-Q4, P-Q4
7. N-K5, Q-Q3
8. N×N

Better is 8. N-N5 which sets a trap with negligible risk since, if 8. . . ., Q-N5; 9. N×QBP, R-N1; 10. N-R6!, P×N; 11. N×N winning R for N; whereas if 8. . . ., Q-Q1; White can at least repeat position by 9. N-QB3 and presumably Black would then mechanically play Q-Q3. Then White could play 10. B-KB4 or 10. N-N5, Q-Q1; 11. P-QB4 with advantage.

8. . . ., Q×N

* See, for example, *Chess*, 32 (1967) 313.

MACHINE LEARNING AND HEURISTIC PROGRAMMING

If 8. . . ., P×N; 9. P-K4 and then, for example, 9. . . ., P×P, 10. N×P, N×N; 11. B×N, B×P; 12. B-B4, P-K4; 13. B-R6, R-Q1; 14. P-B3, B-N3; 15. B×BP1, Q-B3; 16. Q-B3, Q×Q; 17. B×Q, and White has a slight advantage. (Not here 16. B×R, R×Q; 17. QR×R, B-R6 and Black wins).

9. B-KN5 (threatens the QP), R-Q1

10. Q-Q2 (prevents P-KR3), B-K3

If 10. . . ., P-QN3; 11. B×N, B×B; 12. P-K4, P-K3; 13. P-K5, and the game has the character of a French defence difficult to evaluate.

11. P-QR3

Both programs reveal what Nimzowitch called 'the lust of the pawns to expand', once the development of the pieces is more or less completed. White did not play 11. P-QR4 since the extra square for the R, in the program's estimation, is more than counterbalanced by the loss of a square for the N. White should have prepared for P-K4.

11. . . ., P-QR4 (Better is QR-B1 with Q-K1 and P-QB4 in 'mind')

12. Q-Q3, P-R3

13. B×N, B×B

14. P-K3 (P-K4 is again double-edged), P-R4

15. P-KB4 (if 15 N-K2, B-B4; but better is QR-B1 followed by N-K2), P-KR5

16. P×P

Prevents 16. . . ., P-R6, but the pawn could be won. For example, 16. K-B2, P-R6; 17. B-B3, K-N2; 18. P-KN4, B-R5 ch.; 19. K-K2, R-R1; 20. R-KN1 with the plan 21. QR-KB1, 22. B-R1, 23. R-B3.

16. . . ., B×RP

17. K-R1, B-B3

18. Q-K2, R-R3? (better to bring the Rs to the KR file.)

19. P-QR4

The square QN5 has more value for White now that the Black QRP has advanced.

19. . . ., R-N3

20. N-N5, B-B4

21. P-B3, B-K5? (leads to a weak P at K5 and reduces his attacking chances)

22. B×B, P×B

23. P-B4, P-K3 (if B-N2; 24. P-KB5 prevents P-B4)

24. P-QB5, R-R3

25. N-B3 (good, if 26. Q-N2 had been intended), K-N2 (belated)

26. P-R3?? (even a machine should have seen 26. Q-N2), B-K2

27. P-N3?

Better is 27. Q-N2, P-B4; 28. R-KN1, Q-K1; 29. N×P, P×N; 30. Q×KP, R-B3; 31. R×P ch., Q×R; 32. R-KN1, Q×R ch.; 33. K×Q.

27. . . ., P-B4

28. K-N2, B-B3

29. QR-K1 ?, B-R5
30. R-B1, B-K2
31. R-KR1, R-KR1
32. R-QR1 ?? (only a machine could play such a move), B-B3
33. Q-QN5, R-Q1
34. Q×Q, R×Q
35. P-R4 ??, P-N3
36. P-R5, P×RP
37. R(KR1)-KN1 ?? (underestimates the value of pawns), P×P
38. K-B2 ch., K-R3
39. N-N5, P×P
40. N×P(Q4), R×N!

Slightly better than 40. . . ., B×N; since it reduces the material more.

41. P×R, B×P ch.
42. K-N3, R-B6 ch.
43. K-N2, B×R
44. K×B, R×P
45. R-QB1, P-K6 ?
46. R×P, K-N3
47. K-B1, R-N5
48. K-K2, R×P(B5)
49. K×P, R-K5 ch
50. K-Q3, P-R5
51. R-B8, R×P
52. K-B2, P-R6
53. K-B3, P-R7
54. R-KR8, R-R7
55. K-N3, R-K7
56. K-R4, K-N2
57. R-R3, R-R7 ch.
58. K-N5 ??

White had a chance of a draw with 58. K-N3 in the hope that the game would continue, for example, 58. . . ., R-K7; 59. K-R4, R-R7 ch.; 60. K-N3, R-K7; etc. For all White knows, the Black program might have no precaution against a draw by repetition.

58. . . ., P-K4
59. Resigns?

This (human) resignation seems premature since the Black program might play the late end-game very badly. There are principles in the end-game, such as the gradually decreasing net and the avoidance of a draw by stalemate, which are unimportant in the opening and middle-game. I have known a contender for the West of England championship fall into a stalemate trap.

PROSE-Parsing Recogniser Outputting Sentences in English

D.B. Vigor¹

Hoskyns Systems Research, London

D. Urquhart

International Computers Limited, Reading

and

A. Wilkinson

Computing Department
University of Glasgow

PROSE is a program which is capable of adaptively improving its linguistic behaviour by conversation with a human being. The aim of this paper is to present the sequence of events which have led to the development of this program in its present state, and to outline the model which underlies its operation.

The program itself is surprisingly quick at mimicking the linguistic behaviour of its conversational partner, considering the simplicity of this model. Only a small initial priming dictionary of a hundred words is used to start off the system, and there are only six initial relations built into the system. The program through conversation increases both its vocabulary and its range of sentence forms.

PROSE has developed to its present state from a program called GASP, or Grammatically Analysed Sentence Producer. This program was a simple demonstration program, written for amusement only, which outputs grammatically correct sentences assembled from words in a number of internal dictionaries labelled by word classes. The basic idea of GASP was that every word should call in a related word, and that sentences are somehow constructed by starting at a full stop which calls in a verb-like word, which then calls in a subject-like word and an object-like word, the subject-like word calls in an article, and so on.

¹ Present address: Hoskyns Systems Research, Boundary House, London, E.C.4.

In order to do this we assigned a subroutine for each type of word. These subroutines called in, and marshalled by the use of internal tables, other subroutines which put words into a sentence. This can be illustrated by the example in figure 1.

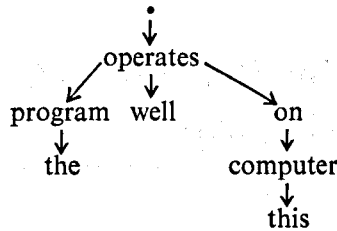


Figure 1

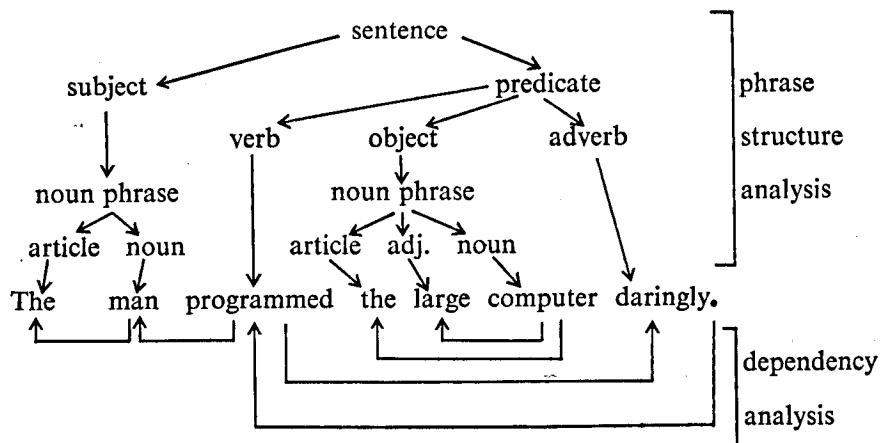
The sentence is assembled by calling in a full stop, which then calls in the verb 'operates'. This then requires to be satisfied by a subject-like word, and it calls in the noun 'program', which in order to be satisfied requires the article 'the'. We now find that 'the' needs nothing else to satisfy it, so we return to 'program'. This needs nothing else to satisfy it. We return to 'operates', which can now ask for an adverb. So we produce 'well', which requires nothing else to satisfy it, so we return to 'operates'. 'Operates' is then capable of producing a prepositional clause, so it calls in 'on' which calls in 'computer' as its noun, and 'computer' calls in 'this', an article. 'This' requires nothing else to satisfy it, so we then return to 'computer', to 'on' and then back to 'operates'. 'Operates' requires nothing more to satisfy it so we return to full stop, and output the sentence: 'The program operates well on this computer.' The fulfilment of the sentence is the tree with the full stop at the root, and 'the', 'well' and 'this' at the tips.

A sentence in GASP was a hierarchy of subroutines. Some of these subroutines had a dictionary look-up facility. This was carried out in GASP by taking a random word from a list of words which had a certain property. For example, the lists were lists of nouns, verbs, adjectives, etc. The analogy between this class of grammar and Hays' dependency grammars (Hays 1964) became obvious. We also recognised that the dependencies between the various words depended not only upon factors that one could call syntactic, but also upon semantic ones, such as 'is abstract', 'animate' or 'concrete'. These are properties of nouns which restrict the class of verbs to which those nouns can be subject.

It is very important for the understanding of this paper that one should clearly distinguish between dependency grammar and a phrase-structure grammar. Although both of these systems are formally equivalent (Gaifman 1965) they are different in that, in a phrase-structure system, constructs called phrases are postulated. A sentence is looked at as a collection of phrases. These phrases may have sub-phrases, and so on. Eventually the

words of the sentence are hung upon the tips of the tree. In a dependency system the words are actually on the internal nodes of the tree representing the sentence. An example is shown in figure 2.

A modified version of the GASP sentence producer was written by Urquhart using an improved dependency system. A number of sentences were produced using the random number generator. The sentences so generated were much better than we had expected.



The dependency tree can be rewritten:

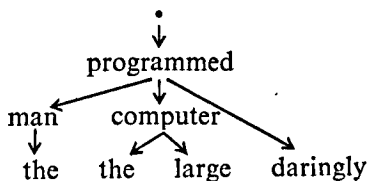


Figure 2

In parallel with this work, Bratley and Dakin (1968) had been developing a parser for English language. Using the model of their surface structure analyser, we developed a simplified parsing routine called SPUD (Sentence Parser Using Dependency). This generated the crude dependency structure for particular input sentences. We linked this with a modified GASP so that we could use the structure generated by the parser for input sentences to generate sentences with a similar dependency structure, but with words taken from an internal dictionary. We then realised that this process of analysis by outputting test sentences could be used as a test bench for studying the structure of English.

We also realised that outputting sentences would allow us to distinguish between ambiguous interpretations of the sentences analysed by SPUD. This

could be done by outputting simple sub-sentences, each one of which tested the validity of one particular dependency in the SPUD-analysed structure.

We developed a conversational system around GASP and SPUD, using the GOLD (Glasgow On Line Desks) System on the KDF9. However, the abandonment at the University of Glasgow of the GOLD System forced us to implement this system in a non-real-time environment, by making each conversational run into a set of batch-processing runs which simulated a conversation. The first run parses a batch of sentences and outputs for each sentence a set of sentences aimed at testing particular hypotheses about the input sentence's structure. An illustrative example is given in the Appendix. The second run processes any replies to the first output, and re-outputs sentences to test any newly found inexplicable rejections. From this evidence the program updates its various dictionaries. The batch method is not as effective in this application as conversation.

We built in to the system the ability to add words and sentence structures to dictionaries. Sentence structures were interpreted as the map of a stack of subroutine entries. The analysis resulted in the formation of such stacks. The polishing process labelled the sentences according to the type of dictionaries from which they could 'choose' their words. We have since changed our viewpoint, and now label words according to the local structures (dependencies) in which they can participate.

The dictionaries of both words and structures are partitioned by a learning program called the Finisher. This takes its cues from a stylised conversation with the user on the merits of particular sentences which it has output. We then realised that the sentences in this stylised conversation were amenable to analysis by the same techniques as the sentences being discussed. This increases our freedom by allowing us to use in the discussion of the structure of the sentence any stylistic trick which we have learnt through this discussion. In order to do this we only needed to introduce a few 'semantic' key words. These will set up dictionary subclasses and name them, and place markers in sentence structure maps.

GASP, SPUD and the Finisher were amalgamated by this process into a single program which we call PROSE. Since 1967 we have incorporated a number of extra features which do not detract from the intrinsic simplicity of the system, yet increase its power immensely. With the birth of PROSE we were able to start formalising a grammatical model for the processes we were carrying out. A version of PROSE is being implemented in POP-2. All this should allow a much deeper investigation of the power and usefulness of PROSE.

We recognised that there were basically only four classes of subroutines in the GASP program. These four classes of subroutines could be classified by inventing two two-valued properties. Each subroutine corresponded to a particular class of dependency. These dependencies could be classified into two groups of two, using these binary properties.

We call the two properties *binding* and *determinacy*. Binding is a measure of how much the dependency is determined by the physical position of the words concerned in the dependency. Determinacy is a measure of the rigidity of the context in which this dependency could appear. Our four classes of dependencies, each of which is associated with a particular set of GASP subroutines for assembling words related by that particular class of dependency, could be named by a combination of the values of these two properties.

The two values for binding are *bound* and *loose*. Those for determinacy are *determinate* and *recursive*. The dependency between noun and article, and between verb and personal pronoun are examples of bound determinate dependencies. The dependency between the word governing a prepositional clause and the preposition of that clause, or the relation between the verb of a main clause and the connective pronoun of a subordinate clause, are examples of loose determinate dependencies. The relation between adjectives and the noun they qualify is an example of bound recursive dependency.

The relations between a conjunction and both words which it connects are an example of loose recursive dependencies. An adverb which precedes the word it modifies has a bound determinate dependency to the word. An adverb which follows the word it modifies has loose recursive dependency. This means that there can be only one preverb which must immediately precede the verb or adjective, whereas there may be a number of adverbs which will appear after the object or indirect object. This indicates that the dependency type is not necessarily related to the grammatical function of the word. This is a strong argument, we believe, against the phrase structure model for English. Loose recursive dependencies are extremely tricky to handle in phrase structure models, but can be handled naturally in the PROSE system.

The recursive dependencies are capable of carrying two types of restriction on the word classes involved in these dependencies. These restrictions we call *negative* dependencies, and classify them as *global* or *local* negative dependencies. Such properties as plural and metaphorical are examples of global negative dependencies over a sentence structure. They restrict the classes of words which a dependency may govern. Properties such as abstract, animate and inanimate are local negative dependencies between a noun-like word and a verb-like word. If we are willing to throw away our conventional classification of grammatical classes, then we notice that the distinction between noun and adjective in a sentence is an example of local negative dependency, and that the distinction between transitive and intransitive verbs is also local negative dependency. Moving towards a semantic model for English, we see that the relation between the two clauses in an 'if . . . then . . .' or 'either . . . or . . .' construction, as represented by the dependency chain through the conjunctions 'then' and 'or', is an example of global negative dependency. PROSE has built in the facility to generate, recognise and learn new instances of each of these six dependencies.

We embodied these ideas into a functional notation which described the process of generation which was carried out by the particular subroutines. We found that we could simplify the whole of our model to five basic functions, which represented the skeleton of a sentence.

We describe these five functions in a number of equivalent ways. In the first description, A is an expression which represents a dependency subtree of the sentence.

<i>Prefix form</i>	<i>Infix form</i>
1. $f(A_1, A_2)$	$A_1 f A_2$
2. $p(A_1, A_2)$	$A_1 p A_2$
3. $o(A)$	$o A$
4. $n(A)$	$n A$, where n is an integer ≥ 1
5. $e(A_1, A_2)$	$A_1 e A_2$

f means expression is followed by expression.

p means expression is preceded by expression.

o means optional.

$\langle n \rangle$, where n is an integer ≥ 1 , means up to n occurrences.

e means 'exclusive or'.

o could be included as a subfunction of n by a change of definition, but this leads to complications in the learning process. It should also be noted that the arguments of p and f do not commute. p and f differ from simple concatenation in that they are non-associative.

A Backus Normal Form description of the use of these functions is:

$$\langle \text{expression} \rangle ::= \langle \text{name} \rangle \mid (\langle \text{expression} \rangle \langle \text{function 2} \rangle \langle \text{expression} \rangle) \mid \langle \text{function 1} \rangle (\langle \text{expression} \rangle)$$

$$\langle \text{function 2} \rangle ::= p \mid f \mid e$$

$$\langle \text{function 1} \rangle ::= o \mid n$$

$$\langle n \rangle ::= 1 \mid 2 \mid 3 \mid \dots$$

$$\langle \text{name} \rangle ::= \langle \text{terminal symbol} \rangle$$

An example expressed in infix bracketed notation of a description of a sentence is given below. The sentence is of the form:

subject verb object

(with three adjectives allowed before a noun, and an optional adverb allowed either before the verb or after the object). To generate a sentence in the sentence producer such an expression is interpreted directly from the bracket-free prefix form. We assume that the program is at this stage capable of recognising the following classes of words ($\langle \text{terminal symbol} \rangle$ s):

T transitive verb

N noun

Q general quantifier (adjective)

D noun determiner (article)

M modifier (adverb)

. full stop

$$\cdot p(Tf((Np(o(3(Q)))) p(o(D))) o(pMefM) \\ p((Np(o(3(Q))) p(o(D)))));$$

In this form of the notation we can clearly see the correspondence between the bracketed entities and phrases in a phrase structure model. The dependency tree is always rooted at the full stop, and sentences are generated by the yo-yo or depth-first method of tree searching. Each partial flip of the yo-yo corresponds in some sense to the generation of a phrase in a phrase structure grammar.

In the sentence recogniser part of PROSE we synthesise a model of the sentence from the determiners in all directions along the branches of the tree. In a fully determined sentence, for example 'The girl likes the boy.' (see Appendix), the parser acts similarly to a phrase structure parser which constructs its tree simultaneously from top to bottom and from bottom to top until the two parts of the tree meet. In non-fully determined or ambiguous sentences there is apparently no immediate analogy between our representation and that of phrase structure.

In order to illustrate the relation between the notation and the operation of the sentence generator and parser, we will adopt the following notation in all figures. A dependency will be represented by an arrow directed from the governing word to the governed word. On this arrow will be an ordered pair of integers (r, g). $r=1$ for the dependency first recognised by the parser, $r=2$ for the second one, and so on. $g=1$ for the first dependency called by the generator, $g=2$ for the second one, and so on.

In PROSE the order for generation of dependencies is fixed by the program structure, but the order for recognition by the parser is dependent on the state of learning of the system. As the program becomes capable of recognising more words as determiners, a larger proportion of dependency assignment takes place in the first pass of the parser.

It is possible for us to imagine each dependency to be divided into two parts, one attaching to the word which governs the dependency, and the second to the word which is governed. We can imagine that on the end of each one of these dependency halves we have some sort of plug, which is typical of the type of dependency we are considering. For bound dependencies the plug can be thought of as being attached to the word itself, whereas for loose dependencies the plug may be considered to be attached to the word by a flexible lead. The plug will carry markers denoting the properties of the words to which this plug may be attached.

The process of parsing is a process of setting up a dependency tree by plugging in plugs, until all plugs are satisfied. For example, 'the' has a plug which can only be satisfied by a plug from a noun. A full stop has a plug

which requires a verb to satisfy it. 'Sits' has a plug which requires a singular noun or pronoun to precede it, and an optional plug for a positional preposition to follow it. We may, during learning, find that the singular noun must also have a property or negative dependency called *animate*, to satisfy the plug on 'sits'. The learning program operates by attaching to words in the dictionary a property list, structured into plugs which must be satisfied (subroutine calls) and negative dependencies associated with each plug (arguments to the above subroutines).

The generator operates by choosing words which satisfy the existing unsatisfied plugs. To start generation we simply produce a full stop. The negative dependencies on plugs may be either local or global. Local ones only apply to that part of the sentence tree which hangs on this plug. Global negative dependencies apply to the whole sentence tree.

This model justifies our learning mechanism, which only learns by attaching properties to words. These properties seem sufficient to specify sentence structure. Once a property has been established as belonging to more than three words (in the present implementation) then these words are factored out into a subclass of the dictionary in which they appeared previously. This means that the dictionaries are arranged hierarchically, and the learning process sets up the hierarchy. The words in dictionaries at the top of this hierarchy can be used in more general contexts than those lower down. The total dictionary is thought of as one such tree, with nodes of two types, *or-exclusive* nodes and *and* nodes, where *or-exclusive* means that the properties in the subtree hanging on this node are disjoint, and on the *and* nodes the properties of the node apply to all lower branches. For example, verbs and nouns are separated by an *or* node because they require completely disjoint contexts (positive dependencies). Concrete nouns and abstract nouns are on an *and* node.

In the parser, when we form a dependency structure of the sentence, we place the words tentatively into a specific dictionary inside this dictionary hierarchy. If a sentence is rejected, we move down the tree creating a node if none is available, and assign the value of this node as a property to the word; that is we place a tentative restriction on the use of the word. If the word is accepted, we may try to test it by moving up the tree to the node above the one at which it was last accepted. For example, 'fish', accepted as a noun in the first instance, may then be tested as a verb.

Although the learning process could proceed 'automatically' by using this simple trial and error method, it is intolerably wasteful in terms of the number of bad sentences output. In PROSE, therefore, we allow the conversational partner (human being) to prompt the system by doing a limited amount of manipulation on the dictionary tree. This is done by introducing the meta-linguistic determiners *isa* and *isp*. These are recognised by the parser as closed class verbs, which, when recognised, call in a subroutine to manipulate the dictionary entries.

Isa is a verb which may appear in a sentence such as 'Sat isa past participle.', if no past participles have been seen before. This sets up a new dictionary whose name will be *pastpart*, that is, the first eight characters of the name. This allows learning to be speeded up by introducing new classes of words when the conversational partner thinks it is useful. Past participles will have initially the dependencies which 'sat' had. If this is not the first past participle seen, it will add the dependencies of 'sat' to those of the subjects of previous isa past participle statements. We can also, by the use of the character '-' denote a suffix which can be used to distinguish a class. For example, we could make the statement '-ed isa past participle.' and words ending in -ed would be classified that way. In this case the past tense of certain verbs would be classified as an *or* instance of past participle when they occurred. Another example is '-ly isa adverb.' or '-ing isa present participle.' Both present and past participles also end up in the system as subclasses of adjectives on an *or* node. The same physical dictionary of past participles would be accessed *via* both adjectives and verbs.

Isp sets up a marker on the word which appears as its subject. This marker is used to denote a negative dependency which is assumed to be local until proved otherwise by the rejection of a sentence in which the word appears. This marker defines a subclass of the class to which the subject of **isp** belongs. It is represented by a bit on the plug which is assigned to this property. For example, 'Men isp plural.', 'Man isp animate.', 'On isp positional.' The **isp** property is assigned to both the word which appears in the **isp** statement and that word on which it depended in the statement being discussed previously, to which the **isp** statement is an answer. The subject of **isp** should be chosen as the governed class in the dependency. If we had had a sentence such as 'The boys like girls.', we could possibly obtain as an output sentence 'The girls sits.' We could use this to make a general statement about nouns: '-s isp plural.' The conversation is referring to 'girls', as 'sits' did not appear in the original sentence. Therefore, the general rule 'Nouns with -s at the end are plural' can be sneaked in in this way. The rule has a number of exceptions and will result in bad analyses in later sentences, to be refined by further conversation concerning this property 'plural'.

We have been studying the possibility of applying a dependency model of word structure to generate new words. A tentative stand-alone program has been written which processes the dictionary of PROSE and produces an enlarged one by recognising and removing prefixes and suffixes from words.

The dictionary is sorted alphabetically. All initial sequences of characters which appear in two or more words are removed from all but the first of these words. The blank spaces at the start of each word are given a count value which is the number of characters which have been blanked out. If more than five consecutive words have the same count, then the remaining characters of each word with this property are tested in the same part of speech as the word from which it was derived. If an acceptable word has been

generated, the count and the letters are entered into a prefix dictionary, and the word less the prefix is entered on to the end of the main dictionary. If an acceptable word has not been generated and the word totally rejected, then this particular prefix is labelled fallacious, and its last character is returned to the stem. The test process is then repeated.

The dictionary words are then inverted (turned back to front) and sorted into another area of store, and the same process is used to recognise suffixes. This gives a rhyme dictionary which may be useful for generating poetry.

The next stage of the process to be implemented will be to use the prefix suffix dictionary to construct new words for PROSE. Suffixes can change the part of speech, for example '-e' in a verb to '-ation' in a noun, addition of '-ly' generates adverbs. The meta verbs *isa* and *isp* will offer the ability to recognise and learn such transformations.

At the sentence level the model is symmetric in the processes of analysis and generation, in that it can recognise any sentence that it could generate. At the word level the learning at present takes place only in the prefix, suffix and stem recogniser. The symmetry could be retained by the introduction of this transformational part. This could be used to investigate how new semantically suggestive words could be or have been generated in English. We can generate recognisable adjectives by straight addition or replacement of a terminal *-e* on a noun or verb by *-al*, *-y*, *-ive*, *-ful* or *-able*. We could extend the use of a Latin stem by use of *re-*, *in-*, *pre-*, *pro-*, *inter-*, and so on, or of Greek stems by use of *epi-*, *endo-* or *-ic*. Who in the jungle of present mathematical terminology would argue with a learned paper on the interproduction of epimorphistic conformability?

In the present version of PROSE the starting dictionary is of approximately one hundred selected determiners. This is sufficient to give adequate initial power to allow efficient bootstrapping of more words and structures into the PROSE memory. However, we do not claim that this is in any way a minimal dictionary. Provided that care was taken in choosing the initial sentences, one would require many less initial words with well-defined dependencies. In compiling this initial dictionary we were guided by the results of Thorne, Bratley and Dewar (1968).

What we have been describing is a performance model for recognition and generation of sentences in an ill-defined subset of English. Such a program as PROSE could be used by allowing it to converse with a linguist to develop a competence model for English based on dependency structure.

There is evidence, based on our limited experience of using PROSE, that a number of problems facing phrase-structure grammar users can be quite simply overcome by using a dependency model. One such difficulty arises when dealing with conjunctions which are handled in our system by loose recursive dependencies. However, we do not have enough experience as yet to be able to assess the full potential of our model.

To check the present state of learning in the system, we use the sentence

generator to produce, with the aid of a random number generator, a sample of the linguistic ability of the system. The following few sentences, taken from such an output after the system had ingested and discussed thirty sentences, chosen from *Punch*, *Computers and Thought* and *The Organisation Man*, illustrate the power of the system even in its present embryonic state:

'Reports stretch simply.'

'A state generates their money.'

'If its class touches the connection then it, their magical diabolical circuit, exterminates this material.'

'The director has created the society thru a child.'

'Will the programmer retire?'

'Annihilate all books.'

It appears also that the essence of style and semantic context can be abstracted by the use of the conversational mode of PROSE. The system organises itself by setting up relation markers (negative dependencies), which means that it uses the vocabulary in sentences of similar structure to those from which the words were taken. This can be seen to a limited extent in the preceding examples.

The dependency relations in a sentence are capable of carrying information about the universe which is being discussed in that sentence. A very tentative initial experiment in rejecting sentences which were not contextually meaningful indicates that the model is capable of learning over a much larger context than a single sentence. This is done by introducing interdependencies which carry context markers (global negative dependencies) and which assemble sentences into paragraphs.

The method of parsing seems very effective for a small subset of English. Nouns can correspond to files representing sets of objects; adjectives and prepositional clauses can correspond in statements to predicates on these sets, and in questions to selectors (Vigor 1968). Verbs then correspond to procedures, adverbs to procedure typing, conjunctions to logical connectives, and subordination to subroutine call. The dependency model of syntax adds a clarity and simplicity to the structuring of the requests and updating information in the system.

The possibility of applying this to a real life situation is being investigated by Vigor for parsing both questions and input data for a natural language information retrieval system on large commercial data bases (Hoskyns Systems Research 1968).

APPENDIX

1. Example of operation of parsing routine.

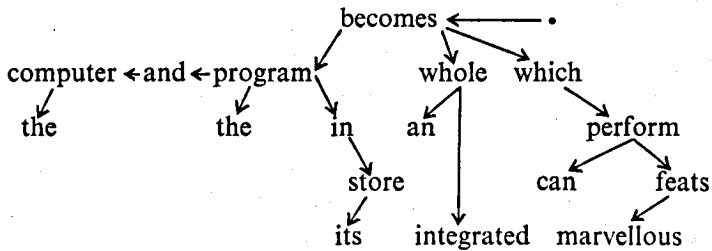
'The computer and the program in its store become an integrated whole which can perform marvellous feats.'

Information at the end of the first pass:

MACHINE LEARNING AND HEURISTIC PROGRAMMING

The	noun determiner
computer	substrate noun
and	conjunction
the	noun determiner
program	substrate noun
in	preposition
its	closed class noun determiner
store	substrate noun
become	verb
an	noun determiner
integrated	verb or adjective
whole	noun
which	closed class subordination
can	closed class auxiliary verb
perform	verb determined by can
marvellous	noun, adjective or adverb
feats	plural noun or singular verb.

Information at the end of the second pass:



Words in doubt are 'marvellous' and 'feats'.

Also to be checked are 'integrated' and 'whole'.

We must also confirm that 'program' and 'computer' are of the same class.

The sentence producer is called with the following words to be tested:

'feat, whole, program, computer' as nouns,

'marvellous, integrated, whole, feats' as verbs or adjectives.

The latter class will also be tested as complements.

Examples of output

- 'The feat is performed.'
- 'The feat is marvellous.'
- 'The feat sits.'
- 'The whole becomes the computer.'
- 'The whole is integrated.'
- 'The computer and the program perform.'
- 'The whole computer performed the feat.'
- 'The whole is marvellous.'

And so on

Sample reply

- 'Right'
- 'Right'
- 'Feat is abstract.'
- 'Whole is abstract.'
- 'Right'
- 'Check'
- 'Right'
- 'Right'

And so on

In conversation a single word of reply signifies approval of the output sentence.

Third pass (Finisher):

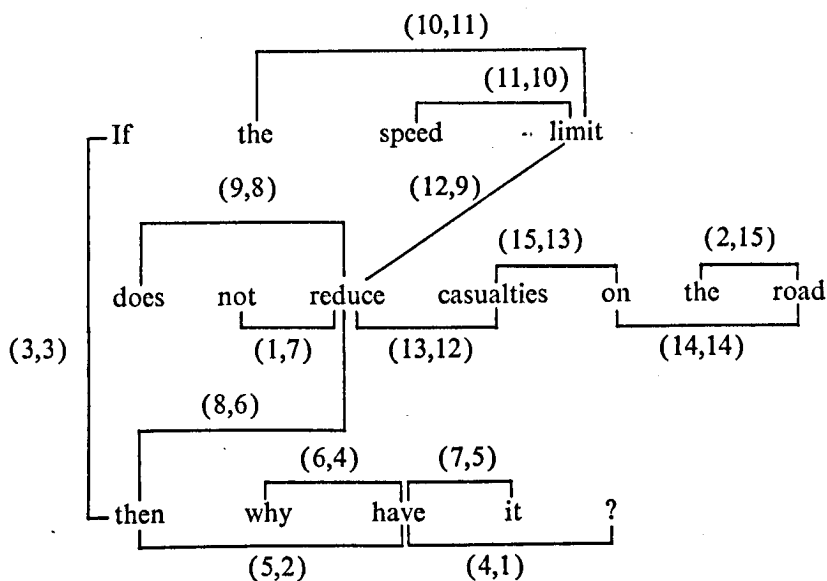
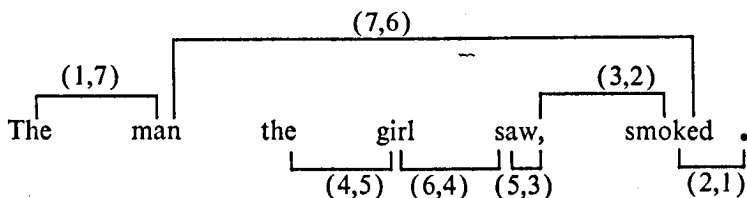
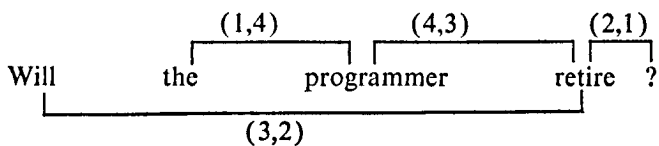
From the results of the conversation the dictionary will be updated by

1. the unequivocal words in the initial sentence,
2. the dubious words from the context and remarks made during the above conversation.

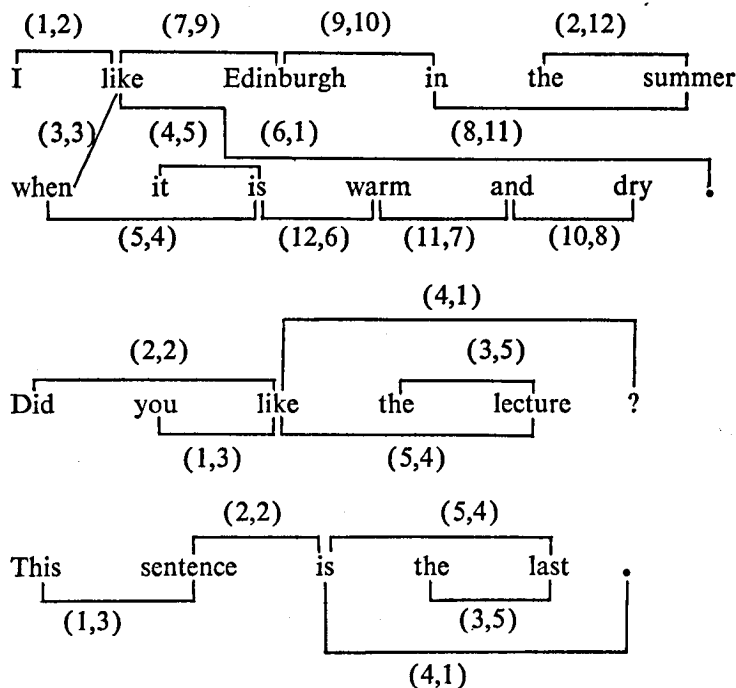
From the two abstract corrections we would have the output:

'The whole is performed.' Reply: 'Right'

2. Examples of PROSE dependency trees on some simple sentences:



MACHINE LEARNING AND HEURISTIC PROGRAMMING



REFERENCES

Bratley, P. & Dakin, D.J. (1968) A limited dictionary for syntactic analysis. *Machine Intelligence 2*, pp. 173-81 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.

Gaifman, H. (1965) Dependency and phrase structure systems. *Information and Control*, 8, 308-37.

Hays, D.G. (1964) Dependency theory - a formalism and some observations. *Language*, 40, No. 4, 511-25.

Hoskyns Systems Research (1968) (Details of software projects available on request from A.E.L. de Watteville, H.S.R., Boundary House, London, E.C.4.).

Thorne, J.P., Bratley, P. & Dewar, H. (1968) The syntactic analysis of English by machines. *Machine Intelligence 3*, pp. 281-309 (ed. Michie, D.) Edinburgh: Edinburgh University Press.

Vigor, D.B. (1968) Data representation - a key to conceptualisation. *Machine Intelligence 2*, pp. 33-44 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.

The Organization of Interaction in Collectives of Automata

V. I. Varshavsky

Leningrad branch of the Central Economic-Mathematical Institute

The study of the collective behaviour of automata, originated and directed by M. L. Tsetlin and continued after his premature death by his pupils, has made it possible to construct a number of interesting models and to develop an effective approach to the study of control processes in complex systems (Tsetlin 1963, Tsetlin and Varshavsky 1965, Tsetlin and Varshavsky 1966, Varshavsky 1968). The first models were developed to study regularities of the behaviour of a collective of autonomous objects (automata) whose 'interaction' derives entirely from the effects of the external surroundings on their joint behaviour. Thus each automaton is ignorant both of the problem to be solved by the collective and of the number of members in the collective. Some interesting and significant results were obtained in this direction. But even during Tsetlin's lifetime models were developed in which the above restrictions had to be abandoned. These were the model of the distribution of 'computation means' (Ginzburg and Tsetlin 1965) and the problem of the organization of behaviour in a periodic random environment (Varshavsky, Meleshina and Tsetlin 1965). Further development of this line of investigation has forced us to pay serious attention to the solution of the internal problems for a collective, that is the organization of interaction and exchange of information between the automata. Here we shall consider some very simple models of the organization of such interaction.

We shall consider a Goore game of N identical automata (Borovikov and Bryzgalov 1965). Following Tsetlin (1963), we define a game of N automata. Let $s^j(t)$ and $f^j(t)$ be the values of the input and output variables of the automaton A^j at time t . $s^j(t)$ is assumed to take only two values, $s^j(t)=0$ and $s^j(t)=1$, corresponding to (unit) gain and loss respectively by the automaton A^j at time t . An output variable $f^j(t)$ takes values from the set f_1^j, \dots, f_k^j . These values are called strategies of A^j , and if $f^j(t)=f_a^j$, then automaton A^j is said to use its a th strategy. A set of strategies

$f(t) = (f^1(t), \dots, f^N(t))$ used at time t by the automata A^1, \dots, A^N is called a play played at time t . The outcome $s(t+1)$ of a play $f(t)$ is the set $s(t+1) = (s^1(t+1), \dots, s^N(t+1))$ of values of the input variables (unit gains and losses) of the automata at time $t+1$. Automata A^1, \dots, A^N take part in a game if for each play $f(t)$ the probability $P(f(t), s(t+1))$ of the outcome $s(t+1)$ is given, and the equation $\sum_s P(f(t), s(t+1)) = 1$ holds for any f .

The game consists of a sequence of plays.

The game Γ is called a game with independent outcomes if

$$P(f, s) = P(f, s^1, \dots, s^N) = \prod_{j=1}^N P(f, s^j).$$

A Goore game is a game of N automata with independent outcomes in which

$$f^j = \{0, 1\} \quad (j=1, \dots, N)$$

and
$$P(f, s^j = 1) = P\left(\sum_{j=1}^N f^j / N\right) = P(a) \quad (j=1, \dots, N).$$

Thus in each play of a Goore game the probabilities of winning are the same for all players and depend only on the fraction $a (0 \leq a \leq 1)$ of automata choosing strategy 1. Essentially, a Goore game can be interpreted as an example of collective behaviour whose expediency is determined by a rational distribution of 'effort'. For simplicity, the function $P(a)$ is assumed to have only one minimum at a point a_0 . Borovikov and Bryzgalov (1965) showed that the probability distribution of a tended to a δ -function at the point a_0 as the memory capacity of the automata increased. On the other hand, Volkonskii (1965) and Pittel' (1965) established that with a fixed memory capacity n , the quality of the performance of the collective deteriorates as N increases, and as $N \rightarrow \infty$, the distribution of a tends to a normal distribution with mean at $a = \frac{1}{2}$. To maintain the quality of performance as N increases, n must increase at least as fast as $N^{1+\epsilon}$.

This example shows that in a collective without internal interaction the struggle against entropy requires that the individual resources of its members increase as the size of the collective increases. Strictly speaking, in a Goore game we have to deal not with a collective of automata, but a 'crowd'. However, introduction of the simplest type of interaction between the automata leads to improvement of the behaviour characteristics. For this, we shall construct a game on a circle (Gel'fand, Pyatetskii-Shapiro and Tsetlin 1963). Let the automata in a Goore game be situated on a circle, that is, each automaton A^j has two neighbours¹ A^{j-1} and A^{j+1} .

Now let s^j be the gain or loss of the j th automaton in a Goore game, \tilde{s}^j be the input signal for the j th automaton and

$$\tilde{s}^j(t+1) = \tilde{f}^{j-1}(t) \cdot f^j(t) \cdot \tilde{f}^{j+1}(t) \vee f^{j-1}(t) \cdot \tilde{f}^j(t) \cdot f^{j+1}(t) \vee s^j(t+1)[f^{j-1}(t) \oplus f^{j+1}(t)],$$

¹ Addition and subtraction of indices are performed modulo N .

that is, the j th automaton loses if its action differs from the actions of its neighbours, wins if its action coincides with those of its neighbours and acts according to the outcome of the play if the actions of its neighbours are different: $f^{j-1} \neq f^{j+1}$. Such interaction causes automata to try to assemble into groups with 'community of interests' so that the only automata taking part in the game are virtually those at the ends of a group.¹ Figure 1 gives experimental curves for the distribution of a in a Goore game of 32 automata with linear tactics for $n=2, 3, 4, 5$; the left-hand column is without interaction and the right-hand column with interaction in a game with 2^{16} plays. The effect of interaction is evident.

Both in the above example and in other examples of collective behaviour in a random environment automata solve the problem of selection against a background of noise. Introducing interaction, as in the Goore game, makes it possible to organize in some sense a directed search. On the other hand there are problems in which randomness of environment and the influence of noise are not dominating factors. An example of such a problem is the organization of collective behaviour for achieving a one-dimensional allocation of resources (Varshavsky 1968, Varshavsky, Meleshina and Perekrest, in press). We usually meet such situations when attempting to organize collective behaviour in control problems. The organization of collective behaviour is achieved rather easily if the effect of collective behaviour is the sum of the effects of the functioning of the separate parts of the system under certain restrictions, as in the problem of allocation of resources,² or if the problem admits a more or less obvious decomposition into sub-problems ('sub-games' in Michie and Chambers 1968). In other cases we have to study methods for organizing interaction between subsystems and assessing ways of exchanging information. In this case gradient methods are often suitable. On the one hand they make it possible to organize local behaviour, and on the other hand the form of the partial derivatives of the minimized (maximized) function often makes it possible to find ways of organizing interaction for local estimation of the current value of the partial derivative. In recent years a number of successful attempts have been made to use gradient methods for the solution of a wide class of problems including linear and nonlinear programming problems (Arrow *et al.* 1958). Problems like this arise in connection with the necessity of organizing control in systems for which a purposeful collective behaviour seems to be most natural. From this point of view it seems to be of interest to give here some preliminary considerations of the possibility of using a gradient approach to the travelling salesman problem.

¹ Naturally the plays $f(t) = (0, \dots, 0)$ and $f(t) = (1, \dots, 1)$ are stable, but it is easy to exclude the possibility of arriving at these situations.

² Such an approach can be trivially extended to the case when the overall performance-criterion of the system is the product of the performance-criteria of the sub-systems – for example, when the probability of a correct observation is the product of the probabilities of correct solution of the problem by the sub-systems and observation time is limited.

MACHINE LEARNING AND HEURISTIC PROGRAMMING

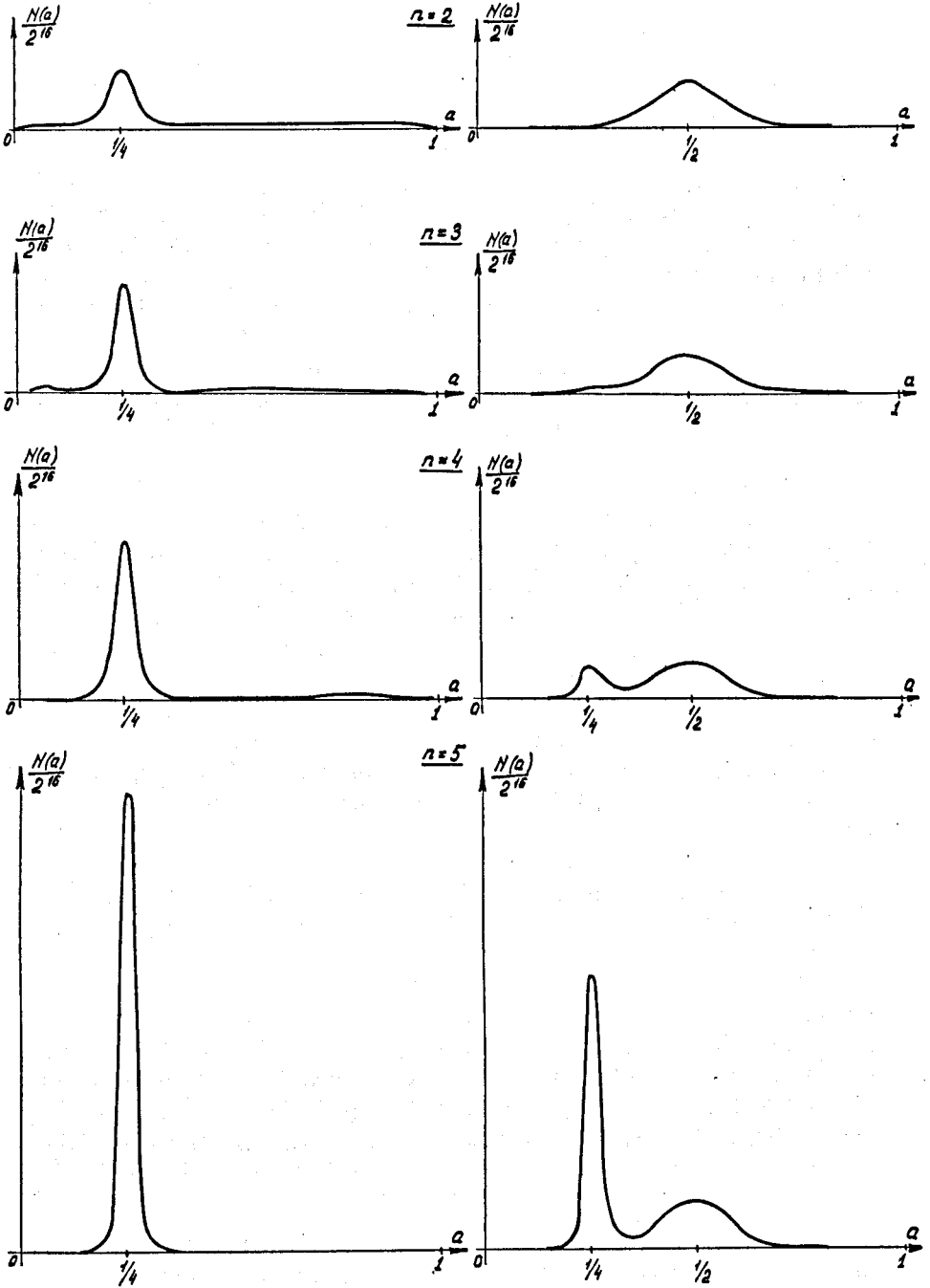


Figure 1

We shall consider the travelling salesman problem in the following form. There are n points all connected to each other. The path joining the i th point to the j th point has length r_{ij} satisfying the triangle inequality $r_{ik} + r_{kj} \geq r_{ij}$. We want to find a closed path of minimal length passing through all the points.

Under the above restrictions a closed path of minimal length is a Hamiltonian path, that is, it passes through each point once and only once. Any path of this kind can be specified by a cyclic permutation matrix. Thus the solution of the travelling salesman problem reduces to finding a cyclic permutation matrix $C = \|c_{ij}\|$ which minimizes $\sum_i \sum_j c_{ij} r_{ij}$.

Note that permutation matrices are extreme points of the space of bistochastic matrices and so it is reasonable to try to organize a continuous search in the space of bistochastic matrices. We used a similar method in the search for transition matrices of finite automata with expedient behaviour in random environments by employing stochastic automata with variable structure (Varshavsky, Vorontsova and Tsetlin 1962, Varshavsky and Vorontsova 1963, Varshavsky and Vorontsova 1964).

Let m_{ij} be the probability that the point i is connected to the point j , and let the matrix $M = \|m_{ij}\|$ describe the current state of the connections in the system. If M is stochastic then the elements of $M^k = \|m_{ij}^{(k)}\|$ are the probabilities that a path starting at the point i reaches the point j in k steps. Hence the condition $M^n = E$, where E is the unit matrix and n is the dimension of M , is the condition that for any i a path beginning at the point i returns to the point i after n steps.

It is not difficult to see that if n is a prime number, $m_{ii} = 0$ and $\sum_{j=1}^n m_{ij} - 1 = 0$ ($m_{ij} \geq 0$), then the condition $M^n = E$ implies that M is a cyclic permutation matrix. Thus the travelling salesman problem can be stated for prime n as follows:

Find

$$\min \sum_i \sum_j m_{ij} r_{ij}$$

under the conditions:

$$1. \sum_{j=1}^n m_{ij} - 1 = 0; \quad m_{ii} = 0; \quad m_{ij} \geq 0.$$

$$2. \text{Sp } M^n = \sum_{i=1}^n m_{ii}^{(n)} = n, \text{ that is, } \text{Sp } M^n - n = 0.$$

Generally speaking, in this presentation the problem can already be solved by gradient methods using Lagrange multipliers. But here we are faced with a number of difficulties which accompany gradient methods for solving the problem of minimizing a linear functional under certain restrictions (Arrow, Hurwicz and Uzawa 1958). To avoid some of these difficulties we introduce a nonlinear functional.

Consider the matrix $L = \|l_{ij}\|$, where $l_{ij} = \exp(-r_{ij})m_{ij}$; we shall maximize $\text{Sp } L^n$ under the above restrictions.

Consider the function $Q = \text{Sp } L^n - \lambda(\text{Sp } M^n - n)$ and its partial derivatives with respect to elements of M .

$$\begin{aligned} \frac{\partial Q}{\partial m_{ij}} &= \frac{\partial}{\partial m_{ij}} \text{Sp } L^n - \lambda \frac{\partial}{\partial m_{ij}} \text{Sp } M^n = \text{Sp } \frac{\partial L^n}{\partial m_{ij}} - \lambda \text{Sp } \frac{\partial M^n}{\partial m_{ij}}, \\ \frac{\partial M^n}{\partial m_{ij}} &= \frac{\partial}{\partial m_{ij}} (M^{n-1}M) = M^{n-1}E_{ij} + \frac{\partial M^{n-1}}{\partial m_{ij}}M \\ &= \sum_{s=0}^{n-1} M^{n-1-s}E_{ij}M^s, \end{aligned}$$

where $E_{ij} = \frac{\partial M}{\partial m_{ij}}$

is a matrix in which $e_{ij} = 1$ and all the other elements are equal to 0. It is not difficult to see that

$$\text{Sp } \frac{\partial M^n}{\partial m_{ij}} = nm_{ji}^{(n-1)},$$

and similarly

$$\text{Sp } \frac{\partial L^n}{\partial m_{ij}} = n \exp(-r_{ij})l_{ji}^{(n-1)}.$$

Hence
$$\frac{\partial Q}{\partial m_{ij}} = n[\exp(-r_{ij})l_{ji}^{(n-1)} - \lambda m_{ji}^{(n-1)}],$$

where the Lagrange multiplier λ represents $\exp(-s)$, s being the length of the shortest path.

These considerations motivate a search for new computational procedures and principles of collective behaviour for the solution of the travelling salesman problem. It should be emphasized that we have given here only preliminary considerations about one possible approach to this and a number of similar problems. The search for effective methods for solving the problem by applying the principles above presents several difficulties which have yet to be overcome.

From the point of view of studying the interaction of automata, the models of purely logical interaction developed by J. von Neumann (1966), Ulam (1962) Burks (1964) and others are of particular interest. Recently much work has been published on the so-called 'firing squad synchronization problem' (Moore 1964, Levenstein 1965, Waksman 1966, Balzer 1967, Varshavsky 1968, Varshavsky, Peschanskii and Marakhovskii 1968). We shall consider some variants of this problem in more detail.

The problem of synchronizing a chain of automata posed by J. Myhill has given rise to several models of particular interest. We modify the statement of the problem as follows. Is there a finite automaton A such that a chain of n automata A can be synchronized at time T after being 'switched on' by a starting signal applied to an arbitrarily chosen automaton at time $t=0$?¹ Each automaton is assumed to be connected to its immediate neighbours and the complexity of each automaton is independent of n .

Here synchronization means that all the automata in the chain simultaneously pass into a state called the synchronized or terminal state at time T , and each automaton reaches this state only at time T . We consider Moore automata in which the internal states are the output signals. We first give a general idea of the solution of the problem in the original form (Levenstein 1965, Varshavsky 1968). The basic idea is to arrange for successive bisections of segments of the chain of automata. Consider figure 2. The first bisection is carried out as follows. The starting signal puts the end automaton into the preterminal state and two signals p_1 and p_3 start to travel down the chain from this automaton.

The first signal travels with velocity 1 and the second with velocity $1/3$ (a signal travels with velocity $1/m$ if it passes to the next automaton after having stayed in the preceding one for m time units). The signal p_1 reaches the end of the chain, puts the end automaton into the preterminal state, and returns with the same velocity. The reflected signal meets the signal p_3 at the centre of the chain and the corresponding automaton (or two, if the number of automata in the chain is even) passes into the preterminal state. If the reflected signal continues to travel down the chain with velocity 1 and if the first automaton emits a signal at $t=0$ with velocity $1/7$ (signal p_7), these signals will meet at a distance of $1/4$ from the beginning of the chain. If in addition each automaton emits a sequence of signals with velocities $1/(2^{m+1}-1)$ when it enters the preterminal state and if the automata at the meeting point of the signals pass into the preterminal state, then a sequence of successive bisections of segments of the chain will be produced, as shown in figure 2.

Now suppose that the starting signal is applied to an arbitrary automaton in the chain. The general picture of the propagation of the signals is shown in figure 3. After the starting signal has been applied, two signals p_1 and p'_1 start to travel from the initial automaton in opposite directions. Both signals have velocity 1 (the initial automaton does not pass into the preterminal state unless it is an end automaton). When the signals p_1 and p'_1 reach the ends of the chain they put the end automata into the preterminal state and give rise to reflected signals with the same velocity. As stated above, an automaton which has been put into the preterminal state begins to generate a sequence of signals with velocities $1/(2^{m+1}-1)$.

If the starting signal had been applied to the automaton O at the end of the

¹ In Myhill's problem the starting signal is applied to the automaton at one end of the chain.

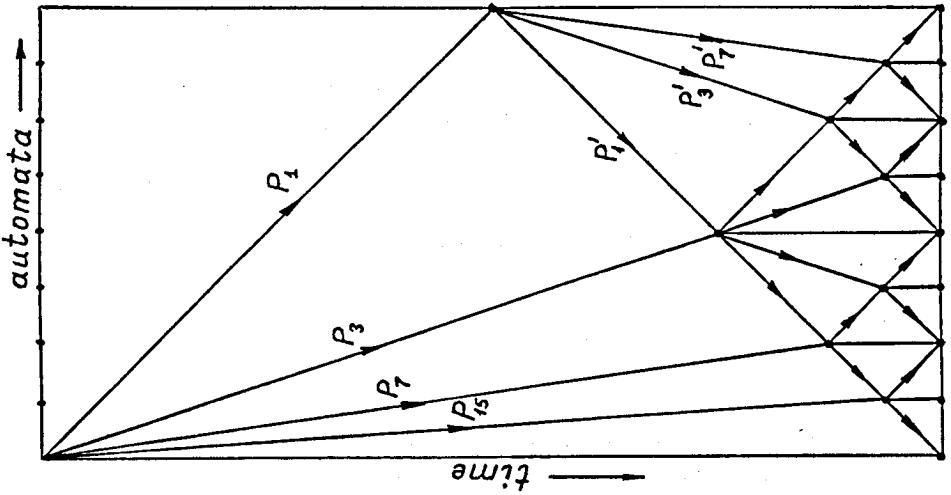


Figure 2

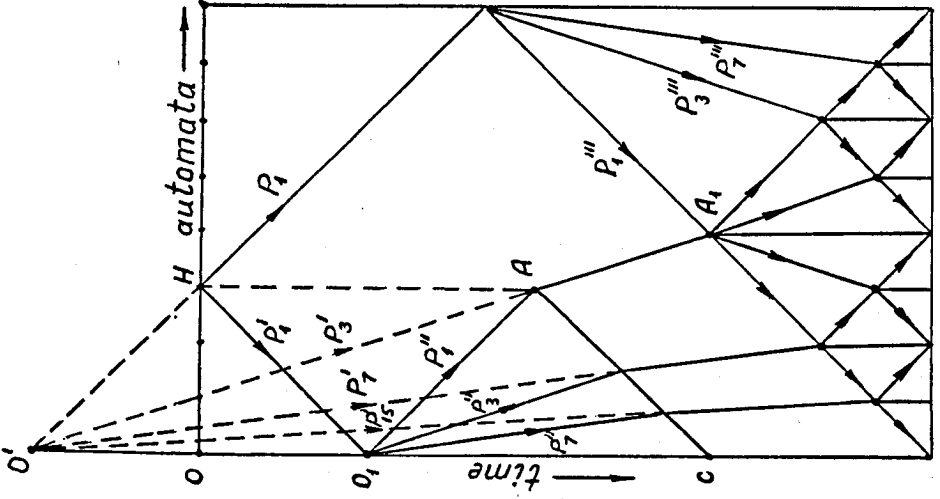


Figure 3

number	rule	condition
1	x A z	$x=A; z \in \{A, \vec{D}, E_1, E_2\}$ or $x \in \{\vec{C}, E_1\}; z = \vec{D}$ or
	A	$x=D; z = \vec{D}$ or $x=E_1; z=E_2$ or $x=E_2; z=E_2$
2	x \vec{C} z	$x=A; z \in \{\vec{D}, \vec{D}\}$ or $x \in \{E_1, E_2\}; z = \vec{D}$
	A	
3	x \vec{D} z	$x \in \{B, \vec{C}, E_1\}; z=A$ or $x=E_2; z = \vec{C}$ or
	A	$x=A; z \in \{A, \vec{C}\}$
4	B B B	
5	A	
	x E_1 \vec{D}	$x \in \{A, E_2\}$
6	A	
	A y z	$y=B; z=A$ or $y=A; z=B$
7	B	
	x A z	$x \in \{\vec{C}, R\}; z \in \{A, \vec{D}, E_1, E_2\}$
8	\vec{C}	
	\vec{C} \vec{C} z	$z \in \{A, E_1, E_2\}$
9	\vec{C}	
	x \vec{D} z	$x \in \{\vec{C}, R\}; z \in \{A, \vec{C}, \vec{D}, \vec{D}\}$
10	\vec{C}	
	x \vec{D} z	$x \in \{\vec{C}, \vec{D}\}; z = \vec{D}$ or $x=R; z=A$

Table 1

MACHINE LEARNING AND HEURISTIC PROGRAMMING

number	rule			condition
11	R	B	B	
		\vec{C}		
12	x	A	z	$x = \vec{C}; z \in \{\vec{C}, \vec{C}\}$ or
		R		$x = B; z = B$
13	x	\vec{C}	z	$x \in \{R, \vec{D}\}; z = E_2$ or
		R		$x = \vec{D}; z = E_1$
14	x	R	z	$x = \vec{C}; z = \vec{C}$ or $x = B; z = B$ or
		R		$x = R; z \in \{A, \vec{C}, \vec{D}, E_2\}$ or
		R		$x \in \{A, \vec{D}\}; z \in \{A, \vec{D}\}$ or $x = \vec{D}; z \in \{A, \vec{D}, \vec{D}\}$ or $x = E_2; z = E_2$
15	x	y	z	$x = \vec{C}; y = \vec{D}; z \in \{E_1, E_2\}$ or
		R		$x = R; y \in \{B, \vec{D}, \vec{D}, E_2\}; z = R$
16	\vec{C}	y	z	$y = E_1; z \in \{A, \vec{D}, E_2\}$ or
		R		$y = E_2; z \in \{A, R\}$
17	A	B	z	$z \in \{B, \vec{D}\}$
		\vec{D}		
18	x	A	z	$x \in \{\vec{C}, \vec{D}\}; z \in \{A, E_1, E_2\}$ or
		\vec{D}		$x = \vec{D}; z = \vec{D}$
19	x	\vec{C}	z	$x \in \{A, \vec{C}, \vec{D}, E_1\}; z \in \{R, \vec{D}\}$
		\vec{D}		
20	\vec{D}	\vec{C}	z	$z \in \{A, \vec{C}, \vec{D}, E_2\}$
		\vec{D}		

Table 1 contd.

number	rule			condition
21	x	\vec{D}	\overleftarrow{D}	$x \in \{A, B\}$
		\overleftarrow{D}		
22	x	\overleftarrow{D}	z	$x \in \{\overrightarrow{C}, \overleftarrow{D}\}; Z \in \{A, B\}$ or $x = \overrightarrow{D}; z \in \{E_1, E_2\}$
		\vec{D}		
23	\vec{D}	E_2	z	$z \in \{A, R\}$
		\vec{D}		
24	\overleftarrow{D}	\overrightarrow{C}	\overleftarrow{D}	
		E_1		
25	x	\vec{D}	E_2	$x \in \{A, \overrightarrow{D}\}$
		E_1		
26	x	E_1	z	$x = A; z \in \{A, \overrightarrow{D}, E_2\}$ or $x = \overleftarrow{C}; z = \overrightarrow{D}$
		E_1		
27	x	\vec{D}	E_1	$x \in \{A, \overrightarrow{D}\}$
		E_2		
28	x	E_2	z	$x = A; z \in \{A, \overrightarrow{C}, \overrightarrow{D}, R\}$ or $x = R; z \in \{E_1, \overrightarrow{D}\}$
		E_2		
29	x	\overleftarrow{C}	\overleftarrow{D}	$x \in \{R, \overrightarrow{D}\}$
		E_2		
30	x	\vec{D}	R	$x \in \{\overleftarrow{C}, E_1\}$
		E_2		
31	R	R	R	
		F		
32	F	F	F	
		A		

Table 1 contd.

chain nearest to the initial automaton the picture of the propagation of signals would have been similar to that in figure 2, but with origin O' . In this case, the signal p'_3 , travelling from O' with velocity $1/3$, would have met the reflected signal p''_1 at the point A_1 (the centre of the chain). It is easy to see that the line representing the signal p'_3 meets that representing the reflected signal p''_1 at the point A corresponding to the position of the initial automaton. Hence, to carry out the first bisection of the chain, the velocity of the reflected signal p''_1 must be changed from 1 to $1/3$ at the point A . The line representing the signal travelling from O' with velocity $1/(2^{m+1}-1)$ meets that representing the signal travelling from O_1 with velocity $1/(2^m-1)$ at a point on the line AC which represents the line on which the velocities must change. To obtain the correct sequence of bisections every signal leaving O_1 with velocity $1/(2^m-1)$ must change its velocity to $1/(2^{m+1}-1)$. The slope of the line AC corresponds to a signal with velocity 1. Otherwise, the picture of signal propagation in figure 3 is similar to that in figure 2.

To construct the state transition table for an automaton as in Levenstein (1965) we introduce the relation of contraposition of internal states and transition functions of automata of the chain. Internal states \bar{D} and \bar{D} , \bar{C} and \bar{C} will be regarded as opposite. Each of the other states is opposite to itself. Values of transition functions are opposite when they are of the form

$$F(x_1, x_2, \dots, x_n) \text{ and } F^*(x_n^*, \dots, x_2^*, x_1^*),$$

where x_n and x_n^* are opposite states of the automaton. For the j th automaton of the chain the following relation holds:

$$F_j(x_{j-1}, x_j, x_{j+1}) = F_j^*(x_{j+1}^*, x_j^*, x_{j-1}^*). \quad (1)$$

Hence Table 1 only gives the transition functions for half the set, since their values for the other half can be obtained from (1).

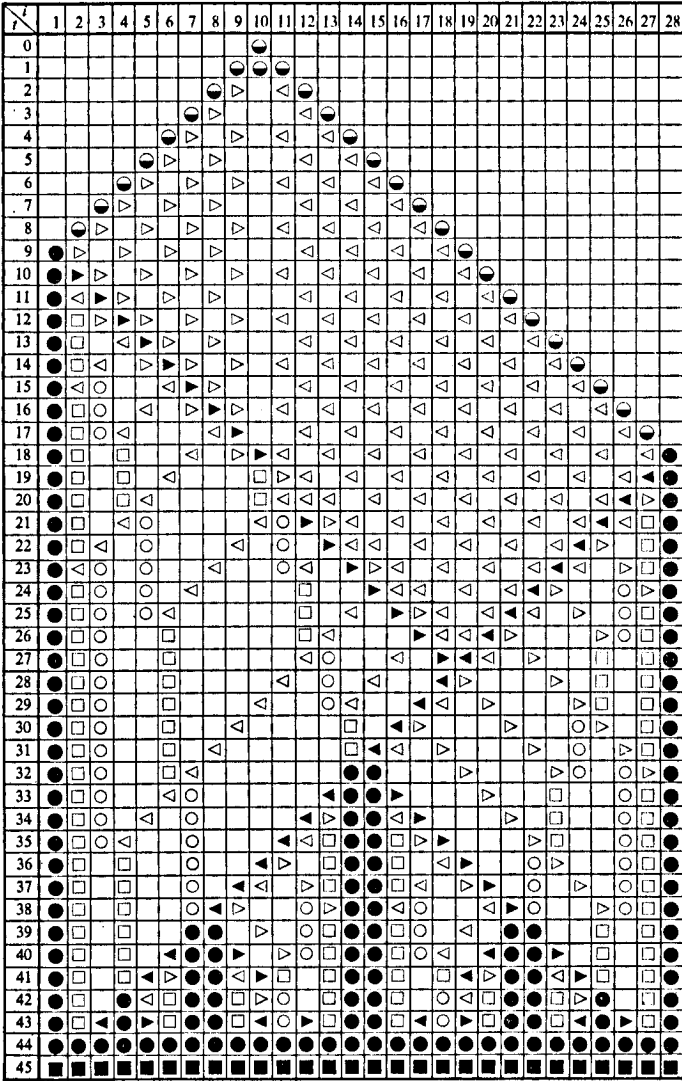
Figure 4 gives an example of the synchronization of a chain of 28 automata when the starting signal is applied to the tenth automaton.

Comparison of figures 2 and 3 shows that synchronization is achieved more quickly in the case of figure 3; the time saved is exactly that needed for a signal with velocity 1 to travel from O to H . Thus, if the starting signal is applied to an arbitrary automaton, the time required for synchronization is $T=2n-2-a_{\min}$, where a_{\min} is the distance from the initial automaton to the nearest end of the chain.

For practical applications of the synchronization problems, the following form of the problem seems more natural.¹

Suppose there is a chain of objects (devices) which have different set-in times (latent periods), that is, the i th object starts to operate τ_i time units

¹ In practice, it is not particularly difficult to arrange for a signal to be supplied simultaneously to a number of objects, using, for example, common communication lines.



State Notation

A	B	C	C'	D	D'	E ₁	E ₂	R	F
	●	▼	▲	▽	△	○	□	●	■

Figure 4

after a starting signal is applied to it. Time is assumed to be discrete, so that the τ_i are integers. The question is whether automata exist satisfying the following conditions:

1. Each object is associated with one automaton.
2. The automata form a chain in which each automaton is connected only to its two neighbours (except for the end automata which are each connected to only one neighbour).
3. The complexity of an automaton depends only on the set-in time of its associated object and does not depend on the length of the chain and the set-in times of the other objects.
4. After a starting signal has been applied to some automaton of the chain at time $t=0$ the objects must simultaneously begin to work at time T .

We shall not aim at a solution which is optimal as regards the time required for synchronization or the number of internal states. Here we shall merely show that a solution is possible. If the chain of automata is to solve the above problem, the i th automaton must produce a starting signal for the i th object τ_i time units before the moment of synchronization. Suppose we have a chain of automata, of the type considered in the previous problem, which passes into the terminal state simultaneously with the chain of objects. Any three automata occupying consecutive places in the chain are in the preterminal state one time unit before the transition to the terminal state and at no other time. The states of these automata at this moment depend on their states and on those of their right- and left-hand neighbours at the preceding moment, that is, on the states of five consecutive automata of the chain. In general, the states of any $2k+1$ consecutive automata at time t are determined by the states of $2k+3$ automata at time $t-1$. Hence the state of any automaton at time t is determined by its own state and the states of τ_i of its right- and left-hand neighbours at time $t-\tau_i$. Thus by observing a chain of $2\tau_i+1$ consecutive automata we can determine the moment of time which precedes the moment of synchronization by τ_i time units.

Consider a chain of $2\sum_{i=1}^n \tau_i + n$ automata which solves the synchronization problem. Decompose it into subchains of length $2\tau_i+1$, and consider each such subchain as a separate automaton. We keep the conditions of operation of the original automata in the chain. The original automata will be called subautomata, and a chain of $2\tau_i+1$ subautomata is called an automaton. From what has been said above, it is clear that by observing the states of the automaton formed by $2\tau_i+1$ subautomata, we can determine the moment of time which precedes by τ_i time units the moment of synchronization of the whole chain of $2\sum_{i=1}^n \tau_i + n$ subautomata. Detection of the states of an automaton at exactly τ_i time units before the synchronization of the whole chain can be

carried out by a logical network whose inputs are the states of the subautomata. Thus there is a solution to the problem of synchronizing a system of objects with differing set-in times by means of a chain of automata. The solution may be summarized as follows. As the automaton associated with the i th object, we take a subchain of $2\tau_i + 1$ subautomata which solves the synchronization problem for the whole chain of subautomata. The subautomata of all the automata form this whole chain of length $2\sum_{i=1}^n \tau_i + n$. The starting signal for the object is produced by a logical network whose inputs are the states of the subautomata of the given automaton. Note that the automaton continues to operate after it has supplied the starting signal to the object and all the subautomata pass into the terminal state at the same time as the objects start to work. The proposed construction of the automaton satisfies the conditions of the problem.

We shall assume that the starting signal given to the i th automaton is applied to its middle subautomaton. Then the time from the application of the starting signal until the objects start to work is equal to

$$4 \sum_{i=1}^n \tau_i + 2n - 2 - \tau_i - \min \left\{ \left[2 \sum_{j=1}^{i-1} \tau_j + (i-1) \right], \left[2 \sum_{j=i+1}^n \tau_j + (n-i-1) \right] \right\}.$$

If $\tau_i = 0$ for $1 \leq i \leq n$, then $T = 2n - 2 - a_{\min}$.

We shall consider the conditions satisfied by the states of the automaton τ_i time units before the moment of synchronization.

1. τ_i time units before synchronization, a chain of subautomata of length $2\tau_i + 1$ must contain at least one subautomaton in the preterminal state (state R , see Table 1 and figure 4).

At the k th bisection of the chain of subautomata, which occurs $\left[\frac{n-1}{2^k} \right]$ time units before synchronization, the distance between two non-adjacent subautomata in state R is $\left[\frac{n-1}{2^k} \right]$. Choose k so that

$$\left[\frac{n-1}{2^k} \right] \leq \tau_i < \left[\frac{n-1}{2^{k-1}} \right]. \tag{2}$$

Let $n-1 = \alpha 2^k + \beta$, where $\beta < 2^k$.

Then $\left[\frac{n-1}{2^k} \right] = \alpha$, and $\left[\frac{n-1}{2^{k-1}} \right] = 2\alpha + \left[\frac{\beta}{2^{k-1}} \right] \leq 2\alpha + 1$,

that is, $\left[\frac{n-1}{2^{k-1}} \right] \leq 2 \left[\frac{n-1}{2^k} \right] + 1$,

$$\text{or } \left\lceil \frac{n-1}{2^k} \right\rceil \geq \frac{1}{2} \left\lceil \frac{n-1}{2^{k-1}} \right\rceil - \frac{1}{2}. \quad (3)$$

It follows from (2) and (3) that

$$\tau_i \geq \frac{1}{2} \left\lceil \frac{n-1}{2^{k-1}} \right\rceil - \frac{1}{2} \quad \text{and} \quad 2\tau_i + 1 \geq \left\lceil \frac{n-1}{2^{k-1}} \right\rceil.$$

Hence at $\left\lceil \frac{n-1}{2^{k-1}} \right\rceil > \tau_i$ time units before synchronization, a segment of length $2\tau_i$ contains at least one subautomaton which is passing into state R . Then it is always possible to select from a subchain of length $2\tau_i + 1$ a subchain of length $\tau_i + 1$ having at least one end subautomaton in state R . If there are two consecutive subautomata in state R , then only one of them is included in the subchain of length $\tau_i + 1$. Henceforth we only consider such subchains.

2. If both end subautomata are in the preterminal state R and there is no subautomaton between them in the states \vec{C} , \vec{D} or R , then synchronization will occur after τ_i time units.

These two assertions follow directly from figure 4.

If the left (right) end automaton is in the preterminal state and the other end automaton is in state \vec{C} (\vec{C}), and there is no subautomaton between them in states \vec{C} , \vec{D} or R , then synchronization will occur after τ_i time units.

These two assertions also follow directly from figure 4.

3. If a chain of length $\tau_i + 1$ does not satisfy condition 2 and the left (right) subautomaton is in state R , then it contains a subautomaton in state \vec{C} (\vec{C}) and also a subautomaton in state E_1 or E_2 between the subautomaton in state \vec{C} (\vec{C}) and the right (left) end subautomaton.

For the proof, consider figure 5. The subautomaton in state R at the left-hand end of the chain emits a signal \vec{C} which travels to the right with velocity

1. Since $\tau_i > \left\lceil \frac{n-1}{2^k} \right\rceil$, on its way the signal \vec{C} meets inside the chain a signal

E_1 or E_2 coming in the opposite direction. The subautomaton at the meeting point passes into state R . At time τ_i time units before synchronization, the signal E_1 or E_2 is inside the subchain of length $\tau_i + 1$, since it travels toward the meeting place with velocity less than 1, and so must be to the left of a fictitious signal travelling from the point A to the meeting place with velocity 1. The number of subautomata through which the signal E_1 or E_2 passes before it meets \vec{C} is equal to the number of signals \vec{D} travelling in the direction of E_1 or E_2 and enclosed between E_1 or E_2 and the approaching signal \vec{C} .

Thus the distance from the meeting point B to the left end subautomaton is

$$\delta = d(E_1 \vee E_2, R) - n_D,$$

where $d(E_1 \vee E_2, R)$ is the distance from the subautomaton in state E_1 or E_2 to the left-hand end of the chain, and n_D is the number of subautomata in state D .

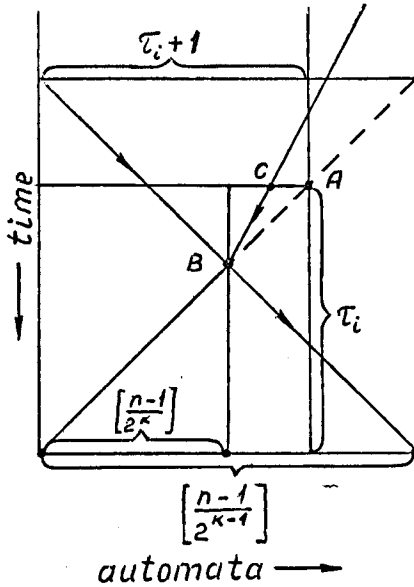


Figure 5

The time required for synchronization is the sum of the interval δ and the interval required for the signal \vec{C} to reach the meeting point, that is

$$\delta + [\delta - d(R, \vec{C})] = 2\delta - d(R, \vec{C}) = 2d(E_1 \vee E_2, R) - 2n_D - d(R, \vec{C}),$$

where $d(R, \vec{C})$ is the distance between the subautomata in state R and the subautomaton in state \vec{C} .

The condition for supplying the starting signal to an object is then expressed as

$$\tau_i = 2d(E_1 \vee E_2, R) - 2n_D - d(R, \vec{C}).$$

We shall now consider a variant of the synchronization problem in which transmission of a signal from one automaton to another takes τ time units, that is, the communication lines between the automata all have the same delay time. We are interested in a solution in which the complexity of the automata is independent of τ . The general idea of the solution is as follows. Each automaton is represented as a pair of subautomata (figure 6).

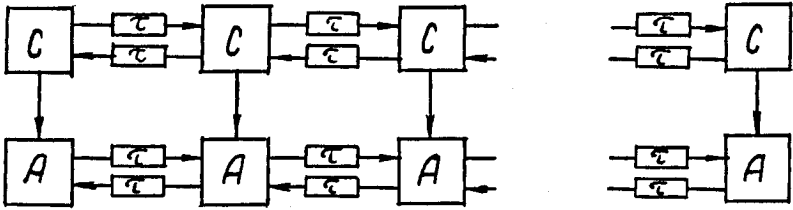


Figure 6

The subautomata C must generate gating signals with time interval $\tau + 2$, and the subautomata A solve the synchronization problem by the method described above in synchronism with the gating signals, that is, the subautomata C send signals for change of state to the subautomata A . Now to solve the synchronization problem for a chain with delay time it is sufficient to solve the problem of obtaining synphase periodic functioning of the subautomata C . The solution is trivial for period $2(\tau + 1)$. In order to put the subautomata C into periodic operation with period $\tau + 2$ we use an algorithm of successive approximation of signals. Consider two subautomata C . After the starting signal has been applied to automaton C_1 at time $t = 0$, it sends three signals a, b, c to automaton C_2 at three consecutive moments of time. Automaton C_2 sends these signals back, delaying signal a by 1 time unit, signal b by 2 units and signal c by 3 units. Automaton C_1 processes the signals it receives in a similar way. As a result, in $(\tau + 1)(\tau + 2)$ time units one of the automata will simultaneously emit a pair of signals $\{a, c\}$ which is identified with the signal b . An automaton emits the first synchronous signal when after emitting signal b it receives signal b at its input. This occurs simultaneously for both automata at time $(\tau + 2)^2$. From this moment on, when an automaton receives the synchronous signal it sends it back again. Thus signals begin to circulate in the system with period $\tau + 2$.

The above algorithm is illustrated by a diagram of the synchronization of two automata with delay $\tau = 5$ in the communication line (see figure 7). Each automaton has 12 internal states and its complexity is independent of the value of τ . The rules for changing internal states and forming output signals are given in Table 2.

We now return to the original statement of the problem. After the starting signal has been applied to an arbitrary automaton of the chain, this automaton becomes synchronized with its left- and right-hand neighbours according to the above algorithm. At the moment of synchronization of this subchain of three automata, the original automaton passes into a state corresponding to the starting state in the synchronization problem without delay. Subsequently an automaton only changes states at gated moments of time. Thus, each automaton of the chain can change its state only once during $\tau + 2$ time units and with this gating the synchronization problem is solved just as in a chain without delay.

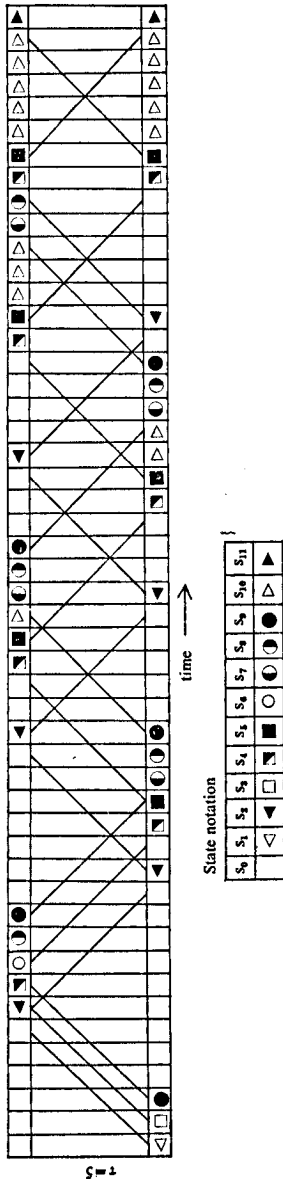


Figure 7

MACHINE LEARNING AND HEURISTIC PROGRAMMING

	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
S_0	S_0	S_2	S_2	—	S_0	S_4	S_4	—	—	—	S_0	S_0
S_1	S_3	—	—	—	—	—	—	—	—	—	—	—
S_2	S_0	—	—	S_4	S_0	—	—	—	—	—	—	—
S_3	S_9	—	S_4	—	—	—	—	—	—	—	—	—
S_4	S_5	—	—	—	S_5	—	—	S_5	S_5	S_6	S_5	—
S_5	S_{10}	—	—	—	S_{10}	S_{11}	—	S_{10}	S_{10}	S_7	S_{10}	—
S_6	S_8	—	—	—	—	—	—	—	—	—	—	—
S_7	S_8	—	—	—	—	—	—	—	—	—	—	—
S_8	S_9	—	S_4	—	—	—	—	—	—	—	—	—
S_9	S_0	—	—	—	—	—	—	—	—	—	—	—
S_{10}	S_{10}	—	—	—	S_{10}	S_{11}	—	S_{10}	S_{10}	S_7	S_{10}	—
S_{11}	—	—	—	—	—	—	—	—	—	—	S_0	S_0

Table 2

An essential feature of the problems discussed above was the unchanging rigid pattern of connections between the automata. It seems to be of interest to investigate the possibility of synchronization in a collective of automata with random dual interaction.

Consider a collective of N identical automata. By random dual interaction we mean that at each moment of time an independent equiprobable partition of the N automata into $\frac{1}{2}N$ pairs takes place (for simplicity N is assumed to be even). In each of these pairs the output signal of each automaton is the input signal for its partner. We shall consider Moore automata in which the output signal is the number of an internal state. Let $x_i(t)$ denote the internal state of the automaton A^i at time t . Then if automata A^i and A^j are paired at time t ,

$$x_i(t+1) = F[x_i(t); x_j(t)] \quad \text{and} \quad x_j(t+1) = F[x_j(t); x_i(t)].$$

If the transition function is symmetric, then $x_i(t+1) = x_j(t+1)$. We pick out the state $x=0$, called the initial state, and $x=n$, called the terminal (synchronous) state. We take $F[0;0]=0$, that is, two automata each in the initial state remain in this state when paired.

Let $r_j(t)$ be the number of automata in state j at time t . The fraction $\rho_j(t) = r_j(t)/N$ of automata in the state j at time t will be called the filling

number. Let $\bar{\rho}_j(t)$ denote the mathematical expectation of $\rho_j(t)$. For sufficiently large N , the mathematical expectation of the fraction of automata which form pairs¹ in which both automata are in state j is equal to $\rho_j^2(t)$. Similarly, the mathematical expectation of the fraction of automata forming pairs in which one is in state j and the other in state i is equal to $2\rho_j(t)\rho_i(t)$.

We now consider the synchronization problem for this collective of automata. We shall say that the collective is ε -synchronizable if after a starting signal has been applied to an arbitrary automaton at time $t=0$, we have

$$\begin{aligned} \Delta\bar{\rho}_n(t) &\geq 0 \text{ for all } t > 0, \\ \lim_{t \rightarrow \infty} \bar{\rho}_n(t) &= 1, \end{aligned}$$

and there exists T such that

$$\begin{aligned} \bar{\rho}_n(t) &\leq \varepsilon \text{ for } t \leq T, \\ \bar{\rho}_n(t) &\geq 1 - \varepsilon \text{ for } t \geq T + 1. \end{aligned}$$

Thus ε -synchronizability means that there is a moment at which at least $1 - 2\varepsilon$ automata simultaneously pass into the terminal state. We shall be interested in the construction of automata which produce ε -synchronization and in the asymptotic behaviour of the number of states of such automata as $N \rightarrow \infty$ and $\varepsilon \rightarrow 0$.

We consider two possible constructions of automata.

$$\begin{aligned} 1. \quad x_i(t+1) = x_j(t+1) &= \max [x_i(t); x_j(t)] + 1 && \text{if } \max [x_i(t); x_j(t)] \neq 0, \\ & && \max [x_i(t); x_j(t)] \neq n, \\ x_i(t+1) = x_j(t+1) &= n && \text{if } \max [x_i(t); x_j(t)] = n, \\ x_i(t+1) = x_j(t+1) &= 0 && \text{if } \max [x_i(t); x_j(t)] = 0. \end{aligned} \tag{4}$$

The starting signal transfers an automaton from state 0 to state 1. Consider the first few moments of operation of the system.

1. $\rho_0(0) = 1; \rho_j(0) = 0$ for all $j > 0$.
2. $\rho_0(1) = 1 - \frac{1}{N}; \rho_1(1) = \frac{1}{N}; \rho_j(1) = 0$ for all $j > 1$.
3. $\rho_0(2) = 1 - \frac{2}{N}; \rho_1(2) = 0; \rho_2(2) = \frac{2}{N}; \rho_j(2) = 0$ for all $j > 2$.
4. $\bar{\rho}_0(3) = \rho_0^2(2) = \left(1 - \frac{2}{N}\right)^2; \rho_1(3) = 0; \rho_2(3) = 0;$
 $\rho_3(3) = 1 - \left(1 - \frac{2}{N}\right)^2; \rho_j(3) = 0$ for all $j > 3$.

¹ The probability of a pair (j, j) being formed is

$$\frac{r_j}{N} \cdot \frac{r_{j-1}}{N-1} = \frac{r_j}{N} \cdot \frac{r_j}{N-1} - \frac{r_j}{N(N-1)} = \rho_j^2 \frac{N}{N-1} - \frac{\rho_j}{N-1} = \rho_j^2 + \frac{\rho_j}{N-1} - \frac{\rho_j}{N-1} = \rho_j^2 - \frac{\rho_j(1-\rho_j)}{N-1}.$$

Similarly, the probability of the pair (j, i) is $2\rho_j\rho_i + \frac{2\rho_j\rho_i}{N-1}$.

It is obvious that $\bar{p}_0(t)$ is the mathematical expectation of the number of automata in state 0 at time $t-1$ which are paired with an automaton in state 0, that is, $\bar{p}_0(t) = \bar{p}_0^2(t-1)$.¹

The solution of this difference equation with the above initial conditions is $\bar{p}_0(t) = (1 - 2/N)^{2^{t-2}}$. Taking into account the rule for changing states, for sufficiently large N we may write:

$$\bar{p}_0(t) = \exp(-2^{t-1}/N); \rho_t(t) = 1 - \exp(-2^{t-1}/N); \rho_j(t) = 0 \quad (0 < j \neq t).$$

It is also obvious that

$$\rho_n(t) = \begin{cases} 0 & \text{for } t < n, \\ 1 - \exp(-2^{t-1}/N) & \text{for } t \geq n. \end{cases}$$

To achieve ε -synchronization it is necessary that

$$1 - \exp(-2^{n-1}/N) \geq 1 - \varepsilon,$$

that is, $n \geq \log_2 N + \log_2 \ln(1/\varepsilon) + 1$. (6)

Hence to achieve ε -synchronization the memory capacity of the automata must increase as N increases.

$$\begin{aligned} 2. \quad x_i(t+1) = x_j(t+1) &= \min [x_i(t); x_j(t)] + 1, & \text{if } \min [x_i(t); x_j(t)] \neq n, \\ & \max [x_i(t); x_j(t)] \neq 0, \\ x_i(t+1) = x_j(t+1) &= n, & \text{if } \min [x_i(t); x_j(t)] = n, \\ x_i(t+1) = x_j(t+1) &= 0, & \text{if } \max [x_i(t); x_j(t)] = 0. \end{aligned}$$

The starting signal transfers an automaton from state 0 to state 1. Introduce a new variable $\gamma_j(t) = \sum_{i=j}^n \rho_i(t)$. It is not difficult to see that at time t , only those automata will be in states with numbers greater than $j-1$ which at time $t-1$ were in states with numbers not less than $j-1$; that is,

$$\bar{\gamma}_j(t) = \bar{\gamma}_{j-1}^2(t-1) \quad \text{for } j \geq 2. \quad (7)$$

We consider the system of difference equations (7) with the initial conditions:

$$\gamma(0) = 1; \gamma_1(0) = 0; \gamma_1(1) = 1/N; \gamma_1(2) = 2/N,$$

and, similarly to the previous case, $\gamma_1(t) = 1 - \exp(-2^{t-1}/N)$. Note that $\gamma_2(1) = 0, \gamma_3(2) = 0$ and in general, $\gamma_j(j-1) = 0$.

Solution of the system (7) under the above initial conditions gives

$$\bar{\gamma}_j(t) = \begin{cases} 0 & \text{for } t \leq j+1, \\ [\bar{\gamma}_j(t-j+1)]^{2^{t-1}} & \text{for } t > j+1. \end{cases}$$

Note that $\gamma_n(t) = \rho_n(t)$ and

$$\bar{\gamma}_n(t) = \begin{cases} 0 & \text{for } t \leq n-1, \\ [1 - \exp(-2^{t-n}/N)]^{2^{t-1}} & \text{for } t > n+1. \end{cases} \quad (8)$$

¹ Obviously this difference equation only describes the process approximately. Here we use this approximate description, disregarding the influence of the spread of the distribution of $p_0(t)$.

Note that in this construction we allow the automata to leave the terminal state. In spite of this, it is easy to see that $\gamma_n(t)$ is a monotonic increasing function of t and $\lim_{t \rightarrow \infty} \gamma_n(t) = 1$.

We shall introduce the auxiliary notations $\exp(-2^{t-n}/N) = x$ and $2^{n-1} = 1/\alpha$. Consider the system of inequalities

$$\begin{aligned} \gamma_n(t) &= (1-x)^{1/\alpha} \leq \varepsilon, \\ \bar{\gamma}_n(t+1) &= (1-x^2)^{1/\alpha} \geq 1-\varepsilon. \end{aligned} \tag{9}$$

We shall determine the domain of existence of solutions of the system (9).

$$\left. \begin{aligned} 1-x &\leq \varepsilon^\alpha \\ 1-x^2 &\geq (1-\varepsilon)^\alpha \end{aligned} \right\} \begin{aligned} x &\geq 1-\varepsilon^\alpha \\ x^2 &\leq 1-(1-\varepsilon)^\alpha \end{aligned} \tag{10}$$

The system (10) is equivalent to (9). For $\alpha \leq 1$,

$$(1-\varepsilon)^\alpha = 1 - \alpha\varepsilon - \frac{1-\alpha}{2} \alpha\varepsilon^2 - \frac{(1-\alpha)(2-\alpha)}{2 \cdot 3} \alpha\varepsilon^3 - \dots = 1 - \alpha\varepsilon - R(\varepsilon),$$

$$R(\varepsilon) > 0,$$

and hence $1 - (1-\varepsilon)^\alpha > \alpha\varepsilon$. Expanding $(1-\varepsilon^\alpha)$ in a Taylor series in powers of α gives

$$(1-\varepsilon)^\alpha = \alpha \ln(1/\varepsilon) - \frac{\alpha^2 \ln^2(1/\varepsilon)}{2} + \frac{\alpha^3 \ln^3(1/\varepsilon)}{2 \cdot 3} - \dots$$

and so $(1-\varepsilon^\alpha) < \alpha \ln(1/\varepsilon)$ for $\alpha < 1/\ln(1/\varepsilon)$.

Then the domain of existence of solutions of the system

$$\begin{aligned} x &\geq \alpha \ln(1/\varepsilon), \\ x &\leq \sqrt{\alpha\varepsilon}, \\ \alpha &< 1/\ln(1/\varepsilon) \end{aligned} \tag{11}$$

is a domain of solution for system (10) and a solution of system (11) is a solution of system (10). Figure 8 shows the functions $x = \alpha \ln(1/\varepsilon)$ and $x = \sqrt{\alpha\varepsilon}$, and the domain of existence of solutions of (11) formed by these curves. It is obvious that a solution exists only for $\alpha < \varepsilon/\ln^2(1/\varepsilon)$. Hence

$$n \geq \log_2(1/\varepsilon) + 2 \log_2 \ln(1/\varepsilon) + 1. \tag{12}$$

Now let

$$\hat{n} = \log_2(1/\delta) + 2 \log_2 \ln(1/\delta) + 1 \geq \log_2(1/\varepsilon) + 2 \log_2 \ln(1/\varepsilon) + 1,$$

that is, $\delta \leq \varepsilon$. Consider the moment

$$\hat{t} = \log_2 N + \hat{n} + \log_2 \{ \ln(1/\delta) + \ln \ln(1/\delta) \}.$$

Then $\exp(-2^{\hat{t}-\hat{n}}/N) = \delta/\ln(1/\delta)$ and $2^{\hat{n}-1} = \{ \ln(1/\delta) \}^2/\delta$. Hence

$$\bar{\gamma}_n(\hat{t}) = \left(1 - \frac{\delta}{\ln(1/\delta)} \right)^{\{ \ln(1/\delta) \}^2/\delta}.$$

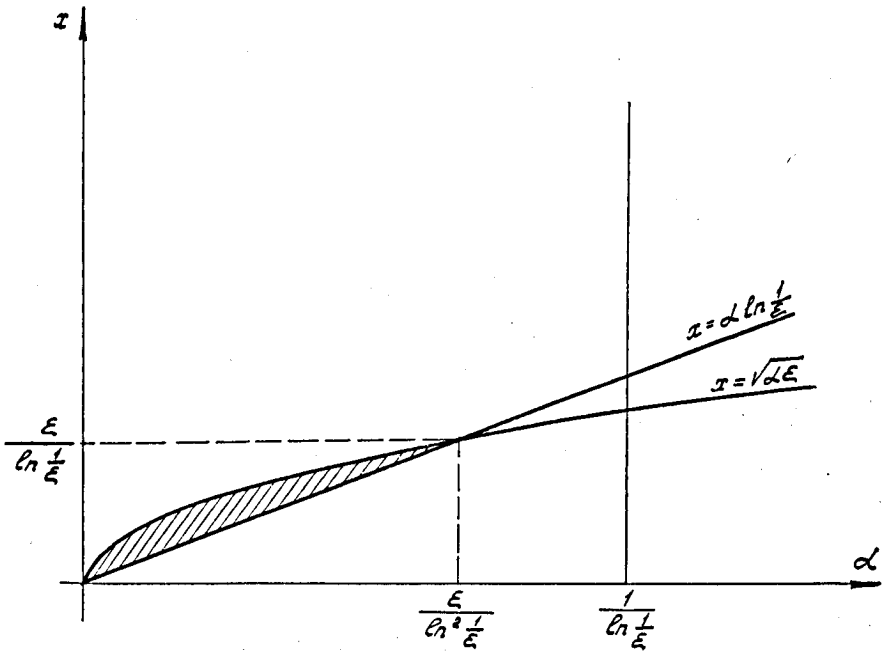


Figure 8

Since $\delta / \ln (1 / \delta)$ is small, we can assume that

$$\bar{\gamma}_n(i) \approx \exp (-\ln (1 / \delta)) = \delta \leq \epsilon.$$

On the other hand,

$$\bar{\gamma}_n(i+1) = \left(1 - \frac{\delta^2}{\{\ln (1 / \delta)\}^2}\right)^{(\ln (1 / \delta))^2 / \delta} \approx e^{-\delta} \approx 1 - \delta \geq 1 - \epsilon.$$

Thus we can assert that, *independently of N* , for any arbitrarily small $\epsilon > 0$ there exists $n_0(\epsilon)$ such that for any $n > n_0(\epsilon)$, more than $1 - 2\epsilon$ automata simultaneously pass into the terminal state at time $i(N, n)$. Since $\gamma_n(t)$ is a monotonic increasing function of t and $\gamma_n(t) \leq 1$, the time at which the terminal state is attained is unique.

The solution to this problem was based on two assumptions: (i) for large N the equations can be taken for mean values, ignoring dispersion; (ii) terms of order $1 / N$ can be neglected. The validity of these assumptions is not obvious and to check the correctness of the results obtained the behaviour of a collective of 1024 automata was simulated on a computer. For each n , 200 experiments were carried out. The results are shown in figure 9. The upper solid curve represents the mean of the maximal number of automata simultaneously passing into the synchronized state ($\Delta \rho_{\max}$). The lower solid curve represents the mean-square deviation ($\sigma(\Delta \rho_{\max})$) and the

dotted curve represents $1 - 2\varepsilon(n)$. It is clear that the experiment does not contradict the results found above.

The above models succeed in meeting the requirements of synchronization: all the automata to within ε pass into the terminal state at exactly the same time. The second version succeeds in preserving an important feature of the

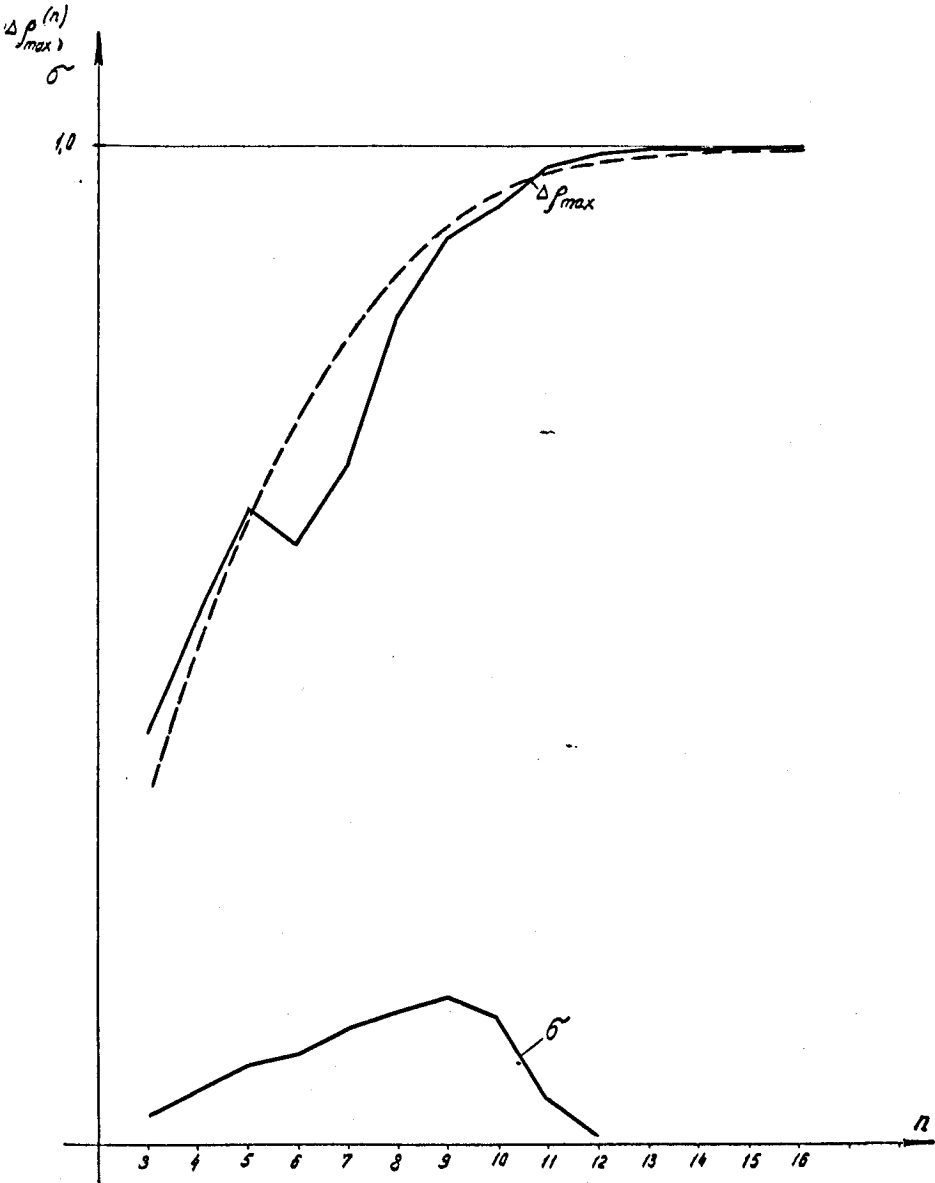


Figure 9

determinate synchronization problem – the independence of the complexity required in each automaton of the total number of automata.

The problems considered here naturally do not exhaust all the possible ways of organizing interaction but I hope that the work presented above will give some appreciation of the possibilities and the basic features of interaction in collectives of automata.

Acknowledgements

I am happy to express my gratitude to V.B. Marakhovskii and V.A. Peschanskii, with whom I have worked on these problems for the last two years and some of whose results are used here. I also thank B.L. Ovsievich for his help in writing this paper.

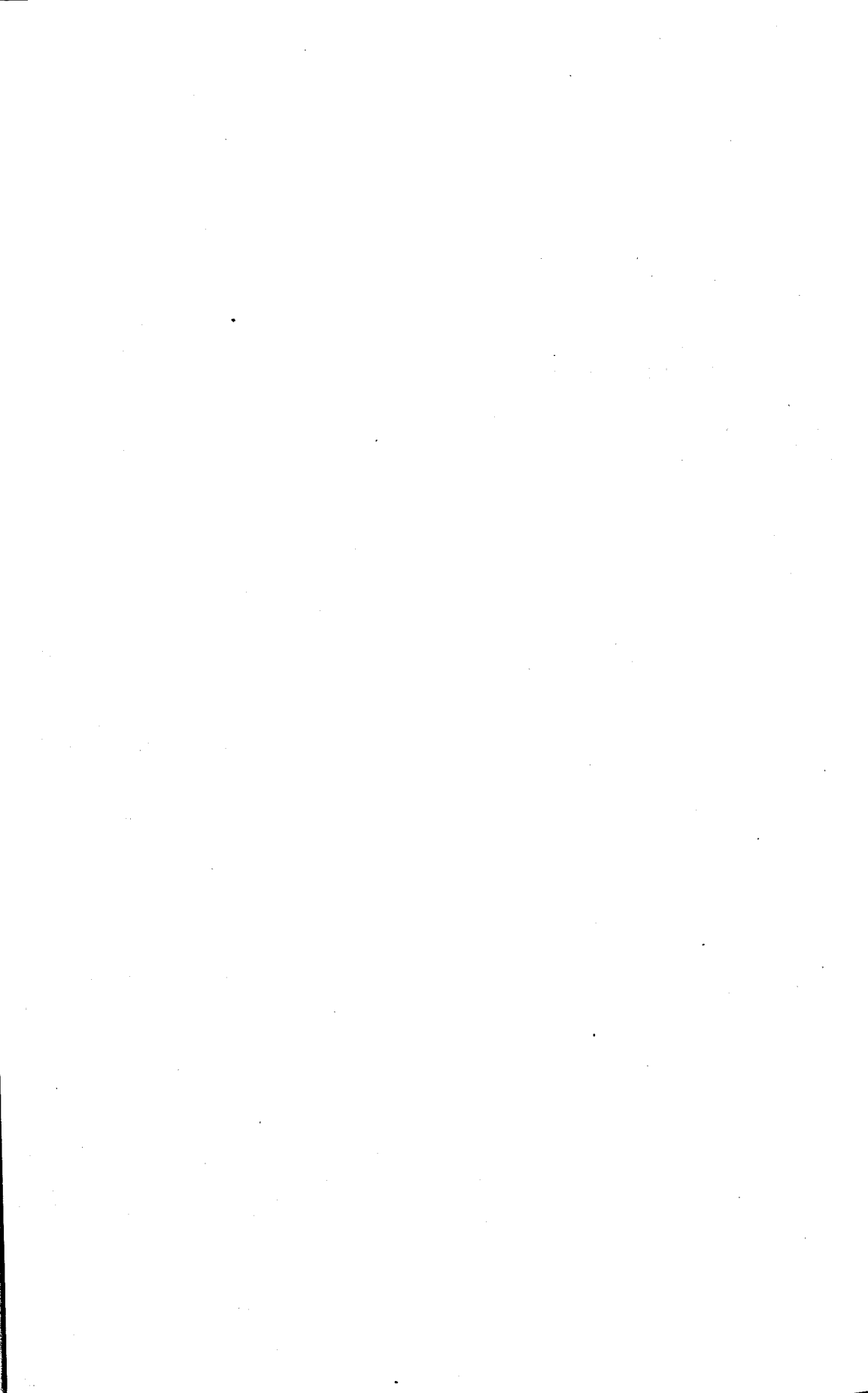
REFERENCES

- Arrow, K. J., Hurwicz, L. & Uzawa, H. (1958) *Studies in linear and non-linear programming*. Stanford.
- Balzer, R. (1967) An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, **10**.
- Borovikov, V.A. & Bryzgalov, V. I. (1965) The simplest symmetric game of several automata. *Avtomat. i Telemekh.*, **26**, 683–7. (1965) Translated in *Automation and Remote Control*, **26**, 681–4.
- Burks, A. W. (1964) Cellular automata. *Trans. Proc. Relay Systems and Finite Automata*, Burroughs Corp., Tape No. MT, art. 26, 1964.
- Gel'fand, I. M., Pyatetskii-Shapiro, I. I. & M. L. Tsetlin, M. L. (1963) On some classes of games and games of automata. *Doklady Akad. Nauk SSSR*, **152**, 845–8.
- Ginzburg, S. L. & Tsetlin, M. L. (1965) Some examples of models of collective behaviour of automata. *Problemy Peredachi Informatsii*, **1**, No. 2. Translated in *Problems of Information Transmission* **1** (1965).
- Levenstein, V. I. (1965) A method of solving the synchronization problem for a chain of automata in minimal time. *Problemy Peredachi Informatsii*, **1**, No. 4. (1965) Translated in *Problems of Information Transmission*, **1**.
- Michie, D. & Chambers, R. A. (1968) BOXES: an experiment in adaptive control. *Machine Intelligence 2* (eds Dale, E. & Michie, D.). Edinburgh: Oliver & Boyd
- Moore, E. F. (1964) The firing squad synchronization problem. *Sequential machines: selected papers* (ed. Moore, E. F.). Reading, Mass.
- von Neumann, J. (1966) *Theory of self-reproducing automata* (ed. and completed by Burks, A. W.). Illinois.
- Pittel', B. G. (1965) On asymptotic properties of a variant of the Goore game. *Problemy Peredachi Informatsii*, **1**, No. 3, 99–112. Translated in *Problems of Information Transmission*, **1** (1965).
- Tsetlin, M. L. (1963) Finite automata and models of simple forms of behaviour. *Uspekhi Mat. Nauk*, **18**, No. 4, 3–28. Translated in *Russian Math. Surveys*, **18**, No. 4 (1963) 1–27.
- Tsetlin, M. L. & Varshavsky, V. I. (1965) Collectives of automata and models of behaviour. *Trudy III Vsegoynuznogo Soveschaniya po Avtomaticheskomu Upravleniyu*, Odessa 1965.
- Tsetlin, M. L. & Varshavsky, V. I. (1966) Automata and models of collective behaviour. *Proceedings of the Third Congress of the International Federation for Automatic Control*, London.

- Ulam, S. M. (1962) On some mathematical problems connected with the patterns of growth of figures. *Proc. Symposia in Applied Mathematics*, Vol. 14: *Mathematical problems in the biological sciences*. American Math. Soc. 1962.
- Varshavsky, V. I. (1968) Collective behaviour and control problems. *Machine Intelligence* 3 (ed. Michie, D.). Edinburgh.
- Varshavsky, V. I. (1969 in press) Synchronization of a collective of automata with random dual interaction. *Avtomat. i Telemekh.*
- Varshavsky, V. I., Meleshina, M. V. & Perekrest, V. T. (in press) Collective behaviour in the problem of allocation of resources. *Avtomat. i Telemekh.* (in press).
- Varshavsky, V. I., Peschanskii, V. A. & Marakhovskii, V. B. (1968) Some variants of the synchronization problem for chains of automata. *Problemy Peredachi Informatsii*, 4, No. 4. (1968) Translated in *Problems of Information Transmission*, 4.
- Varshavsky, V. I. & Vorontsova, I. P. (1963). The behaviour of stochastic automata with variable structure. *Avtomat. i Telemekh.*, 24, 353-60. (1963) Translated in *Automation and Remote Control*, 24, 327-33.
- Varshavsky, V. I. & Vorontsova, I. P. (1964) Stochastic automata with variable structure. *Trans. Proc. Relay Systems and Finite Automata*, Burroughs Corp., Tape No. MT, art. 38.
- Varshavsky, V. I., Vorontsova, I. P. & Tsetlin, M. L. (1962) The training of stochastic automata. *Biologicheskie Aspekty Kibernetiki*, Izdat. AN SSSR, Moscow.
- Varshavsky, V. I., Meleshina, M. V. & Tsetlin, M. L. (1965) Behaviour of automata in periodic random environments and the synchronization problem in the presence of noise. *Problemy Peredachi Informatsii*, 1, No. 1. Translated in *Problems of Information Transmission* 1.
- Volkonskii, V. A. (1965) Asymptotic properties of the behaviour of simple automata in a game. *Problemy Peredachi Informatsii*, 1, No. 2, 36-53. Translated in *Problems of Information Transmission* 1 (1965).
- Waksman, A. (1966) An optimum solution to the firing squad synchronization problem. *Information and Control*, 9, 66-78.



**COGNITIVE PROCESSES:
METHODS AND MODELS**



Steps towards a Model of Word Selection

G. R. Kiss

Medical Research Council

Speech and Communication Research Unit, Edinburgh

INTRODUCTION

This paper is a progress report on the work related to the word storage model first described in Kiss (1967a). In that paper eight postulates have been outlined to characterize a set of hypothetical processes and structures which could account for the behaviour of human beings when they use meaningful words in certain experimental and natural situations. An outline of some phenomena in verbal behaviour is to be found there, and the general direction of the work remains the same.

Of the eight postulates of the previous paper Postulate 5, describing the transmission of excitation among the elements of a parallel processor, and Postulate 8, describing the process of word selection, are most affected by the more recent work. In this paper the idea of a 'word store' is introduced and takes the place of the parallel processor. This is more in line with current terminology in the psychological literature, but essentially covers the same idea.

The changes are, mainly, the introduction of stochastic process theory for the characterization of the transmission of excitation, and the introduction of a more 'passive' rôle for the central processor in executing a search over the elements of the word store.

At first a general description of the word store will be given, followed by a restatement in more mathematical terms. The model will then be related to the behaviour of subjects in the word association experiment and the results of some tests of the model will be presented.

1. THE WORD STORE AS A STOCHASTIC INFORMATION RETRIEVAL SYSTEM

The basic idea behind this paper is that the word store should be regarded as an information retrieval system which is governed by stochastic processes.

That probabilistic considerations should dominate a discussion of the word store will occasion no surprise for anyone who ever felt frustrated at his

apparent inability to recover a familiar word or name from his store on some occasions. We all accept the fact that some rare items may not be available when needed, that is, that there is only a certain *probability* – and sometimes a rather small one – that an item which has not been used very often will be found. It comes as a greater shock that occasionally even quite well-known words are ‘lost’ in the system.

When I say that the word store is an information retrieval system, I want to emphasize the contrast with a mere *storage* system. The essential difference between the two is in the way of gaining *access* to an item of information. It seems that most of the interesting psychological problems are associated with this function, and not with the fact that information is *held* in the store. By a storage system I shall mean here a system which is *guaranteed* to give the required item of information when interrogated. In contrast, an information retrieval system will provide that information with a probability which can be less than one, and sometimes may well be zero.

The same system may operate in either of these two ways on different occasions. This may be due to many reasons, two of which are that we may not know where an item is in the store and have to search for it, or we may not even know what item we want from the store, although we may be able to recognize the item we want, if it is brought out from the store and presented for testing. The mere necessity of a search process does not necessarily make the system an information retrieval system. If exhaustive searches of the store can be tolerated then, as long as an item is in the store, it will always be recovered, given sufficient time.

When an information retrieval system is interrogated the required information has to be specified in some form. Such a specification may be satisfied by various items to a varying degree. If this is the case, then it becomes necessary to define the *relevance* of an item to a query. Instead of coming up with one unique answer, the system may provide a whole series of answers, possibly graded in their relevance.

I add the word *stochastic* to the description of the word store, because I want to emphasize that the actual processes whereby the operation of the word store are implemented are themselves governed by probabilistic laws. This statement refers to the dynamics of the system as far as its *mechanism* is concerned, in distinction from the discussion in the preceding paragraphs where its overall behaviour was discussed.

2. A MODEL OF THE WORD STORE

General

Having outlined what I mean by a stochastic information retrieval system, let me now describe in an intuitive manner how this system may be organized.

The word store contains *representations of words*. For simplicity, I shall talk as if the store contained words, although of course it only contains their representations. At any given instant of time a word in the store can have a

certain level of *activity*. This activity changes in time for a number of reasons, to be discussed later. *A state of the word store is specified by giving the current level of activity of each word in it.* The word store is therefore a system which can be in a large number of different states. These states define the state-space of the system.

The word store operates by going from one state to another. These *transitions* are caused by changes in the activity levels of words: some will increase, some decrease, and some may remain unchanged. The state of the word store may change in response to a variety of influences. First, information may be received from the sense organs through perceptual mechanisms. Such information may selectively alter the activities of some words in the store. Second, some higher level parts of the overall human information processing system may alter the activity levels. This is the influence of thought processes. Third, the state of the word store may change because of intrinsic influences between words in the store.

It is particularly this third kind of mechanism with which I shall be dealing in this paper. I shall call these transitions 'free'. Once the system is started from an initial state, the time course of its evolution would be determined by the connections between the words if all other external influences are absent. This would rarely be the case and would happen for relatively short periods only.

The idea that the representations of words are inter-connected with each other could be traced back to early associationistic psychology. In a more specific form it has been restated by Treisman (1960). The suggestion is that the activity of one word may influence the activities of other words by means of such connections. The state of the word store may therefore change in the absence of any external influence, merely because of the interactions between the words themselves. I am suggesting here that *these free transitions are stochastic in nature, owing to the fact that the interactions between words are probabilistic.* The activity in one word has a probability of influencing by a certain amount the activity of another word. Once these probabilities and the initial state of the system have been specified, it should be possible to determine the evolution of the system through time, according to the mathematical theory of stochastic processes.

At any instant during this process a word can 'enter consciousness', or, to use a more neutral terminology, it can become available for further processing in the central processor. The probability that a word will do so is assumed to be a function of its level of activity at that instant. The higher the activity, the more probable it is that the word will be selected for further processing. In the language of some writers on choice behaviour (Luce 1959), the activity level corresponds to the 'response strength' of an alternative. The choice is then made randomly, so that the probability of choosing an alternative is the ratio of the response strength of that alternative to the sum total of the response strengths over all alternatives in the system.

The two essential components of the retrieval process are then, in summary: the evolution of the word store in time through free transitions from some starting state, and a 'decision stage', in which a random choice is made among the words so that the relative probability of choosing any one word is proportional to its relative activity at that moment.

Physiological

Although it is claimed by some psychologists that research and theorizing can be pursued at a psychological level without reference to physiological mechanisms, there is no *virtue* in doing so. Moreover, it will be seen that a consideration of possible physiological mechanisms can be useful in eliminating some of the possible models. In fact, I would like to put it on record that it was the consideration of neural mechanisms which helped me resolve some of the inconsistencies of my earlier model.

Let me now outline one possible implementation of the word store in terms of neurons. It should be kept in mind, however, that the model presented here is not dependent on this particular implementation. Other mechanisms could also be assumed. My purpose here is merely to show that the model is not inconsistent with current physiological knowledge.

I shall assume that the representations of words are made up of sets of neurons. These sets would be distributed over a large area of that portion of the cerebral cortex which is concerned with the storage of words. No attempt will be made here to identify the anatomical location of this cortical area. Let us call the set of neurons which represent any given word the *representative set* of that word. Any representative set would then have members distributed over a relatively large cortical area. It is at present an open question whether the representative sets of different words are disjoint or overlapping, allowing the sharing of neurons between several representative sets.

Since neurons are interconnected through synapses, the firing of one neuron can have an influence on the state of a number of other neurons. This influence may not be limited to synaptic transmission. The nature of the interaction between neurons is still very much a matter of controversy amongst physiologists. Apart from the rôles of spatial arrangement and temporal patterning of synaptic interactions, additional effects arise from ephaptic interaction between adjacent neurons (Grundfest 1959) and from gross electric field influences (Bullock 1957). In addition to the complexity of these mechanisms, there is evidence that there are some truly random processes operating at this level in the form of random threshold fluctuations (Frishkopf and Rosenblith 1958), internal noise, and irregular spontaneous activity (Fatt and Katz 1952; Verveen 1961).

It is not, therefore, entirely without support to suppose that *for our purposes here* the interaction between neurons could be regarded as a probabilistic affair. I shall summarize this by saying that *the firing of a single neuron can bring about the firing of a number of other neurons only with a certain probabil-*

ity. I am arguing that as far as the word store model is concerned, it makes sense only to talk in terms of such probabilities. Whether there is true randomness here, or the randomness is only apparent because of the complexity of the processes and our ignorance about their details, is irrelevant for our purpose.

Owing to these interactions between neurons the firing of any neuron in one representative set will have a probability of activating a certain number of neurons within the same set and also in other sets. According to these numbers we can theoretically define the *strength of association*, or *transmittance* between any two sets.

Let us now see how we can map the concepts outlined in the previous section into neurophysiological mechanisms.

Each word in the store has a representative set of neurons. The activity of a word is determined by the number of neurons currently firing in its representative set. The state of the word store is specified by the momentary levels of activities of the representative sets of all words. Due to synaptic and other interactions between neurons, there is a certain transmittance connecting any two representative sets. The transitions of the system correspond to changes in the activity levels.

It should perhaps be discussed a little further how the levels of activity can be said to be maintained in the representative sets over time. I said that the activity level corresponds to the number of neurons firing at any given time in the set. Since the firing of a neuron is followed by an abrupt increase in its threshold which then decays in a roughly exponential fashion towards the resting level, there is a refractory period during which the neuron is not available for firing. If a continuously varying activity level in the set is required, this can only be accomplished if at any two successive instants of time separated by a short interval the activity is carried by different neurons within the same set. The mechanism will work as long as the level of activity (the proportion of neurons in the set firing at any instant) is not too large. If this proportion approaches unity, then it must necessarily fall to a value near zero at the next instant, since there will be no cells available for firing (all of them being refractory). Certain discontinuities are therefore to be expected if the activation of any word is increased in an extreme fashion. Such discontinuities are observed behaviourally in various satiation phenomena.

Looking at the word store system as a whole, one can assume two rather different models. In one of them the system starts from a quiescent state. The stimulus puts the system into some starting state by injecting a certain amount of activity into one of the representative sets (or into several sets) and then the excitation spreads to other sets in an exponentially increasing fashion. It can be shown mathematically that in a system of this kind the activity will either die out or will increase indefinitely. Which of these two cases will happen depends on the mean number of 'progeny' produced by any neuron.

Physiologically, of course, the activity cannot increase indefinitely and must necessarily fall after reaching a maximum, as discussed previously.

In the second version of the model the system is never in a quiescent state. The stimulus impinges on some ongoing activity, and the starting state is determined both by the stimulus and by the momentary state of the system when the stimulus is applied. In this model there is always some general level of activity, maintained possibly by the regulating action of the nonspecific activating systems. The stimulus may momentarily increase the general activity, but this effect dies out relatively quickly compared with the time intervals involved in word selection tasks (normally between one and two seconds). In this model the overall level of activity does not increase in the system as a whole during the word selection process, but the *relative proportion* of this activity carried by any given word changes in the course of time. The activity of any word can only increase here at the expense of others. The sum total of activity is constrained to a fixed level. The possibility can still be left open that this level is adjustable over longer time intervals corresponding to degrees of 'activation' or 'arousal'.

Mathematical

I shall now turn to the problem of how the behaviour of such a model can be characterized mathematically. The essential content of this section will be a formulation of the general ideas outlined in the previous sections in a mathematical form.

What is needed is a mathematical tool which is able to express the probabilistic nature of the word store and is also able to cope with the structural organization of the model. By this second point I mean to emphasize the need for a mathematical formalism which is not only able to reflect some idealized general characteristics of the system, such as frequency or probability distributions, relationships between mean values and time variables, but can be extended to the study of the 'fine structure' of the system's behaviour. An illustration of a mathematical model's failure to achieve this is the use of the so-called pure death process model by McGill (1963) for describing the empirical curve obtained by Bousfield and Sedgewick (1944), when subjects are asked to name four-legged animals or cities in the US, and a cumulative record of the number of items produced as a function of time is plotted. The pure death process model does lead to a mathematical expression which fits the empirical curve well, but as McGill comments:

Several weaknesses in the extinction chain model of Bousfield and Sedgewick's data should be recorded. For example, the associations come out in clusters triggered by naming a member of an obvious subclass (e.g. geographic clusters). This sort of clustering produces roughness in the data with which a simple extinction model cannot cope. In addition, systematic deviations from the exponential curve form at early stages are [often noted. These discrepancies do not appear to be large, and the rough

grain in the data due to clustering does not seem to affect the curve shape seriously, but both effects are observed and they are simply ignored by the model.

If a model could be formulated which would be able to deal with these ignored characteristics of the data, then that model would be preferable to the one described by McGill.

The mathematical apparatus of stochastic processes is now widely used by psychologists for the description of behaviour. Most of these applications of stochastic models are not structural models in the above sense. Structural models have been extensively discussed by Harary, Norman and Cartwright (1965) in a recent textbook. The main preoccupation of that book is with graph theory. The relationship between stochastic processes (especially Markov chains) and graph theory is of course well known. The state transition diagrams of Markov chains are by now familiar to most psychologists from Shannon's work and subsequent developments of finite state grammars. The relationship with graphs has not, however, been emphasized in most applications of stochastic theory to psychology, for the reason that these applications are not structural.

It is suggested in this paper that the word storage model should be an application of stochastic models in this structural sense. As we shall see later, this leads to the possibility of using graph theoretical tools for the treatment of structural problems, while the analytical methods of the theory of stochastic processes remain useful for the study of large-scale behaviour.

In specifying a stochastic process, one must make a choice between two alternatives both as to the state space and as to the time scale of the system. For reasons of simplicity, mainly, the discussion will be at first in terms of a discrete state space and discrete time. In choosing the discrete model I am assuming that there are a finite or denumerably infinite time points for which the process is defined. Whether these assumptions are justified can only be decided on the basis of more detailed knowledge of brain physiology. These assumptions are not essential, however, and the model can be extended to the continuous case. Discrete models are often used as approximations to the continuous case.

The most reasonable model for this system seems to be a multi-type branching process, or vector Markov process (Harris 1963; Bharucha-Reid 1960). I shall omit here a general mathematical characterization of the multi-type branching process and turn directly to an interpretation of the word store model in these terms.

The state of the system is defined by a random vector

$$X_n = (X_{1n}, X_{2n}, \dots, X_{Nn}),$$

where X_n is the number of neurons active in the system at the n th transition of the process, N is the number of words in the system, and the component scalar random variables X_{in} ($i=1, 2, \dots, N$) represent the number of neurons

active in the representative set of the i th word at the n th transition of the process, that is, the X_{in} are the activity levels of the words of the n th transition.

A neuron belonging to word i has the probability

$$p_i(a_1, a_2, \dots, a_N)$$

of activating in the next transition a_1 neurons for word 1, a_2 neurons for word 2, . . . , a_N neurons for word N . If an initial random vector X_0 and the probabilities p_i are given, then this determines the probability law for the free transitions of the system. If it is assumed that X_{n+1} depends only on X_n and is independent of all previous values, then the process has the Markov property and can be called a vector Markov process.

One can now define generating functions for the $(n+1)$ st 'generation' of neurons which are 'progeny' of a neuron of type i active at time zero:

$$F_{i,n+1}(s) = [F_i(s)]^{n+1},$$

where $s = (s_1, s_2, \dots, s_N)$, and the square brackets denote functional iterates, and

$$F_i(s) = F_{i1}(s) = \sum_{a_1=0}^{\infty} p_i(a_1, a_2, \dots, a_N) s_1^{a_1} s_2^{a_2} \dots s_N^{a_N}.$$

The moments of X_n can be obtained by differentiating the generating function. In particular,

$$m_{ij}^{(n)} = \left. \frac{\partial F_{in}}{\partial s_j} \right|_{s_1=s_2=\dots=s_N=1}$$

is the expected number of neurons of word j at the n th transition, activated by one neuron of word i active at time zero. It can be shown that the relation $[m_{ij}^{(n)}] = [m_{ij}^{(1)}]^n$ exists* between the first moments of the n th transition and those of the first transition. If we let $M = [m_{ij}^{(1)}]$ denote the first moment matrix, then we have

$$E\{X_n\} = X_0 M^n.$$

The first moment matrix is therefore in some sense analogous to the transition matrix of a Markov chain. In fact, Harris (1963, Section 12.2) remarks that if the matrix M is suitably normalized to make it a stochastic matrix, then it becomes the Markov transition matrix of a process which he calls the *expectation process*.

A number of relevant theorems are reviewed in Harris (1963). I shall only mention here the result that as far as the asymptotic behaviour of the system is concerned, when $n \rightarrow \infty$, the vector random variables X_n / ρ^n converge with probability 1 to a vector random variable W . The length of W is truly random but its direction, in case $W \neq 0$, is fixed and is the direction of the positive left eigenvector v of the matrix M . This means that the total activity of the system divided by ρ^n converges to a random variable, but the relative proportions of the activities of various words approach fixed limits.

* The square brackets denote a matrix operation.

3. RELATING THE MODEL TO THE BEHAVIOUR OF SUBJECTS

Word association networks

In the free word association experiment, subjects are presented with stimulus words and asked to say or write the *first* word that comes into their minds. The kind of data generated by this type of experiment is too well known to need detailed description here (Palermo and Jenkins 1964; Russell and

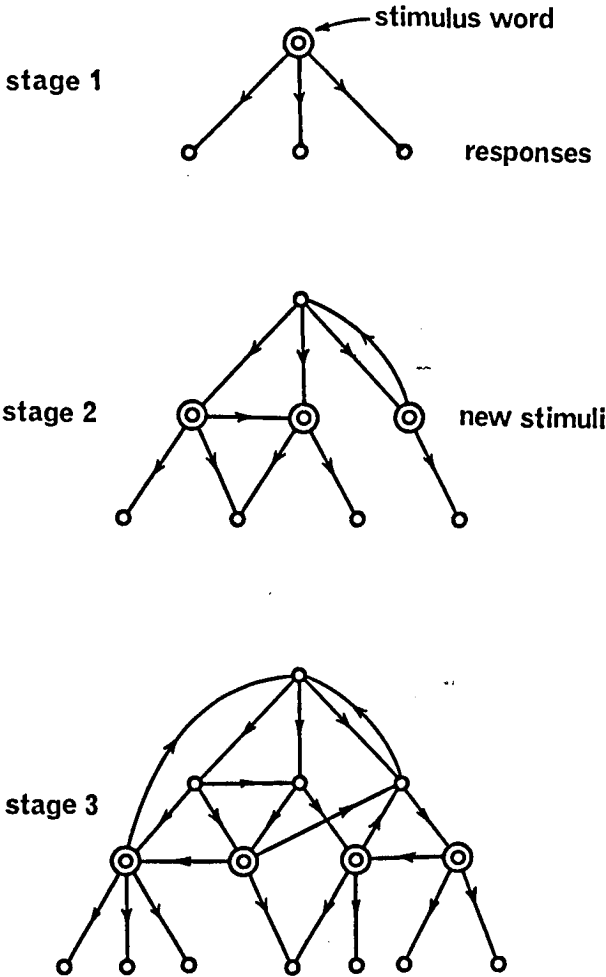


Figure 1. The growth of an association network

Jenkins 1954). These tabulations are known in the literature as word association *norms*. Essentially they are tables showing the frequency distributions of responses to a number of stimulus words.

A number of workers have pointed out recently that associative linkages

COGNITIVE PROCESSES: METHODS AND MODELS

among words not only form lists – as the usual presentation of the norms would lead one to conclude – but that they form complex networks (Deese 1965; Pollio 1966). That verbal habits form complicated structures has been realized for some time, as the terminology of associative gradients, response hierarchies and the like shows. It is only very recently, however, that some precise quantitative formulations of these ideas have been reached (Kiss 1965a; Pollio 1966).

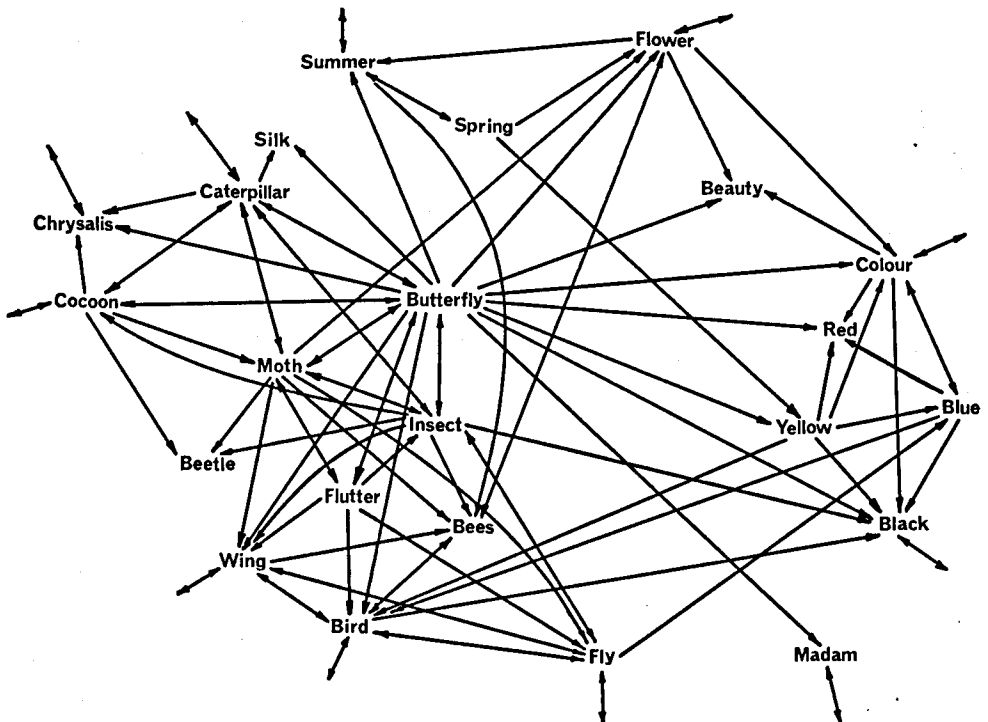


Figure 2. Part of the association network around BUTTERFLY

What I mean by a word association network is illustrated by figure 1. This shows the successive stages in the process of growing an association network from a single *root* point. In the first stage the word assigned to the root point is used as a stimulus in a word association experiment. In the second stage the responses of stage 1 are used as stimuli. This leads to linkages being formed between some of the words which are already in the net, and also to the introduction of new words. In stage 3 this is carried one step further. The structure obtained is a *directed linear graph*. It is composed of *nodes* (which represent words) and *directed lines* (arcs) bearing arrows (which represent an associative connection between the two words connected by the line). In figure 1 the nodes which are used as stimuli in any stage are marked by a ring. The process

of growing the network can be continued indefinitely, or at least until the vocabulary of the subjects or of the language is exhausted. Intuitively it can be seen that the number of words in the network grows approximately exponentially.

Such networks can be grown both for individual subjects and groups of subjects. Some suitably chosen *value* can be assigned to the arcs. In the case of a group, the frequency of occurrence of the response, or its probability estimated by its relative frequency in the group, is a convenient measure. If this is done, the graph resembles the transition diagram of a Markov chain.

An example of an association network is shown in figure 2. This network was derived by using 50 subjects and shows only some of the words obtained in the second stage of growth. It has been drawn in a way which emphasizes the existence of clusters formed by semantically inter-related words. This aspect of the graph has been studied in an earlier paper and will not be discussed further here (Kiss 1965b).

	Butterfly	Moth	Insect	Wing	Bird	Fly	Yellow	Flower	Cocoon	Colour	Blue	Bees	Red	Black	Chrysalis	Silk	Beauty	Madam	Beetle	Flutter	Spring	Summer	Caterpillar	
Butterfly		9	4	6	1																			
Moth	8		2	1		1						2												
Insect	2	1		1		8			1			6		1					4					1
Wing						24	8					2												
Bird					2		15						1	1										
Fly			5	4		1																		
Yellow							1			1	3		3	2										
Flower										1		1						3					1	
Cocoon	5	5																						9
Colour								1			8		7	2			1		1					
Blue					4		2			3			3	1										
Bees						5		2																
Flutter	10	5	1	3	4	4																		
Spring					1		1	1																10
Summer												1												2

Figure 3. The value matrix of the network shown in figure 2

There is of course an intimate connection between matrices and graphs (Harary *et al.* 1965), and the same information is shown in matrix form in figure 3. The rows of the matrix correspond to the stimuli and the columns to the responses. An entry at the intersection of a row and a column indicates that a line joins the nodes corresponding to the stimulus and the response, bearing the value appearing in that cell of the matrix. I shall call this the *value matrix* of the graph. If one replaces all nonzero entries of the value matrix by 1s the so-called *connection matrix* is obtained. This only shows that there is a link between two nodes of the corresponding graph, but does not indicate its 'strength'. Such matrices have been used by Pollio (1966) to

evaluate the 'cohesion' among a set of words. Similar ideas were used by the author in two earlier papers (Kiss 1965a, 1965b). Other uses of graph theory in the context of word association networks are discussed in more detail in Kiss (1968).

It will be perhaps useful to digress now from the main argument and to reflect on the significance of conceptualizing associative connections within the context of networks. I was first led into a consideration of networks as a possible model for word association phenomena by the observation that many of the words which appear in the low frequency range of a normative tabulation of responses for a stimulus word are themselves responses to words which appear higher up in the frequency distribution. An example will make this clearer. In the norms for the word *butterfly* one finds the word *plane* or *aeroplane* occurring with a rather low frequency. The word *plane* usually occurs with a fairly high frequency as a response to stimuli like *wing*, *fly*, *flying*, *flight*, etc., all of which usually appear in the norms, with higher frequencies than *plane*, as responses to *butterfly*. One is led to the idea of lower frequency responses being reached *via* higher frequency responses through a random walk in an associative network. The difficulty lies in the fact that this random walk *cannot be a conscious process*, since the subject is asked to report the *first* word which occurs to him, and in most cases there is no reason to doubt that he is complying with the instructions. One must therefore make this into an unconscious process.

A second difficulty lies in the fact that there is no simple criterion for stopping a random walk of this kind. The only assumption one could make is that the words reached in the course of the random walk are, still unconsciously, examined from the point of view of various criteria (such as being a subordinate, having similar meaning, being an opposite, etc.) and the process stops at the first acceptable word. Essentially this view was advocated in one of my earlier papers (Kiss 1965a). What is wrong with this view is of course the assumption that such detailed logical and semantic decisions could be taken at the unconscious level. This picture of the word selection process can only be maintained if we permit the subject to suppress, *consciously*, some of the early responses which occur to him, and this model does indeed fit many experimental paradigms (e.g. Rosenberg and Cohen 1966). As far as the unconscious decision processes are concerned I would like to keep things simple and not to permit anything more complicated than can be implemented in the form of some unidimensional threshold or comparison process and possibly some affective evaluation.

Added to the argument so far, there are two rather interesting findings in the literature, both showing that responses which were not emitted during the experiment but could have been emitted, have a strong influence on the actual response which *is* emitted. The first of these findings was reported by Pollio (1963) and consists in the fact that there is a positive relationship between the rank of a word in a normative table and the number of other

words in the table which elicit that word as a response. Converting this finding into network terminology, we can say that *the more numerous the indirect pathways between a stimulus and a response in a network, the higher the probability of that response*. So the probability of a response seems to depend on the number of different ways it can be reached by walking about in the network. Yet only one of these pathways could have been used in finding the word if it is emitted when it is reached for the first time. I am again forced to conclude from this that it would be unreasonable to represent the word selection process as simply a random walk over the associative network until the correct word is hit for the first time.

The second piece of evidence comes from Treisman (1965). In this paper she reports that when subjects guess a missing word in a sentence or in an approximation to English text, the latency of the guess depends as much on the possible responses, which have been made by other subjects, as it does on the probability of the response which is actually made by the subject. This finding would again be incompatible with a simple random-walk conception. What is required is a mechanism in which all possible responses can exert an influence on the selection of the eventual response. The model described in the previous section has this property.

Word selection in individuals and groups

As far as a single trial of a word association experiment with a single subject is concerned, the model described gives the following account of events. When the stimulus word is presented a certain amount of activity is produced in one or some of the word representations of the system. The stimulus, therefore, produces a specific starting state. The system is now allowed to evolve through free transitions for a certain amount of time. During this interval the activity levels vary. At some instant the subject makes a decision to emit a response word. The probability of choosing a word is determined by the relative level of activity of that word. Since this decision is a stochastic one, variability of behaviour is a characteristic feature of the model, in accord with our experience with verbal behaviour. Mathematically, the expectation process describes the behaviour of the single subject.

Under normal conditions the stochastic nature of the transmission of activity from one word to another will not manifest itself, owing to the very large number of individual elements which can be expected to participate in representing a word. The mean transmissions between words will therefore be stable and behave in a more or less deterministic fashion. It can be expected, however, that random fluctuations in the transmissions will have appreciable effects under abnormal conditions, such as subliminal presentation of stimuli, brain damage, etc.

When the word association experiment is repeated with a group of subjects, using the same stimulus word, a number of possible factors contributing to the variability of the results becomes apparent. The basic transmittance

structure can be different from one individual to another. There is some evidence in the literature that there is at least a strong similarity between the associative networks of individuals. For the time being let us adopt the homogeneity of transmittance structure hypothesis as a deliberate simplification.

The response probabilities in the word association experiment depend on the time spent on selecting the response. Empirically this phenomenon is known as Marbe's Law which has been demonstrated by several investigators (Woodworth and Schlosberg 1955). In the model, time corresponds to the number of transitions gone through by the branching process. As we have seen in the previous section, the law of this change is described by the successive powers of the transition probability matrix of the expectation process. The different time intervals (latencies) spent by different individuals in a group on selecting their responses will therefore also contribute to the variability of the results. There is some evidence (Woodworth and Schlosberg 1955) that the distribution of latencies is rather close to the lognormal. The fact that this kind of distribution could be expected to appear in a branching process model will be discussed elsewhere.

Most of the available normative word association data is based on groups of subjects and not individuals. Due to the fact that the subjects are not constrained in the time they spend on selecting a response, such normative data is always a composite of responses, which have been selected by letting the word store go through a variable number of transitions. The contribution of different lengths of transition chains to the total results obtained from a group of subjects is dependent on the distribution of latencies. This is known to be nearly lognormal, so that the contribution of shorter latencies is dominant, and that of the longer latencies diminishes in a roughly exponential fashion.

This suggests some ways of testing the model. If a word association network is obtained from a sample of subjects, then an approximation to the one-step network could be obtained by eliminating from the value matrix all entries which fall below some arbitrary threshold value. According to Marbe's law, the remaining high probability values correspond to short latencies, and therefore also to a small number of transitions. The error in this approximation is due to three factors. First, by cutting off all the low probabilities one also cuts off some one-step connections which happen to have a low value. Second, the remaining high probability values are still a composite of one-step and indirect connections. As discussed below, it is possible to eliminate the contribution of indirect connections. Third, there will be an error component due to the limited accuracy with which the probability values can be estimated from a sample of a given size. The collection of network information from a large sample is very expensive. For this reason it is of great interest to see whether an approximation to the one-step matrix could be obtained from a small sample. This turns out to be the case, as the use of a small sample has the effect of the threshold operation described, since most of the information

on the small probabilities is lost. The error of estimating the actual values will be large. Still, as is shown below, if the small-sample network is used as an approximation to the one-step network and the n -step network is obtained from it, the resulting probabilities correlate well with the values obtained directly by using a large sample to provide values for some of the nodes.

Clearly, a method for determining the basic one-step connection structure of the word store for an individual, or for a group in a statistical fashion, would be of great interest. It seems rather difficult to approach this problem empirically, although putting time pressure on the subjects, or selecting responses which were given with a short latency are approaches currently being explored. It is, however, possible to work back to the one-step structure from the normative empirical data with an approximation method, to be described in more detail elsewhere. Essentially this method amounts to the iterative removal of the contribution from indirect transition chains until a structure is obtained which is capable of regenerating most of the information contained in the original data.

Some empirical tests of the model

Two kinds of empirical tests have been carried out on the model so far. One of them is the test discussed in the previous section and is essentially an attempt at predicting the word association norms of a large sample of subjects from association network data obtained from a smaller sample. The second test was an attempt at predicting the word association response probabilities to stimuli, which consist of several words, from a knowledge of small-sample norms for the component words. The results of the first of these tests and the procedures used will now be described. The results of the second test have been described in a separate paper (Kiss 1967b).

Prediction of large-sample norms from small-sample networks

Word association networks were grown from four words as rootpoints: BUTTERFLY, WHISTLE, MUSIC and SLOW. As a short cut the lists of high and low frequency associated to these words listed by Deese (1959) were used instead of the first stage of growing the net. Deese lists 30 responses to each of these words. These $4 \times 30 = 120$ words were interspersed with an additional 140 irrelevant words (also taken from Deese's paper) by randomizing the order of the resulting 260 words. Two lists of 130 words each were then typed and duplicated for use in the collection of association responses. Two separate samples of 50 subjects each were used to obtain responses to each of these lists. The subjects were undergraduate students at Birmingham University. The results were sorted by computer. Parts of one of the resulting networks were shown in figures 2 and 3. For a sample of 50 subjects a matrix of the kind shown in figure 3 has about 500 columns for 30 stimuli. In order to reduce computer time and storage requirements, the matrices were arbitrarily

restricted to 60 columns. The 30 columns corresponding to the stimulus words were always retained if they occurred as responses.

The resulting four 30×60 matrices formed the input data to a computer program which was written to evaluate 'flow graph transmittances'. The use of flow graph methods in the context of word association networks is discussed in detail in another paper (Kiss 1968). Suffice it to say here that these methods are equivalent to the calculation of $M + M^2 + M^3 + \dots + M^{NMAX}$, where M is the input data matrix and $NMAX$ is a parameter in the program,

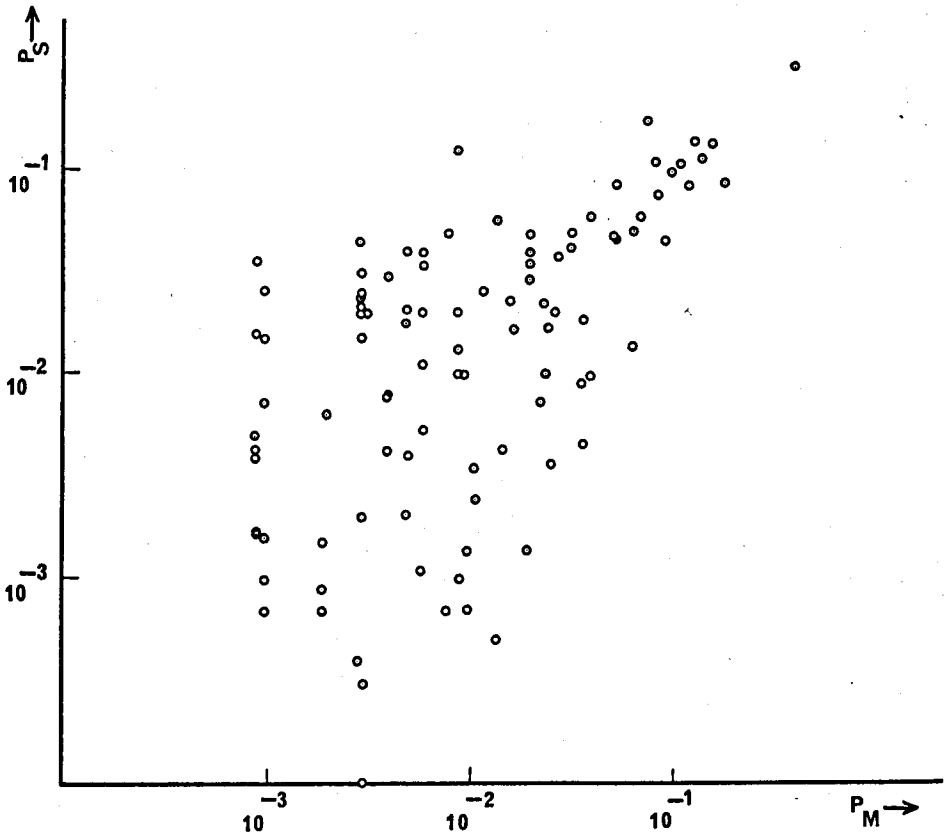


Figure 4. Scatterplot of the response probabilities obtained by simulation of the model (p_s) and from words association norms (p_M)

which determines the length of the longest pathway between stimulus and response, which is taken into account during the calculations. Instead of calculating the whole of the resulting matrix, however, flow graph methods enable one to evaluate any cell of that matrix independently.

The flow graph transmittances were computed for each of the words BUTTERFLY, WHISTLE, MUSIC and SLOW with $NMAX=1, 2$ and 3 . This resulted in sets of words which can be reached in 1, 2 or 3 steps from the

	P_S	P_{M_1}	P_{M_2}
STOP	·1403	·1298	·0560
BLOW	·0864	·0555	·0600
TRAIN	·0678	·0882	·1060
SHRILL	·0518	·0307	·0300
NOISE	·0504	·0723	·0660
DANCE	·0472	—	—
MUSIC	·0428	·0079	·0110
SONG	·0403	·0327	·0370
SING	·0396	·0614	·0540
SOFT	·0325	—	·0010
SHOUT	·0307	·0059	·0030
BIRD	·0304	·0039	·0070
WIND	·0279	·0009	·0020
TUNE	·0269	·0485	·0320
LOUD	·0234	·0168	·0400
GO	·0233	—	—*
HAPPY	·0215	·0049	·0090
PIERCING	·0197	—	·0030
SOUND	·0189	·0386	·0610
BOY	·0172	·0257	·0370
WOLF	·0172	·0178	·0230
GAME	·0172	·0009	—*
CHOIR	·0144	—	—
NOSE	·0122	—	—*
DOG	·0090	·0366	·0180
PIANO	·0072	—	—
WATER	·0059	—	—*
VOICE	·0054	—	—*
RED	·0054	—	—*
BARK	·0052	·0009	—*
RECORD	·0049	—	—
GIRL	·0046	·0386	·0430
TRAFFIC	·0044	—	—
QUIET	·0044	—	—*
EAR	·0034	·0009	—*
ANIMAL	·0034	—	—
GAY	·0032	—	—*
MOUTH	·0032	·0148	·0180
FOOD	·0029	—	—
TALK	·0029	·0039	·0020
LOW	·0029	·0009	—*
SHORT	·0017	·0009	—*
PIPE	·0016	·0019	—
HIGH	·0011	·0059	·0070
SIGNAL	·0009	·0019	—*
SHARP	·0007	·0079	·0130
GONG	·0007	—	—
WOMAN	·0007	·0099	·0050
LAD	·0004	—	—*

Table 1. Responses to the stimulus word WHISTLE

stimulus, together with integers showing the transmittance (the strength of the connection) value when all pathways up to $NMAX$ in length are taken into account. One such set for WHISTLE with $NMAX=3$ is shown in the first column of Table 1. Hereafter such values will be denoted by p_S when the transmittances are normalized so that they sum to 1. These normalized transmittances are shown in column two of Table 1, labelled p_S . A complete matrix of p_S values will be denoted by S .

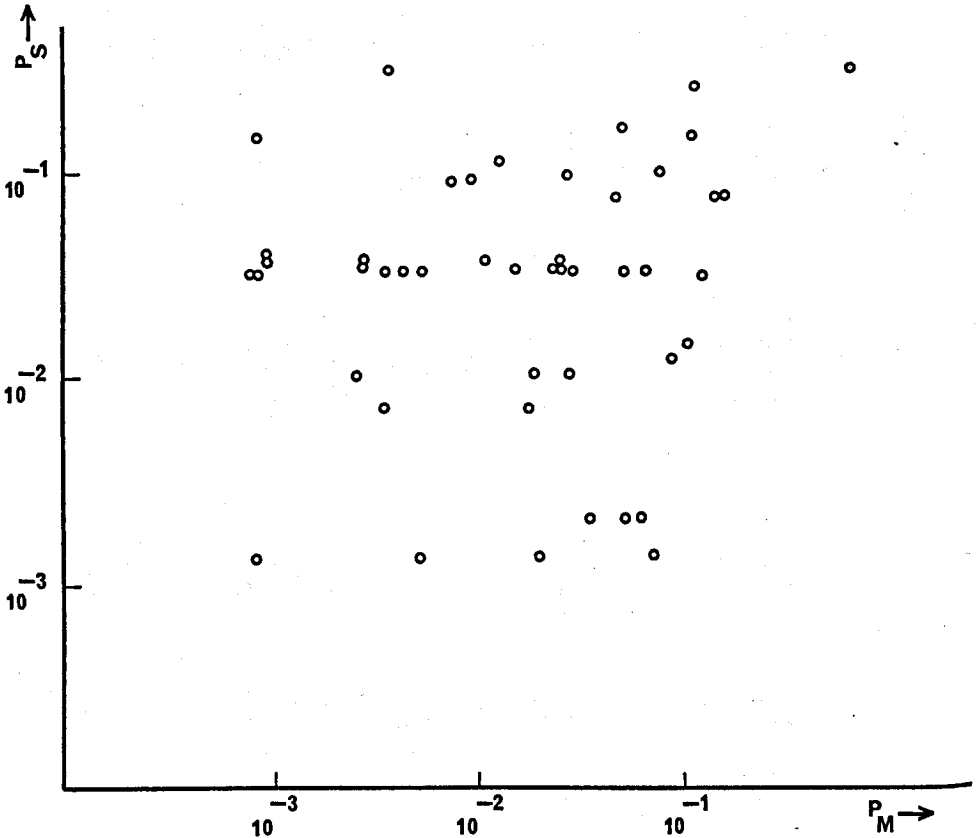


Figure 5. Scatterplot of p_S from the raw input data against p_M

The row of the matrix S corresponding to WHISTLE (shown in Table 1) represents the simulated response probabilities for this word. These values should then correlate with the probabilities estimated from, e.g. the Minnesota norms. In Table 1 the column headed p_{M1} is based on the 1954, and the column headed p_{M2} on the 1964 Minnesota norms.

Ideally the p_S values should be equal to the p_M values, but due to the simplifications involved, one should not expect more than a linear relationship. The results of such a comparison are displayed in figure 4. It can be seen that

a linear relationship does emerge, and the points are fairly symmetrically distributed around a line at 45° to the co-ordinate axes. The scatter increases towards the lower probabilities, and it should be noted that the lowest p_M value which could occur is about 0.001 since the normative data is usually collected from subject samples of the order of 1000. The calculated p_S values do, of course, go below this value. The correlation between p_S and p_M is 0.578.

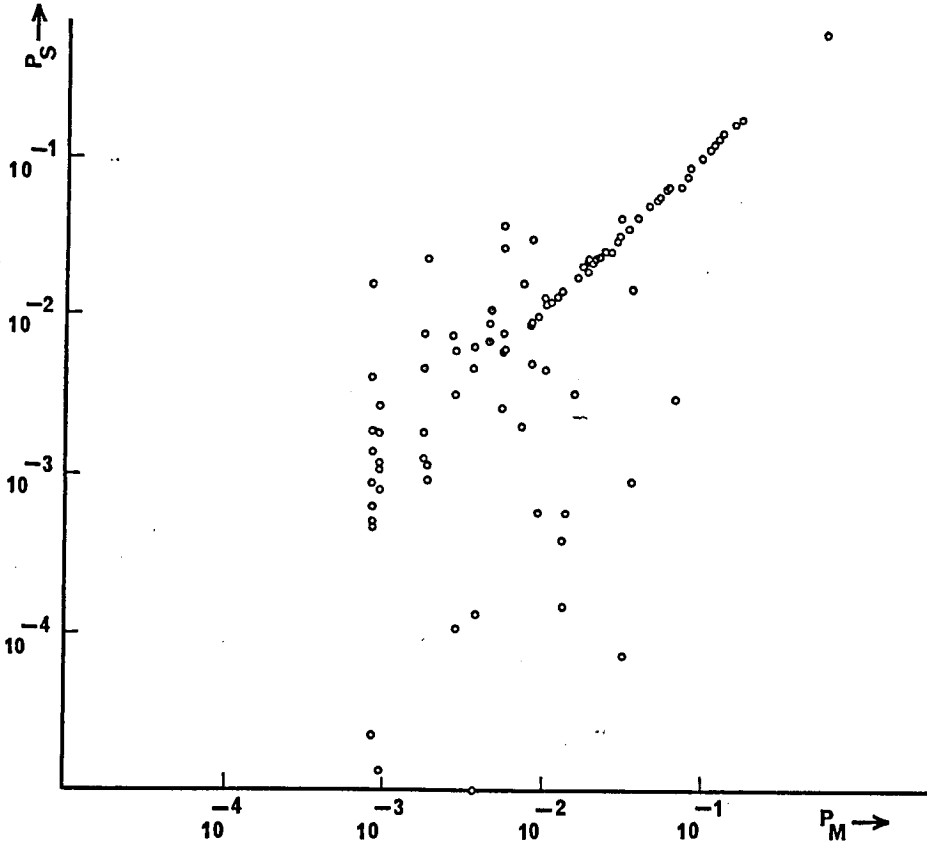


Figure 6. The effects of using an algorithm for obtaining the basic one-step transmittances. The algorithm was applied to the four root words only

In comparison, figure 5 shows the scatter when the p_S values are taken from the raw data which serves as an input to the flow graph solving program. No systematic relationship between the p_S and p_M values can be found in this case. From this one can conclude that the raw input data contains very little information about the correct normative probability values. This information is contained mostly in the structure of the associative network which is mapped out in obtaining the raw data.

The effects of using the algorithm for deriving the basic one-step transmittance

values from the raw data are shown in figure 6. Here the p_s values are based on the resulting network, again using maximum pathways of length three. The correlation is in this case 0.669. It should be noted that the algorithm was used to obtain the one-step transmittances for the root words only, and not for the complete networks. This accounts for the remaining scatter at the lower probabilities.

Additional light is thrown on these results by noting that the reliability of the Minnesota norms is 0.9 for the first 30 responses to the word WHISTLE. This was obtained by correlating the 1954 and 1964 versions. The reliability of the p_s values is also 0.9 on the basis of correlating the p_s values obtained by using the two samples of 50 subjects. The largest expected correlation would therefore be 0.9.

Taking into account the simplifying assumptions which have been made, and also the rather unsatisfactory method which has been used in collecting the data (presenting long unrandomized lists of stimuli), it can be concluded that these results are in favour of the model presented.

The correlation of response probabilities obtained by simulation with those obtained from word association norms is not, however, the only way of evaluating the model. Another criterion can be set up on the basis of evaluating the model as an information retrieval system.

A retrieval system can make errors of two kinds. It can fail to retrieve items which are relevant to the query, or it can retrieve irrelevant items. Indeed, given that there is some way of determining whether any given item is relevant or not, it can belong to one of the following four cases: (1) a 'hit' if it is relevant and it is retrieved; (2) a 'false drop', if it is irrelevant and it is retrieved; (3) a 'miss', if it is relevant and it is not retrieved; (4) a 'correct rejection', if it is irrelevant and it is not retrieved. On the basis of the probabilities that any of these events will occur, one can define a number of ways of evaluating the effectiveness of a retrieval system. One of the most convenient methods was proposed by Swets (1963) in terms of statistical decision theory. The proposal amounts to using the two well-known parameters of signal detection theory, d' and β , as measures of retrieval system effectiveness and query breadth, respectively.

In attempting to apply signal detection theory to the evaluation of the word store model, one way of proceeding is to assume that relevance is decided by whether a given response word occurs in the association norms. The effectiveness of the model is then evaluated by drawing up a four-fold table, divided according to whether words are retrieved or not, and whether they occur in the norms or not. The entries in such a 2×2 table would be the number of words which fit into any of these four classes. From such a table d' and β can be determined. One can also obtain ROC curves by setting up such tables at successive threshold probability levels p_s .

I shall not attempt a detailed application of these ideas here. To convey some idea of how the model performs, Table 1 shows the list of words

retrieved by the model when $NMAX=3$ for the stimulus WHISTLE. Here p_S has its usual meaning, p_{M1} is the probability estimated from the 1954 Minnesota norms, and p_{M2} is the probability from the 1964 norms. It can be seen that there are 50 words which are retrieved by the model. It should be remembered that the input data was arbitrarily truncated to 60 responses from about 500. It is not therefore known at the moment how many words can be reached in 3 steps from the stimulus, using the complete network. Neither is it known how many words among those which would be retrieved beyond the 50 shown would also occur in the norms. However, it can be seen that at the lower probabilities the agreement between the norms themselves is not very good, and therefore it would not be surprising if the model did not perform very well with these words. A star in the p_{M2} column means that although the word does not occur in the sample of 1000 undergraduates, it does occur somewhere in the other age samples. The 1964 Minnesota norms do not list frequencies of 1 unless they occur as responses in some other sample with a frequency greater than 1. For some words, therefore, the probabilities are not known, only the fact that they do occur. It is apparent that out of the 50 words retrieved by the model (one of which is the stimulus word WHISTLE itself, with a probability of .0882, not shown in the table) 30 occur in the 1954 norms, and 40 in the 1964 norms.

In conclusion it can be said that the ability of the model to recover the highly relevant words without too many irrelevant ones is fairly good in view of the simplified manner in which it has been simulated.

Acknowledgements

Most of the ideas related to the application of stochastic processes occurred to me while I was visiting the Experimental Programming Unit (now Department of Machine Intelligence and Perception) at Edinburgh in 1966. It is a pleasure to acknowledge stimulating discussions with Donald Michie, Rod Burstall and particularly with Robin Popplestone.

I also wish to acknowledge advice on stochastic processes from Professor D. R. Cox of Imperial College, and helpful discussions with Fergus Craik at Birkbeck College.

The work has been partly supported by a grant from the Research Fund of the University of London.

REFERENCES

- Bharucha-Reid, A.T. (1960) *Elements of the Theory of Markov Processes and their Applications*. New York: McGraw-Hill.
- Bousfield, W.A. & Sedgewick, C.H.W. (1944) An analysis of sequences of restricted associative responses. *J. gen. Psychol.*, **30**, 149-65.
- Bullock, T. H. (1957) Neuronal integrative mechanisms. In *Recent Advances in Invertebrate Physiology* (ed. Scheer, B. T.). Eugene, Oregon: University of Oregon.
- Deese, J. (1959) Influence of inter-item associative strength upon immediate free recall. *Psychol. Rep.*, **5**, 305-12.

COGNITIVE PROCESSES: METHODS AND MODELS

- Deese, J. (1965) *The Structure of Associations in Language and Thought*. Baltimore: The Johns Hopkins Press.
- Fatt, P. & Katz, B. (1952) Spontaneous subthreshold activity at motor nerve endings. *J. Physiol.*, **117**, 109-28.
- Frishkopf, L.S. & Rosenblith, W.A. (1958) Fluctuations in neural thresholds. In *Symposium on Information Theory in Biology* (ed. Quastler, H.). London: Pergamon Press.
- Grundfest, H. (1959) Synaptic and ephaptic transmission. In *Handbook of Physiology, Section I; Neurophysiology*, vol. 1 (ed. Field, J.). Washington D.C.: American Physiological Society.
- Harary, F., Norman, R.Z. & Cartwright, D. (1965) *Structural Models*. New York: Wiley.
- Harris, T.E. (1963) *The Theory of Branching Processes*. Berlin: Springer.
- Kiss, G.R. (1965a) Computer simulation of word association processes: Theoretical design. Report No. 3 of the D.S.I.R. Project on Computer Simulation of Cognitive Processes. Birmingham, England: Education Department, Univ. of Birmingham.
- Kiss, G.R. (1965b) Clustering of words in association networks and in free recall. *Bull. Brit. Psychol. Soc.*, **18**, 7A.
- Kiss, G.R. (1967a) Networks as models of word storage. *Machine Intelligence 1* (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.
- Kiss, G.R. (1967b) A test of the word selection model using multiple-word stimuli in word association. MRC Speech & Communication Research Unit Report scu/7/67.
- Kiss, G.R. (1968) Words, associations and networks, *J. verb. Learn. verb. Behav.*, **7**, 707-13.
- Luce, R.D. (1959) *Individual Choice Behavior*. New York: Wiley.
- McGill, W.J. (1963) Stochastic latency mechanisms. In *Handbook of Mathematical Psychology*, Vol. 1. (eds Luce, R.D. Bush, R.R. & Galanter, E.). New York: Wiley.
- Palermo, D.S. & Jenkins, J.J. (1964) *Word Association Norms*. Minneapolis: University of Minnesota Press.
- Pollio, H.R. (1963) A simple matrix analysis of associative structure. *J. verb. Learn. verb. Behav.*, **2**, 166-9.
- Pollio, H.R. (1966) *The Structural Basis of Word Association Behavior*. Janua Linguarum, Series Minor, No. 51. The Hague: Mouton.
- Rosenberg, S. & Cohen, B.D. (1966) Referential processes of speakers and listeners. *Psychol. Rev.*, **73**, 208-31.
- Russell, W.A. & Jenkins, J.J. (1954) The complete Minnesota norms for responses to 100 words from the Kent-Rosanoff word association test Techn. Rep. No. 11, Contract N8-ONR-66216, Office of Naval Research and Univ. of Minnesota. 1954.
- Swets, J.A. (1963) Information retrieval systems. *Science*, **19**, 245-50.
- Treisman, A.M. (1960) Contextual cues in selective listening. *Quart. J. exp. Psychol.*, **12**, 242-8.
- Treisman, A.M. (1965) Effect of verbal context on latency of word selection. *Nature*, **206**, 218-219.
- Verveen, A.A. (1961) *Fluctuation in Excitability*. Amsterdam: Drukkerij Holland N.V.
- Woodworth, R.S. & Schlosberg, H. (1955) *Experimental Psychology*. New York: Holt.

The Game of Hare and Hounds and the Statistical Study of Literary Vocabulary

S.H. Storey

and

M. Ann Maybrey

The University, Liverpool

INTRODUCTION

In the early sections of this paper, a technique for enabling a digital computer to be taught to play a reasonably respectable game of Hare and Hounds is described. The results of a number of experiments with the techniques thus developed are described and discussed. The paper concludes with a discussion of the possibilities of using results of this type to test theories of the statistics of literary vocabulary.

THE GAME OF HARE AND HOUNDS

The game of Hare and Hounds is a board game played on the usual 8×8 chessboard. Four hounds and one hare are required. The relative positions of the pieces at the start of a game, in the version of the game used for the purposes of the paper, are shown in figure 1: that is to say, only the white squares of the board are used, and the bottom right hand of the board is assumed white. Moves are made as in draughts, the hounds moving only forwards, that is, from the bottom to the top of the board, whereas the hare may move either forwards or backwards. The object of the hare is to reach the bottom of the board. The object of the hounds is to prevent this and eventually to trap the hare so that it cannot move. The hare may start from any of the top four (white) squares of the board, but normally starts from the centre.

The game is not as simple as it would appear, and the authors know of no concise algorithm for playing either the hare or the hounds. The level of difficulty of the game is substantially below that of draughts, but considerably higher than that of such games as noughts and crosses where a definite algorithm is known to exist (*see* Booth and Booth 1953).

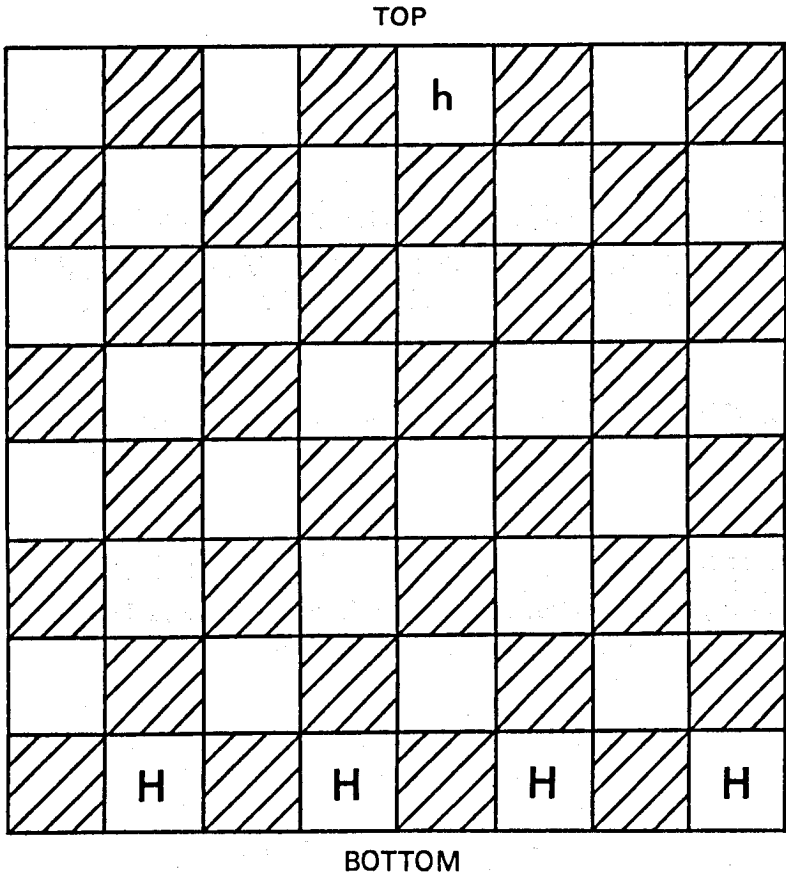


Figure 1. Starting position

THE PATTERN RECOGNITION PROBLEMS

Experience of the game has shown that, on a board of this particular width, the hounds have a distinct advantage, and, in the hands of a player of even moderate competence, should probably win every time. Accordingly, it was decided to teach the computer to play the hounds. The method adopted was that of teaching the computer to recognize a position and remember (from its coaching alone) which move the hounds should appropriately make. However, the number of positions which arise from the distribution of four identical hounds and one hare on the thirty-two available white squares is $(32 \times 31 \times 30 \times 29 \times 28) / (4 \times 3 \times 2) = 1,006,880$. In spite of the fact that roughly half of these positions would not be needed (those for which the hare had succeeded in moving behind the hounds, at which point the game would normally be abandoned), the technical problems involved in storing and searching such large amounts of data could safely be described as forbidding.

At this point two observations were made which at once so greatly reduced the problem that the implementation of a game-playing program became trivial.

First, it was noticed that the move made by a human player in response to the position shown in the top half of figure 2 was frequently the same, *relative to the position of the pieces*, as that to the position in the bottom half. (The two sets of pieces are shown on the same board to save space.)

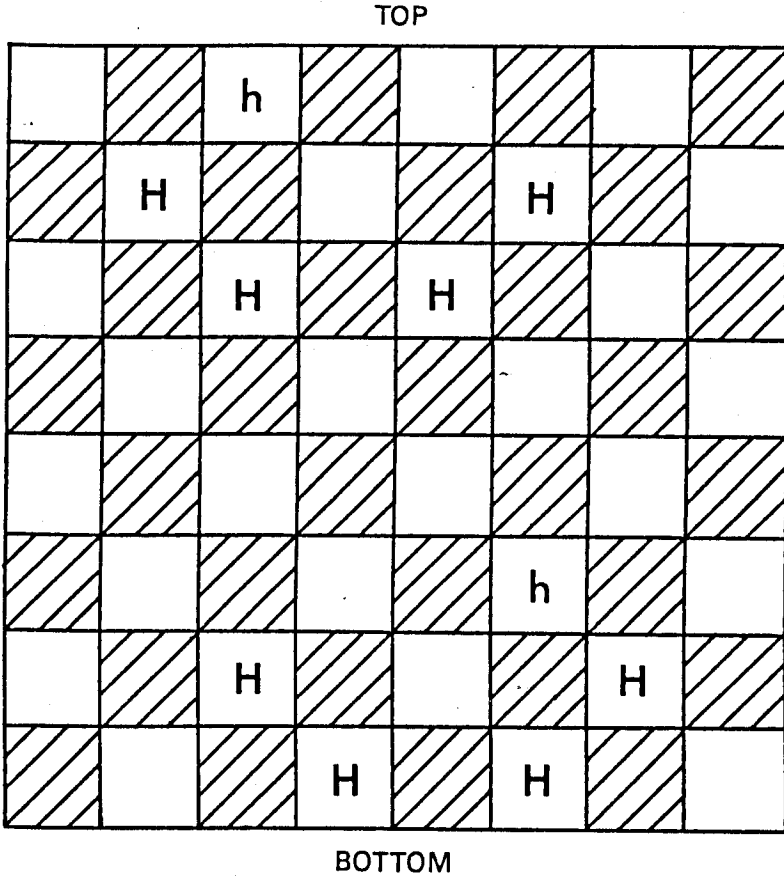


Figure 2. Equivalent relative position

In figure 2, if the correct move in the lower half is to move the rightmost of the two hounds on the bottom row, then the corresponding correct move in the top half is to move the leftmost of the two hounds nearest the bottom of the board. Further, it was found that by numbering the white squares in the serpentine fashion shown in figure 3, any position can be reduced to its equivalent baseline position by simply subtracting the highest possible multiple of four from the numbers specifying the position. Thus, the position

COGNITIVE PROCESSES: METHODS AND MODELS

in the top half of figure 2 is described as having hounds at 21, 22, 25, and 27, with the hare at 29. Subtracting 20 from each of these numbers (a larger multiple of four would yield negative values) leads to the configuration 1, 2, 5, 7, and 9, which is the description of the configuration in the lower half of figure 2.

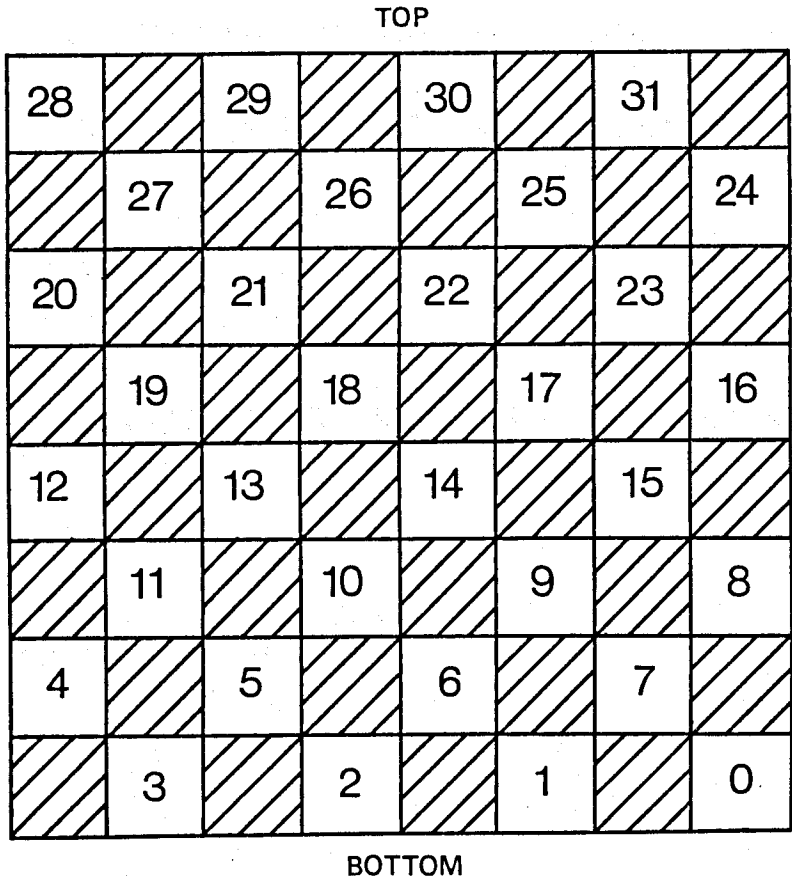


Figure 3. Numbering scheme adopted

It was accordingly assumed that the computer should base its response on the equivalent baseline position only. This assumption had the advantage of reducing the maximum number of patterns to be stored by a further factor of six to seven; such an assumption, however, ignored the possibility of an 'edge effect', where the top edge of the board might influence those moves made in its proximity.

Although this assumption greatly reduced the number of configurations to be considered, there still remained the possibility of having to store some 100,000 or so positions. The second observation made, however, suggested

that the actual number needed would in fact be very much smaller than this. It was noted that, in the case of a human player, once a pattern was recognised and a particular move was found to be successful, the same response was made on all subsequent occurrences of the configuration. Further, as a human player playing the hounds gained experience and control, the number of configurations which he *allowed* to arise appeared to be very small. This seemed partly due to the fact that in retaining control, the experienced player could only permit himself a limited number of interlocking responses, and partly that, in many cases, the position of the hare was irrelevant.

Accordingly, a study was begun on the assumption that only a few hundred configurations would be needed, which of course meant that the whole program could be held within the high-speed memory of the computer.

PROCEDURE

As soon as the concepts described in the previous section were adopted, the techniques necessary to 'teach' the computer to play, and to use the computer to play actual games became very simple. So simple, in fact, that it was found unnecessary to use a computer at all, and a substantial portion of the results presented below were obtained with pen and paper. They will continue to be discussed, however, in terms of a computer.

To start the process, the computer is 'shown' a sample game between two human players (in which the hounds have won). The computer 'studies' the patterns and responses made during the game and 'assimilates' them. In practical terms, a list of the patterns and corresponding moves made during the game is read into the computer. The computer then reduces all the patterns to the equivalent baseline patterns and makes a list of them, ordered, in the first instance, by frequency of occurrence, and, within each frequency, numerically by hound patterns. The form of the list held in memory is given in Table 1.¹ This table shows the state of the computer's memory after assimilating one and five coaching games with each of its two instructors. The information consists of (i) a hound pattern (reduced); (ii) the source of the hound move (*S*); (iii) the destination of the hound move (*D*); and (iv) the positions in which the hare has been observed. In the original computer program, the hare positions, which are also reduced to their baseline equivalents, were packed into a single word, but experience has shown this not to be necessary.

Once the computer has available a memory of the game, that is, a list of baseline patterns and responses, it is in a position to begin playing actual games. The current position of the hounds and hare is fed into the computer. This is then reduced to the equivalent baseline pattern. The computer then searches its memory of the game for the pattern and, on finding it, prints out the move which it has learned to correspond to this pattern (suitably re-converted to match the current position values). There are exceptions to this

¹ The tables employed in the (manual) preparation of the memory included the frequencies.

INSTRUCTOR B

INSTRUCTOR A

after 1 game				after 5 games				after 1 game				after 5 games			
hounds	hare	S	D	hounds	hare	S	D	hounds	hare	S	D	hounds	hare	S	D
1 2 3 7	9 10 22	1	6	1 2 3 7	9 10 14 22 23	1	6	1 2 3 7	10 20	1	6	0 1 2 3	4 6 9 10 14 25 26	0	7
0 1 2 3	6 10 25	0	7	2 3 6 7 5		7	9	0 1 2 3 7		3	5	1 2 3 7	10 15 20 21 22	1	6
2 3 6 7	14 18	2	5	2 3 6 9 10 11		2	5	0 1 2 3 26		0	7	3 5 6 7	11 13 14 15 18	3	4
2 3 6 7	5	7	9	0 1 2 3 5 6 10 25 26		0	7	0 1 2 5 9		0	7	2 3 6 7	8 10 13 17 18	2	5
2 5 6 7	14	5	11	2 3 6 7 14 17 18		2	5	0 1 6 14 9		1	7	0 1 2 5	9 14	0	7
2 3 6 9	10 11	2	5	2 3 6 7 10 13 14		3	4	0 6 7 14 15		6	9	2 3 6 7	5	7	9
2 6 7 11	10	2	5	2 5 6 7 13 14		5	11	0 7 9 14 8		9	15	2 3 6 9	10 11	2	5
3 5 6 7	10 13	3	4	2 6 7 11 9 10		2	5	1 2 5 6 10		1	7	0 1 2 3 7		3	5
1 2 5 6	9	1	7	2 6 7 11 9 10		2	5	1 2 5 6 10		1	7	2 5 6 7 13		5	11
1 2 5 7	9	1	6	3 5 6 9 4 12		5	11	1 2 5 7 6		7	9	2 6 7 11 10		2	5
1 2 3 7	6	3	5	0 1 2 5 7		2	6	1 2 5 9 10		1	6	0 1 6 14 9		1	7
3 5 6 9	14	5	10	0 1 5 6 8 9		0	7	2 3 6 7 5		6	10	0 6 7 14 15		6	9
3 5 6 9	4	5	11	1 2 3 6 10 11		1	7	2 3 6 7 18		2	5	1 2 5 7 10		1	6
3 6 9 10	18	3	5	1 5 6 7 14 15		6	10	2 3 7 10 11		2	5	1 2 7 10 9		1	6
				1 5 7 10 8 9 18		1	6	2 5 6 7 13		5	11	2 3 7 10 11		2	5
				3 5 6 9 13		3	4	2 5 6 9 14		5	10	3 5 7 10 13		7	9
				1 2 5 6 9 10		1	7	2 6 7 11 10		2	5	3 5 9 10 18		9	14
				3 5 6 9 14		5	10	2 6 9 10 18		2	5	3 5 10 14 13		14	18
				1 2 5 7 9		1	6	3 5 6 7 13		3	4	3 5 10 18 11		3	4
				3 6 9 10 18		3	5	3 5 7 10 13		7	9	0 5 6 11 9		0	7
				0 1 2 3 7		3	5	3 5 9 10 18		9	14	0 7 9 14 8		9	15
				0 1 2 5 9		0	7	3 5 10 14 13		14	18	1 2 3 6 14		1	7
				1 2 3 7 6		3	5	3 5 10 18 11		3	4				

1 2 5 7 6
1 2 5 9 10
1 5 6 11 9
2 5 6 9 14
2 6 9 10 18
3 6 9 10 17
3 6 9 11 19
3 6 10 15 14
3 9 10 15 18

7 9
1 6
1 7
5 10
2 5
9 15
3 5
6 9
3 5

1 2 3 7 6
1 2 3 7 11
1 2 5 6 10
1 2 5 7 6
1 3 5 7 13
1 6 7 10 14
1 6 9 10 18
1 6 9 13 14
1 7 9 14 8
1 9 10 13 17
2 5 6 9 10
2 5 6 9 13
2 5 6 9 14
2 5 6 14 13
2 6 7 10 14
2 6 9 10 18
2 6 9 11 10
2 6 11 14 10
3 5 6 9 4
3 5 6 9 13
3 5 6 9 14
3 6 9 10 17
3 6 10 15 14
3 9 10 15 18

3 5
2 5
1 7
5 10
1 6
7 9
10 13
6 10
9 15
1 7
9 14
5 11
5 10
5 11
2 5
2 5
2 5
2 5
5 11
3 4
5 10
9 15
6 9
3 4

Table 1

behaviour. First, if the computer cannot find the required hound pattern in its memory of the game at all it informs its opponent of this and asks for instruction. Second, if the hound pattern is listed, but the hare position is not, the computer selects the response corresponding to the most frequently occurring configurations with the correct hound pattern. If a variety of hare positions are associated with the listed hound pattern, the computer notes that it is uncertain of the correct move, prints the suggested move, and asks for instruction. If only a single position has so far been associated with the listed hound pattern, the computer notes great uncertainty, prints the suggested move and again asks for instruction.

This technique for playing has worked realistically in practice. In the event of the computer having no record of the configuration at all, the move is made for it by the instructor and the game then continues. In the cases so far, when mere uncertainty has been noted, the suggested move has almost invariably been unsatisfactory to the instructor. In cases of great uncertainty a substantial proportion of the suggested moves have been rejected by the instructor and another move substituted. It may be of interest that, using this technique, the computer, with a memory of only one game (the first shown in Table 1), successfully defeated a consortium of two relatively inexperienced human players without assistance.

Normally, however, the computer has played its instructors. If the computer wins the game (and, with the help of its instructors, it invariably does) the resulting game is assimilated into its memory of the game in much the same manner as was the starting game. As has been noted, the computer is capable of defeating human players without help after assimilating a single game. After assimilating five it is probably a better player than its instructors.

EXPERIMENTAL RESULTS

The game of Hare and Hounds, although straightforward, is of sufficient complexity for the possibility of more than one successful method of playing the hounds to be appreciable. In this case, if the computer were to be taught to play by two different instructors, there should be a measurable difference in the style of game played by the computer in each case. If such differences arise, the process described here might form the basis for reasonably precise studies of the influence of the *teacher* on the learning process. The perfect memory of the computer is an advantage for such a study, since it avoids a serious source of experimental error.

Accordingly, a series of five coaching games was played with the computer by each of two instructors, denoted by A and B. The process was halted after five games, since by this time the computer was rarely surprised at anything its instructors did, and simply played remorselessly on to a win. The computer was thus considered, at the end of five games, to have learned most of what its instructors could provide. The results of this experiment are given in Table 1.

One source of error which should be noted at this stage lies in the fact that, although for all games except the first the computer was coached exclusively by the same instructor (playing the hare), the first of B's games the computer saw was played between A and B. Thus, the final result of the teaching process is not quite the work of a single instructor. This should, of course, tend to reduce the differences in the final results, but does not, in fact appear to have had any serious effect.

<i>N</i>	INSTRUCTOR A					INSTRUCTOR B				
	<i>F</i>	1	2	3	4	5	1	2	3	4*
0	40	40	38	36	36	35	33	33	34	34
1	2	0	0	2	2	8	6	4	2	2
2	3	0	2	0	0	4	3	3	4	2
3	2	4	1	2	0	1	3	3	2	3
4	0	0	1	1	3	1	1	1	2	1
5	1	2	1	1	0	—	1	0	0	1
6	1	1	2	0	1	—	1	1	1	1
7	—	0	0	2	0	—	1	1	0	1
8	—	0	1	1	1	—	—	1	0	0
9	—	1	1	0	0	—	—	0	0	0
10	—	0	0	0	2	—	—	1	1	0
11	—	0	0	1	0	—	—	1	1	0
12	—	1	0	1	0	—	—	—	1	0
13	—	—	1	0	0	—	—	—	0	1
14	—	—	0	0	1	—	—	—	0	0
15	—	—	0	0	1	—	—	—	0	0
16	—	—	1	0	0	—	—	—	1	2
17	—	—	—	1	0	—	—	—	—	0
18	—	—	—	0	0	—	—	—	—	0
19	—	—	—	0	0	—	—	—	—	0
20	—	—	—	1	0	—	—	—	—	0
21	—	—	—	—	0	—	—	—	—	0
22	—	—	—	—	1	—	—	—	—	1
23	—	—	—	—	0	—	—	—	—	—
24	—	—	—	—	0	—	—	—	—	—
25	—	—	—	—	1	—	—	—	—	—
<i>S</i> ₁	25	49	74	99	124	23	41	65	79	104
<i>S</i> ₂	93	347	700	1177	1880	49	171	429	725	1328
<i>K</i>	1088	1241	1143	1100	1142	491	773	862	1019	1132

* The last two games are not in a direct sequence.

Table 2. Number of moves appearing with frequency *F* after *N* games

Table 2 presents results which are intended to show that differences in style resulting from different instructors are indeed apparent (although this point will be discussed in more detail below). The hounds have available, on the base line scale as well as on the unreduced scale, 49 possible *moves*. Table 2

gives the number of moves occurring once, twice, and so on, in the course of five coaching games by both A and B. In the case of instructor B, who was initially unacquainted with the game of Hare and Hounds, some initial moves were abandoned as a result of difficulties with the edge effect mentioned above (because of this some moves retained in the memory are very unlikely to recur). The final sequence of games in both cases is such as would be produced by players playing on an infinitely long board. It is worth noting that, of the many configurations common to the two final memories, only two have different moves specified. All of these seem plausible.

GAME PLAYING AND LITERARY COMPOSITION

The frequency distribution of moves shown in Table 2, although of interest, does not make comparisons between the style of play taught to the computer by A and B particularly easy. For direct comparisons to be possible it is necessary to look somewhat more closely at the problem of calculating the characteristic properties of frequency distributions of this form.

The most interesting measure of this type is suggested by the very close analogy between the mechanism of playing a game of Hare and Hounds and that of composing a piece of prose. It is not difficult to see that the selection of the appropriate move for a given configuration corresponds closely to the selection by a writer of the correct word for his vocabulary for a given context. The fact that the hounds are trying to play a winning game can be thought of as equivalent to an attempt to write a piece of prose on a given subject, and so on.

This analogy between game-playing and literary composition suggested that there might be available in the literature on the statistical study of literary vocabulary a suitable measure which could be borrowed. Such a measure is, in fact, available as a result of the work of Yule (1944), who defines a 'characteristic' measure

$$K = 10,000 \left\{ \frac{S_2 - S_1}{S_1^2} \right\},$$

which he uses to study the frequency distributions of nouns in the works of a number of authors. Here

$$S_1 = \sum_i i f_i$$

and

$$S_2 = \sum_i i^2 f_i,$$

where the factor 10^4 is introduced by Yule purely for numerical convenience, and there are f_i moves occurring i times.

It is beyond the scope of this paper to go into the detailed justification of K as a characteristic, independent of sample size, for the distributions of interest here. The reader is referred to Yule's work (1944), with the remark

that all of the arguments used by Yule to justify the use of K in work on literary vocabulary frequency distributions can be carried over unchanged to game-move frequency distributions.

The characteristic K can thus be used as a single measure for the direct comparison of the move frequency distributions produced in the computer by instructors A and B. Table 2 includes the values obtained for K at the end of each of the five games played by the computer with each instructor. For greater ease of comparison, the value of K in both cases are plotted in figure 4.

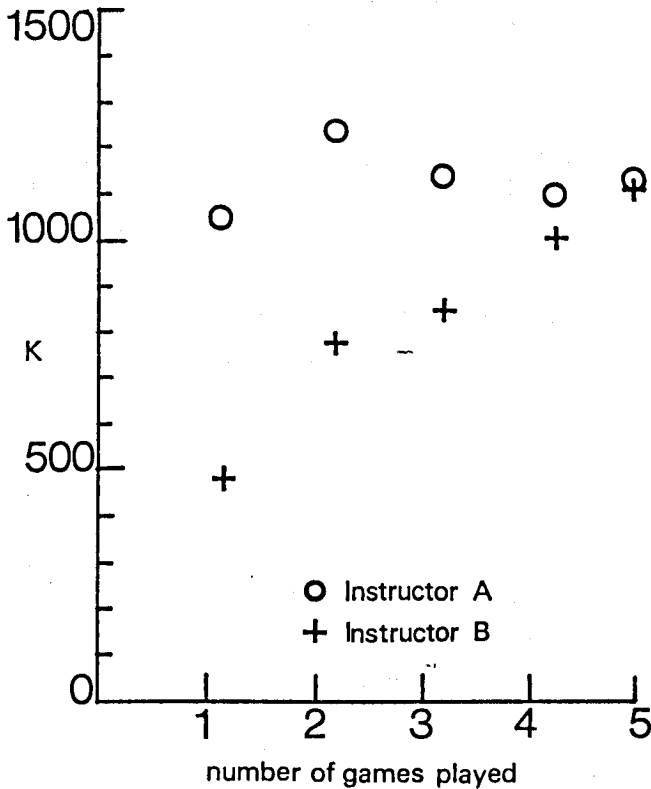


Figure 4. Variation in K with experience

CONCLUSIONS

It is not possible to draw firm conclusions about the answers to the two main questions which this work has raised. However, the results obtained so far have been sufficiently suggestive to encourage further work.

The first question, which was concerned with measuring the effect of the teacher on what is learned by the pupil, is intimately connected with the question of whether or not there is a single 'best' concise algorithm for playing the hounds. The results which are of interest from the point of view of

COGNITIVE PROCESSES: METHODS AND MODELS

both these questions are given in figure 4. That the characteristic K is indeed sensitive to the influence of different instructors on what the computer learns is clear from the very substantial differences in the K values for A and B in the early games learned. Thus the use of K and other measures of this type should make precise studies of the influence of the instructor (and possibly of the teaching method) on the final results possible.

The second point of interest about the results given in figure 4 is the clear suggestion that the K values of the two instructors appear to be tending to the same value. The behaviour observed in the two K values may be due to the fact that initially instructor A had had substantial experience of the game whereas instructor B had not. As the experiment proceeded, instructor B also learned from experience how the game should be played, so that his characteristic tended towards that of the initially more experienced A. This suggests that experienced players should teach the computer to play a game with the same K , and this, in turn, implies that there does appear to be a practical algorithm for playing the 'best' hound game. However, just what this algorithm is would have to be found from other considerations.

Another simple measure obtainable from Table 2 is the proportion of moves not seen by the computer. With B, the percentage of moves not used decreased only from 71 per cent to 69 per cent over the five games, whereas for A the corresponding figures are 82 per cent and 74 per cent. In other words, the inexperienced B introduced most of the practicable moves in the first game, without regard for context; the experienced A introduced new moves much more cautiously.

The analogy pointed out above between literary composition and game-playing has so far proved satisfactory, although further work is needed.

Acknowledgements

The authors would like to thank E. P. V. Storey for his advice and assistance.

REFERENCES AND READING

- Booth, A. D. & Booth, K. H. V. (1953) *Automatic Digital Computers*. London: Butterworth, London.
- Levinson, M. (1967) The computer in literary studies. *Machine Translation* (ed. Booth, A. D.). Amsterdam: North Holland.
- Michie, D. (1966) Game playing and game learning automata. *Advances in Programming and Non-numerical Computation* (ed. Fox, L.). Oxford: Pergamon Press.
- Yule, G. U. (1944) *The Statistical Study of Literary Vocabulary*. Cambridge: Cambridge University Press.

The Holophone - Recent Developments

D. J. Willshaw

and

H. C. Longuet-Higgins

Department of Machine Intelligence and Perception
University of Edinburgh

In this paper we review some of the properties of the holophone (Longuet-Higgins 1968a and b), which is a device analogous to the holograph (Collier 1966) but working in time rather than in space. It was invented to illustrate the principle of non-local information storage as applied to temporal signals, a principle which may very well be used in the human brain. But whether or not this is so, it seems worth while to explore the behaviour of the holophone in some detail, since the device might find application in man-made memory systems.

Before embarking on mathematical details, it may be helpful to indicate some of the properties of the holophone, viewed as a black box with one input channel and one output channel. Three properties are of special interest:

1. It can be used to record any input signal which lies in a certain frequency range and does not exceed a certain length. If part of a recorded signal is then put into the holophone, the continuation of the signal emerges, in real time. In this paper we investigate the amount of noise associated with the playback.
2. Several signals can be recorded on the same holophone. If the signals are random, an input cue from one of the signals will evoke the continuation of that same signal. The accompanying noise increases with the number of recorded signals.
3. The holophone can be used, like an optical filtering system, for detecting the occurrence of a given segment in the course of a long signal. What one does is to record on the holophone the segment of interest followed immediately by a strong pulse. The long signal is then played into the holophone; immediately after an occurrence of the recorded segment a pulse will emerge from the holophone. This property has not been discussed before, and we shall present the relevant theory.

In essence the holophone is a bank of narrow-pass filters, connected in parallel to an input channel, and also connected in parallel, through amplifiers of variable gain, to an output channel. The memory of the system resides in the gains of the various amplifiers. Figure 1 illustrates the layout of the system.

The recording of an input signal is carried out in two stages. The first stage, which corresponds to the formation of a latent image in photography, is to measure the power transmitted by each filter during the passage of the signal. This calls for a set of integrators, which are not shown in figure 1.

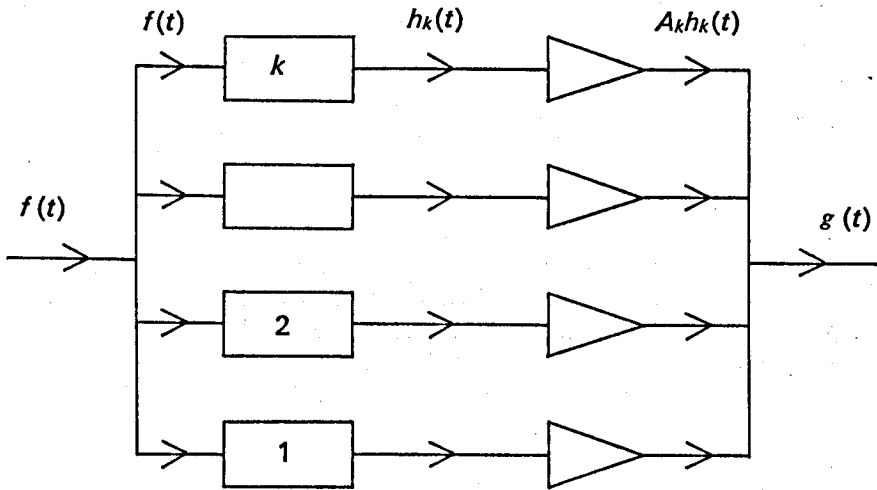


Figure 1

The second stage, corresponding to photographic development, is to turn up the gain of each amplifier by an amount proportional to the value stored in the corresponding integrator. The overall result is to change the response function of the holophone by an amount depending on the temporal auto-correlation of the recorded signal, and this is the secret of the device. But these cursory remarks are unlikely to carry conviction without further argument, so we now give a brief account of the underlying mathematical theory.

Let $f(t)$ be an input signal and let $h_k(t)$ be the output of the k th filter. Each filter must respond linearly

$$h_k(t) = \int_0^{\infty} R_k(\tau) f(t-\tau) d\tau,$$

and its response function must be of the form

$$R_k(\tau) = \frac{\mu}{\pi} e^{-\mu\tau} \cos k\mu\tau.$$

The quantity μ in this expression represents both the bandwidth of every filter and the spacing between the resonant frequencies of neighbouring filters, so that the given frequency range is fully covered. For a particular setting of the amplifiers the output signal $g(t)$ is given by

$$g(t) = \sum_k A_k h_k(t),$$

where A_k is the gain of the k th amplifier.

Suppose now that we wish to record a signal $f(t)$ which is over by the time $t=0$. We arrange for the integrators to measure the quantities

$$W_k(f) = \int_{-\infty}^0 f(t) e^{2\mu t} h_k(t) dt,$$

which may be thought of as the amounts of work done by $f(t)$ upon the various filters, with greater weight attaching to the more recent events. (It can be shown that the W_k are essentially positive quantities, a point of importance for what follows.) Subsequently, at leisure, we increase each gain A_k by a proportional amount, namely

$$\Delta A_k = (2\pi\lambda/\mu) W_k.$$

This process has the effect of altering the response function of the holophone, defined by the equation

$$g(t) = \int_0^{\infty} M(\tau) f(t-\tau) d\tau.$$

Detailed analysis shows that when μ is small the change in M due to the recording of f is

$$\Delta M(\tau) = \lambda \int_{-\infty}^0 f(t') e^{2\mu(t'-\tau)} f(t'-\tau) dt'.$$

This is a time-weighted autocorrelation integral of the recorded signal. If the duration of f is short compared to μ^{-1} , the exponential term in the integrand may be neglected; if it is much longer the earlier part of the signal will be forgotten. The quantity μ^{-1} therefore sets an effective upper limit on the length of signal that can be recorded.

Suppose that initially all the amplifier gains are zero, so that $M(\tau) \equiv 0$, and that a signal f is then recorded. After the recording the response function of the holophone will be given by the above expression for $\Delta M(\tau)$, and a new input signal f' will give rise to an output

$$\begin{aligned} g(t) &= \lambda \int_0^{\infty} \Delta M(\tau) f'(t-\tau) d\tau \\ &= \lambda \int_0^{\infty} \left[\int_{-\infty}^0 f(t') e^{2\mu(t'-\tau)} f(t'-\tau) dt' \right] f'(t-\tau) d\tau. \end{aligned}$$

It might be supposed that this output is merely an indistinct 'echo' of f' , and in general this will be the case. But if f is a sufficiently complicated signal, and if f' happens to be an excerpt from it, a different conclusion must be drawn. To see why, let us begin by writing $g(t)$ in the alternative form

$$g(t) = \lambda \int_{-\infty}^0 f(t') C(t, t') dt',$$

where $C(t, t') = \int_0^{\infty} f(t' - \tau) e^{2\mu(t' - \tau)} f'(t - \tau) d\tau.$

For times t after the end of the cue f' , the integral C becomes a function only of $t - t'$:

$$C(t - t') = \int_{-\infty}^{\infty} f(t' - t + s) e^{2\mu(t' - t + s)} f'(s) ds.$$

It then represents (see figure 2) the degree of resemblance between the cue f' and the section of f that was played into the holophone $t - t'$ units of time ago. If f is sufficiently irregular, $C(t - t')$ will be small unless $t - t'$ equals some fixed time interval θ . We deduce that in these circumstances

$$g'(t) = \lambda \int_{-\infty}^0 f(t') C(t - t') dt' \propto f(t - \theta),$$

so that the output g' will approximate to a continuation of the recorded signal f , carrying on from the moment at which the cue comes to an end. This is our first important result, anticipated at the beginning of the paper.

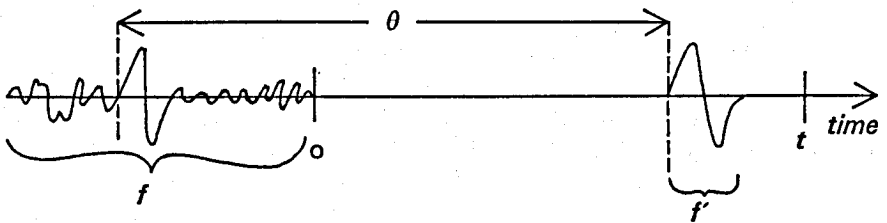


Figure 2

The recall of a whole recorded signal by presentation of an excerpt from it is analogous to the phenomenon of 'ghosts' in holography. Two objects are illuminated by the same coherent light source, and the scattered wavefronts are made to produce an interference pattern on a photographic plate. One of the objects is removed, and the other is illuminated as before and viewed through the interference pattern. A ghost of the absent object is seen beside the object which is actually there. In the temporal case the recorded signal f represents both objects, while the cue f' represents the object which was left

in position during the viewing process. There is only one non-trivial difference between the two cases: the cue f' can only evoke that part of the recorded signal which followed it, not the part which preceded it. It is possible, however, to evoke a time-reversed form of the earlier part of f by playing the cue in backwards! The interested reader may care to establish this curious property for himself.

Before turning to the question of noise in the playback we shall explain how the holophone can be used as a recognition device, since this promises to be one of its most useful applications. Suppose that we are faced with the problem of detecting the occurrence of a short segment f' in the course of a long signal f . (Both f and f' are assumed to be 'noise-like', having no marked periodicities.) What we do is to record on the holophone a signal consisting of f' followed immediately by a strong pulse at $t=0$. The response function of the holophone will then become

$$M(\tau) = \lambda \int_{-\infty}^0 [\delta(t') + f'(t')] e^{2\mu(t'-\tau)} [\delta(t'-\tau) + f'(t'-\tau)] dt'$$

Expansion yields four terms, of which the third vanishes because $f'(\tau)=0$ for positive τ and the fourth may be neglected if the pulse $\delta(t')$ was strong enough compared with the segment $f'(t')$. On this assumption

$$M(\tau) = \lambda [\delta(\tau) + e^{-2\mu\tau} f'(-\tau)]$$

so that $g(t) = \lambda f(t) + \lambda \int_0^{\infty} e^{-2\mu\tau} f'(-\tau) f(t-\tau) d\tau$.

In this expression for the output evoked by $f(t)$, the first term is uninteresting, being merely a playout of f itself. But the other term is a weighted correlation between f and f' , and will make a sudden sharp contribution to the output whenever the recently received section of f is identical with the recorded segment f' . The prepared holophone therefore emits a sharp pulse immediately after any occurrence of the segment which it has been designed to detect. This property of the holophone is precisely analogous to the use of holography in the detection of special features such as printed words in an extended spatial pattern such as a page of a book.

An important question about the holophone is: how much noise will accompany the playback evoked by a cue taken from a recorded message? There is one special case which can be quickly disposed of, corresponding to the case of a collimated reference beam in holography. If the signal to be recorded consists of a strong pulse followed by a weaker signal of some sort, then after the recording has been completed the input of a pulse will evoke the weaker signal virtually free of noise. The reason is simple: in our earlier expression for $C(t-t')$, $f'(s)$ becomes $\delta(s-t_2)$, that is, a pulse at time t_2 , and in

$f(t'-t+s)$ the only significant term is that arising from the recorded pulse, namely $\delta(t'-t+s-t_1)$, where $t_2-t_1=\theta$. Hence

$$C(t-t') = e^{2\mu(t'-t+t_2)} \delta(t'-t+\theta),$$

and our expression for $g(t)$ reduces to

$$g(t) = \lambda e^{2\mu t} f(t-\theta) \propto f(t-\theta).$$

But the more general case must be considered, and to this end we have reformulated the mathematics in discrete terms, assuming all signals to be short enough for the exponential decay factor to be neglected. For further convenience we have also imposed a cyclic boundary condition on the time dimension, and regarded each signal as a set of numbers associated with the vertices of a regular N -sided polygon. The recorded signal is then represented by a set of N numbers, each of which is assigned the value $+1$ or -1 ; the cue is taken to be a limited selection of L of these numbers at adjacent vertices, the other $N-L$ being assigned the value 0 . No loss of generality is then suffered by writing the cue as

$$[f'_1, f'_2, \dots, f'_N] = [f_1, f_2, \dots, f_L, 0, \dots, 0],$$

where the recorded signal is

$$[f_1, f_2, \dots, f_N].$$

With these simplifications the following non-rigorous argument leads to a tentative expression for the signal-to-noise ratio of the 'playback' $[g_{L+1}, \dots, g_N]$ evoked by the cue $[f'_1, \dots, f'_N]$. Defining C_m by the equation

$$C_m = \sum_n f'_n f_{n-m},$$

we may write g_i in the form

$$g_i = \sum_j f_j C_{i-j}.$$

The sum on the right-hand side includes N terms, one of which may be expected to be much larger than the others, namely that for which $i=j$. The value of C_0 is in fact just L , since each of the components of f' matches one of f . But every other term C_{L-j} is the sum of L elements each of which is $+1$ or -1 with equal probability (if the components are random). So taken together these terms have a variance equal to $(N-1)L$, while the square of the amplitude of the signal - the term in C_0 - is just L^2 . The signal-to-noise ratio is therefore $L^2/(N-1)L$, which simplifies to L/N when N is large.

The above argument suggests that the signal-to-noise ratio should be approximately equal to the cue length divided by the length of the recorded signal (for long signals), but we thought it advisable to check the result by computer simulation. POP-2 was chosen as the program language. The following operations were carried out:

1. The N components of an input signal were generated with the aid of a pseudo-random number generator.
2. The first L of these were used to calculate the correlations C_m defined above.
3. The $N-L$ components of the playback signal were then calculated according to the above equation for g_i .
4. Of these components approximately half arise from signal components equal to $+1$, and for this subset the signal-to-noise ratio was calculated from the formula

$$(S/N)_+ = (g_{av})^2 / ((g^2)_{av} - (g_{av})^2).$$

5. The same was done for the components arising from signal components equal to -1 , and the two results were averaged to give an overall signal-to-noise ratio for the entire playback.

A sample set of results is shown below. A single 801-component signal had been recorded, and cues of varying length were used to recall it. The computed and theoretical values of the signal-to-noise ratio are tabulated against the number of components in the cue.

Length of cue (L)	S/N (computed)	S/N (theoretical)
150	0.214	0.187
200	0.252	0.250
250	0.301	0.313
300	0.396	0.375
350	0.412	0.466
400	0.513	0.500
450	0.553	0.572

The computed signal-to-noise ratios bear out the theoretical expression rather well in this case.

We also thought it advisable to test our theoretical estimate of the signal-to-noise ratio when several signals have been recorded on the holophone, and a cue from one of them is used to recall the rest of it. A straightforward extension of our earlier argument indicates that in this case the signal-to-noise ratio should equal the cue length divided by the combined length of all the recorded signals. To test this prediction we recorded ten signals, each of 151 components, and provided cues of varying length from arbitrarily selected signals. The results were as follows:

Length of cue (L)	S/N (computed)	S/N (theoretical)
31	0.0175	0.0206
46	0.0348	0.0303
61	0.0354	0.0404
76	0.0372	0.0503
91	0.0375	0.0604
106	0.0450	0.0701

COGNITIVE PROCESSES: METHODS AND MODELS

Here the agreement is less good, so we checked our last three values by repeating the computation on a fresh set of signals, with the following results:

76	0.0450	0.0503
91	0.0436	0.0604
106	0.0500	0.0701

Presumably the discrepancy between the computed and the theoretical ratios is due to the non-independence of the various C_m , a feature which assumes greater importance for smaller values of N . Be that as it may, the computations show that the primitive theory (which assumes them independent) is at least roughly correct, and may be used as a basis for rough predictions about the behaviour of the holophone (and, for that matter, the holograph).

The above results show that the holophone will indeed function as a content-addressable memory, but that in this rôle it has rather distressing noise characteristics. Used as a recognition device, however, it should perform much more satisfactorily, and might even assume some technical importance. Let us briefly examine the theory of this process, using the same simplifications as were introduced earlier. Using $[f'_1, f'_2, \dots, f'_L]$ to denote the signal which is to be recognized, and $[\dots, f_{-1}, f_0, f_1, \dots]$ to denote the input signal, we obtain the following simple expression for the detection signal:

$$\Delta g_i = \sum_{k=0}^{L-1} f_{i-k} f'_{L-k}.$$

If for some value of i the relationship

$$f_{i-k} = f'_{L-k}$$

holds for $k=0, \dots, L-1$, then the i th component of the detection signal will be a spike of height L , and a spike of this height will signify with certainty the occurrence of f' in f .

A more interesting and realistic problem is that of detecting a slightly noisy version of f' in the longer signal f . The amount of noise in f can be specified by a parameter p which is the probability that the sign of a particular component of f' is wrongly quoted in the input signal f . For this noisy occurrence of f' to be detected, the threshold of the detection device must be lowered below the value L , but not too much or else it will emit false alarms. A sensible criterion for optimizing the threshold is to lower it until the increase in false-alarm probability equals the increase in probability of detection. On this criterion the optimum threshold T is found to have the value

$$T = L(1 + 2/\log_2 p),$$

when p is small and L is large.

Like the holograph, the holophone is a non-local, content-addressable storage and retrieval system. It further resembles the holograph in employing highly parallel logic and in being relatively invulnerable to damage of individual components. It was these characteristics which seemed to recommend it as a possible model of the human temporal memory – though in this context it must be viewed with all possible circumspection. The computations which we carried out to simulate its performance brought home to us the extreme difference in speed between the action of a holophone (which delivers its playback in real time) and the running of a computer program designed to simulate it. The difference is due, of course, to the fact that if the recorded signal is of length N , then N^3 separate acts of multiplication are needed to construct the output evoked by a cue. It is for this reason that the holophone, like the holograph, may be more useful as a hardware device than as a software subroutine – if it eventually finds a place in computing technology.

REFERENCES

- Collier, R.J. (1966) Some Current Views on Holography. *IEEE Spectrum*, July 1966, 67–74.
- Longuet-Higgins, H.C. (1968a) A Holographic Model of Temporal Recall. *Nature*, 217, 104.
- Longuet-Higgins, H.C. (1968b) The Non-Local Storage of Temporal Information. *Proc. Roy. Soc.* (in press).



**PATTERN
RECOGNITION**



Pictorial Relationships - a Syntactic Approach

M. B. Clowes

Division of Computing Research,
C.S.I.R.O., Canberra

1. INTRODUCTION

Grammars or syntax specifications address themselves to the characterisation in symbolic terms of the structure of complex expressions. Two types of expression of empirical interest have been studied: sentences in English and other 'natural' languages, and programs written in some high-level procedural language like ALGOL. Expressions in these languages consist of sets of elements (words and characters) co-ordinated with one another according to the *sensorily* manifest relationship 'alongside', more commonly termed 'followed by'. (In the case of context-sensitive phrase-structure grammars it may also be 'on both sides of', more commonly termed 'bounded by' or 'in the context of'.)

A grammar seeks to relate, by translation or mapping, this manifestation of the expression into another in which the same elements together with others (e.g. 'Noun Phrase', 'Simple arithmetic expression', etc.) are co-ordinated by *abstract* relationships which in the case of English and ALGOL is the single relationship 'parts of'. The notation used to exhibit this relationship is some tree-structure representation in which elements are associated with (i.e., label) the nodes of the tree.

A syntactically motivated parser is a device which *accesses* elements of the sensorily manifest expression, by application of an addressing procedure, e.g. 'Next char' which embodies their sensorily manifest relationship. The parser *develops* a representation in which elements are co-ordinated by the *abstract* relationship ('parts of'), through application of an addressing procedure, e.g. 'Tl' or 'Cdr' (Woodward 1966) which *embodies* this abstract relationship.

In a free paraphrase of Chomsky we might say that a parser translates some set of *surface* relationships on elements, into some set of *underlying* relationships on those (and other) elements. Thus, according to Chomsky

(1965, p. 171), the functional relationship 'Subject of' is to be understood as the relation of 'dominated by' obtaining between a Noun Phrase node and the S(entence) node which immediately dominates it. Needless to say this cannot be translated simply into the surface relationship 'followed by' – hence Transformational Grammar.

The foregoing differs from 'traditional' accounts of generative grammar in according a central rôle to the *relationships* manifest in the surface and the underlying representation of an expression rather than focusing attention upon the categories of element (e.g. *NP, s.a.e., N, var*) and the properties or features which they might be thought to possess. The need to reformulate the account of generative grammar in this way appears essential to the characterisation of expressions of a non-linguistic kind and in particular to pictorial expressions (Clowes 1968).

There have been several attempts to write generative grammars for pictorial expressions notably those of Kirsch (1964), Narasimhan (1966) and Shaw (1967).

In the case of Kirsch's 'triangle grammar' – a set of rules which express addressing (and labelling) operations upon a two-dimensional array of positions – we observe the implicit identification of the surface relationships 'to the left of', 'below', etc., predicated on positions as elements. The 'grammar' specifies no other kind of relationship. Specifically it fails to exhibit as *parts* of the triangle whose structure it purports to describe, the 'edges' of the triangle.

Both Narasimhan and Shaw utilise surface elements consisting of two or more distinct and distinguished positions (called Head, Tail by Shaw and vertices numbered 1, 2, 3 by Narasimhan). Shaw defines these elements by providing co-ordinate values for Head, Tail; thus we may compute the relationship between them, e.g. length or relative position, but this relationship is not *structurally* exhibited. Both authors predicate the surface relationship 'coincidence' on pairs of positions belonging to two different elements. The grammars both assign the underlying relationship 'parts of'. Furthermore on this view of syntax it becomes clear that Minsky's (1961) picture language is one in which a wide *variety* of named relations, e.g. *ABOVE, LEFT, INSIDE* is assumed. Constraints such as 'parallel' in Sketchpad (Sutherland 1963) would appear to have a similar rôle. An immediate distinction between string expressions (English sentences, say) and pictorial expressions now emerges.

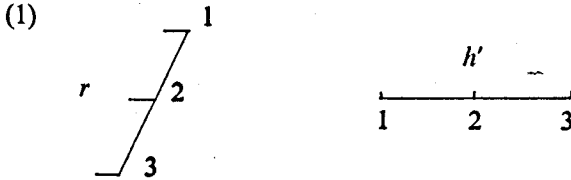
The variety of relationships which we can readily identify and name is much greater in pictorial expressions than in string expressions. We should note that one way to look at Chomsky's 'Aspects of the Theory of Syntax' is an attempt to account for, among other things, the *grammatical relationships* Subject-Verb Verb-Object (pp. 64, 73). This attempt fails in the author's view precisely because of the failure to reformulate the purpose of generative grammar in relational terms. It seems likely that other problems in generative

grammar might benefit from this reformulation, when this distinction between string and pictorial expressions would lose much of its force. It would remain true, however, that linguistic relationships are harder to *identify and name*. This is the essential fact that makes it necessary – if we are to adopt any of the methods of generative grammar – to reformulate syntax as *having to do with relationships rather than with what is related*.

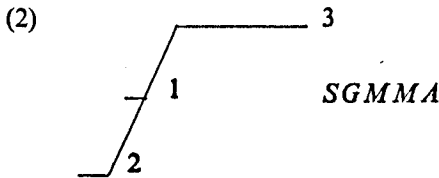
Given that the foregoing analysis is true, it follows that we will have to make provision in the metalanguage (in which we will couch picture grammars) for the overt characterisation of (possibly) large numbers of distinct relationships. Given such a metalanguage, the empirical task becomes that of providing formal definitions of just those relationships as do mediate our grasp of the structure of pictorial expressions.

2. THE METALANGUAGE

As we have noted, both Narasimhan and Shaw are concerned in their notations to exhibit a specific relationship – that of ‘joined’ or ‘connected’. Thus in Narasimhan’s notation the primitives *r* and *h'*



may be considered to be parts of *SGMMA*.



according to the composition rule (3).

(3) $SGGMA(1, 2, 3) \rightarrow r \cdot h'(1 \ 1:2, 3; 3)$

We read this as stating that *r* and *h'* are ‘joined’ at positions or ‘vertices’ designated as 1 of *r* and 1 of *h'*. Furthermore that three positions or ‘vertices’ on *SGMMA* are to be identified with positions 2, 3 of *r* and 3 of *h'*. The descriptions of *r* and *h'* – as each having distinguished positions 1, 2, 3, – implied by (3) is made pictorially explicit in (1). We must imagine of course that in any formal procedural account of this notation, this informal pictorial characterisation would be replaced by a specification of *r* and *h'* in which the positions 1, 2, 3 would be given co-ordinate values. Thus we might replace (1) by

(4a) $r(1(x,y), 2(x+p,y+p), 3(x+2p,y+2p))$

(4b) $h'(1(x,y), 2(x+d,y), 3(x+2d,y)),$

where (x,y) are of course variables assuming different values in each of the three sets of parentheses – that is, h' might have the literal form $h'(1(x,y), 2(x+1,y), 3(x+2,y))$.

The intention of (3) is that it is these co-ordinate values of designated vertices of r and h' which should be 'transferred' to designated vertices of $SGMMA$, rather than the vertices themselves. Similarly that it is an equality of the co-ordinate values associated with 1 of h' of r which underlies this 'join' relationship between these two primitive parts of $SGMMA$. In other words, several pictorial relationships, 'join', 'coincidence', 'same position as' are assumed in our reading of (3). A syntax should provide a formal description of these assumed conventions insofar as they reflect pictorial intuitions. We can rewrite (3) in a form which makes these assumptions explicit and names the varieties of relationship involved.

$$(5) \text{ join } \langle r(1(x,y), 2(x,y), 3(x,y)), h'(1(x,y), 2(x,y), 3(x,y))) \rangle$$

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \end{matrix}$$

$$\qquad \qquad \qquad [Coinc \langle 2,12 \rangle]$$

$$\Rightarrow SGMMA(1(6,7), 2(9,10), 3(19,20))$$

This states that $SGMMA$ is formed from two parts r and h' – enclosed within angle brackets. The relationship which these two parts enjoy in order that they be 'capable' of forming $SGMMA$ is *join*, which entails a further relationship namely of coincidence (*Coinc*) between designated elements of the descriptions of r and h' . The inferior or (suffixed) integers used in (5) merely provide an explicit referencing mechanism to replace the implicit ordering convention of (3). This permits us to state the 'same position as' requirement in respect of the distinguished positions of $SGMMA$ and those of r and h' .

The left-hand side of (5) is descriptive of the structure of $SGMMA$ in exhibiting its parts and specifying the relationship between them. On the right-hand side a further description of $SGMMA$ is provided which does not explicitly state the relationships between the elements (1, 2, 3) which comprise it. The same is true of the descriptions of r , and h' on the left-hand side of (3).

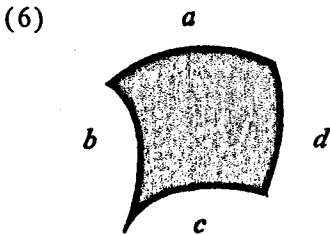
A further difference between (3) and (5) is the use of, and the direction of the \Rightarrow . What we have in mind here is that in discerning that $SGMMA$ is 'made up of two parts' we are recovering relationships, specifically *join*, *Coinc*, on these parts, thereby assigning an underlying structure to $SGMMA$. We identify this process with *parsing*. Accordingly, regarding (5) as a rule of grammar, the arrow points in the generative direction, i.e., towards the surface form of the pictorial expression.

This account of (5) identifies it with a rule of transformational grammar (a T-rule) and we may note a fairly consistent correspondence between the syntactic structure of (5) and the syntactic structure of the *generalised transformation* (Chomsky 1965). Specifically, *join* is the name of the transformation; the pair of descriptions enclosed in angle brackets, are $SD1$, $SD2$; the relationship within the square brackets is the *condition* restriction on the T-rule; the right hand side of (5) is the derived structure.

(5), then, exhibits the metalanguage which we will deploy in discussing pictorial relationships and their rôle in determining our intuitive apprehension of form and shape. The problem now becomes that of determining the relationships and the structures which they co-ordinate. Published accounts of 'picture syntax' have not provided any systematic accounts of the variety of pictorial relationships with which they deal, much less a discovery procedure for those relationships. This omission may of course be intentional in the sense that no attempt is being made to capture our intuitive knowledge of picture structure in these picture syntaxes. In this account, however, we adopt as goal the formal description of our pictorial intuitions and accordingly we shall adopt a more or less systematic methodology for ascertaining what these intuitions are.

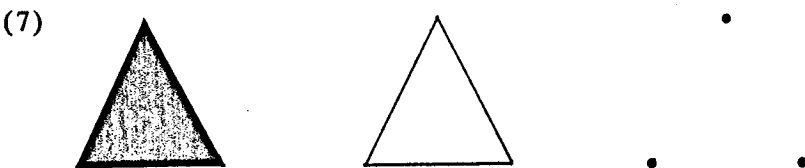
3. THE METHODOLOGY

The methodology to be employed in deciding the specification of a picture grammar is based upon that of Chomsky. Since (as Chomsky remarks) our intuitions are not always immediately apparent it may prove necessary to resort to consideration of particular expressions and pairs of expressions which have the property of rendering our intuitions clear cut. This is the purpose of the study of ambiguous, anomalous and paraphrastic expressions. A good example of the use of ambiguity would be the pictorial expression in (6) which may be seen in two ways: as a 'bellying sail' or a 'sting-ray'.



In the former interpretation we group sides *a* and *c*, *b* and *d*. In the latter 'reading' we group *a* and *d*, *b* and *c*. Any adequate picture grammar must provide the symbolic apparatus by which to exhibit this 'grouping' of edges.

As examples of paraphrase we might take the three pictorial expressions illustrated in (7) which evidently have the same shape.



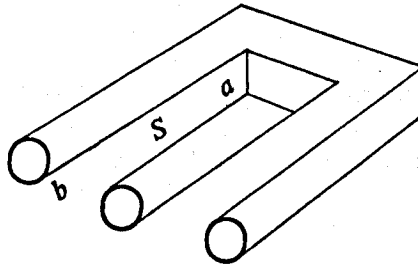
Our intuitions about this similarity of shape apparently involves relations between edges or lines, between the positions we call 'corners', and involves

PATTERN RECOGNITION

among other things the idea that lines can 'function' as edges. We require that the picture syntax give us an adequate formal account of these intuitions.

Finally we might introduce varieties of anomalous picture such as that in (8).

(8)



Attempts to assign an interpretation to (8) break down due to the inconsistency involved in assigning to the region *S* the status of *figure* in the vicinity *a*, but *ground* in the vicinity *b*. Making these assignments is evidently tied up with the recovery of certain relationships between the edges – denoted by lines – in the vicinities of *a* and *b*. The picture grammar must give some account of what sort of relationships between 'edges' mediate or force the assignment of such distinctions as 'figure'/'ground'.

4. THE PICTURE GRAMMAR

4.1. The structural representation of position

Restricting ourselves to mechanical means for displaying or addressing pictorial data (that is, excluding the retina) it is clear that the primitive elements of pictorial expressions must be distinct positions in a two-dimensional array. This conventional view is of course based upon the insight of Cartesian co-ordinate geometry which established the (x,y) notation; that is, the representation of position in a plane by two magnitudes which reflect the operations required to address the given position starting from an origin on a defined axis.

The notation (x,y) implies an axis and an origin but nowhere states it. For our purpose it will be necessary to do so since (x,y) denotes 'position relative to the origin', and we are committed to making all relationships overt. Thus our representation of position would take the form

$$(9) \quad Relpos \langle axis \langle p, p \rangle [bint(t, bint)], p \rangle [bint, bint]$$

1 2
3 4 5
6 7 8

This exhibits the relationship *relative position* (*Relpos*) of a point (or position) suffixed 6 with respect to an axis of co-ordinates defined on the positions suffixed 1,2 as depicted in (10). The value of this relationship is given by a pair of *basic integers* (*bints*) suffixed 7,8 corresponding to the variables

(x,y) in the traditional notation. The characterisation of magnitude by a *bint* rather than some more conventional notation has a specific purpose.

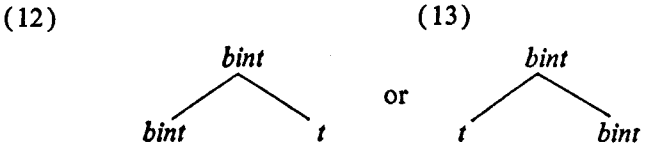
(10) • 6



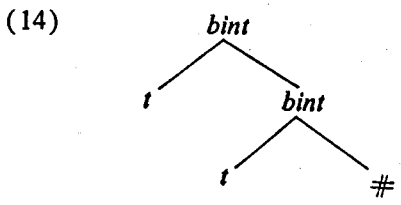
4.1.1. **Bint.** In keeping with the desire to symbolise everything by syntactic structure *bint* is defined by the context-sensitive phrase-structure grammar (11):

- (11)
- (i) $bint \rightarrow \left\{ \begin{array}{l} \{bint\} \\ \# \end{array} \right. , t \left. \right\} / \left\{ \begin{array}{l} t, \\ \{bint\} \\ \# \end{array} \right\} \text{ —}$
 - (ii) $bint \rightarrow \left\{ \begin{array}{l} \{bint\} \\ \# \end{array} \right. , t / \text{ — } \underline{t}$
 - (iii) $bint \rightarrow t, \left\{ \begin{array}{l} \{bint\} \\ \# \end{array} \right\} / t \text{ —}$

The usual notational conventions are implied here, that is $/ \text{ — } t$ specifies that the category *bint* may only be rewritten according to (ii) if it is in the context of — specifically: immediately precedes — a *t*. Braces indicate alternatives. Thus applying (i) we get two main alternatives:

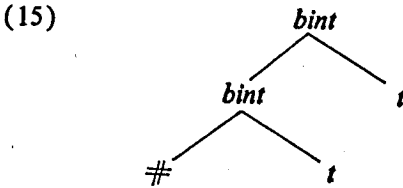


Rules (ii, iii) 'separately' develop (12,13) so that a typical structure resulting from (13) might be (14):

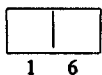


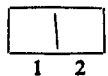
PATTERN RECOGNITION

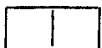
We may think of (14) as specifying the magnitude 1 and of (15) as the magnitude -1



4.1.2. *Axis.* With this interpretation of *bint* we see that the *bint* suffixed 3 in (9) is of unknown but positive (because of its right-branching structure) magnitude. The mapping of *axis* into pictorial relationships is given by the recursive productions (16,17)

(16) $axis\langle p,p \rangle [bint(t,bint)]$
_{1 2 3 4 5}
 $\Rightarrow axis\langle p,2 \rangle [5],$ 

(17) $axis\langle p,p \rangle [bint(t,bint(t,\#))]$
_{1 2}
 \Rightarrow 

Rule (17) asserts that the relationship *axis* of unit magnitude (*bint(t,bint(t,#))*) between positions suffixed 1,2 is identical with the relationship of nearest horizontal neighbours. This pictorial relationship is exhibited in pictorial form thus . In some mechanical device such as a television tube, it would take the form of an incremental voltage applied to the x-deflection plates of the tube.

(16) asserts that an axis defined upon points which are not nearest neighbours – the general case – is manifested as a sequence of co-ordinated overlapping pairs of nearest neighbour positions. The number of such positions is determined by the magnitude of 3 in (9). The ‘;’ in the right-hand side of (16) is to be interpreted as ‘&’.

4.1.3. *Relpos.* While (9) characterises the notion relative position (and, unlike the Cartesian notation (x,y), explicitly identifies the axis of measurement), it does not exhibit the surface (pictorial) manifestation of this underlying relationship; that is, we have so far failed to provide a derived structure for (9). This mapping can now be specified by a recursive production rule having a similar form to (16,17).

$$(18) \quad \text{Relpos} \langle \text{axis} \langle p,p \rangle [bint], p \rangle [bint(t,bint), bint]$$

$$\Rightarrow \text{Relpos} \langle \text{axis} \langle p,p \rangle [3], 4 \rangle [7,8], \begin{array}{|c|c|} \hline & \\ \hline 1 & 9 \end{array}$$

and

$$(19) \quad \text{Relpos} \langle \text{axis} \langle p,p \rangle [bint], p \rangle [bint, bint(t,bint)]$$

$$\Rightarrow \text{Relpos} \langle \text{axis} \langle p,p \rangle [3], 4 \rangle [5,8], \begin{array}{|c|} \hline 9 \\ \hline 1 \end{array}$$

(18) and (19) 'unpack' *Relpos* into a series of nearest neighbour relationships of two types: one corresponding to the *x* dimension being

--	--

, the

other corresponding to the *y* dimension being

. (18) and (19) define

(recursively) relative position for non-zero magnitudes of *x* and *y*. *Relpos* $\langle \text{axis}, p \rangle [bint(t, \#), bint(t, \#)]$ characterises a position coincident with the origin of co-ordinates. Accordingly we may complete the formalisation of *Relpos* by

$$(20) \quad \text{Relpos} \langle \text{axis} \langle p,p \rangle [bint], p \rangle [bint(t, \#), bint(t, \#)]$$

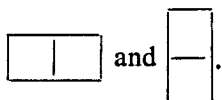
$$\Rightarrow \text{axis} \langle 4, 2 \rangle [3].$$

Evaluating a *Relpos* is akin to changing the pen position on an incremental plotter. We may illustrate this for a very simple case. Consider the position (2,2) i.e., *x* = 2, *y* = 2, which in our terms is

$$(21) \quad \text{Relpos} \langle \text{axis}, p \rangle [bint(t, bint(t, bint(t, \#)))], bint(t, bint(t, bint(t, \#)))].$$

Represent the discrete incremental positions (of the pen) as a square array. Then *axis* of (21) defines some pair of positions labelled 1,2 as shown in (22a) with the associated values of *Relpos*. (For brevity integers are shown as 2, 2 rather than in the explicit structural form of (21)). Applying (18) to this yields a new axis pair denoted 1',2' in (22b), and so on until we reach (22e) where the Cartesian value of *Relpos* is now (0,0). (20) now applies yielding (*f*) which provides a labelled position for 4 correctly positioned with respect to the initial origin 1.

The significance of this lies only in the fact that just two pictorial relationships – sensorily manifest relationships, that is – are utilised, namely



PATTERN RECOGNITION

The other relationships *Relpos* and *axis* are defined on these and are, according to our earlier discussion, 'abstract' or underlying relationships. (22) demonstrates, moreover, that any formulation of pictorial relationships which is reducible to *Relpos* can be effectively computed, that is, defines addressable positions on some plane surface, given only that two directions and a unit separation are defined on that surface.

(22)

- (a) 1 2 *Relpos* <*axis*<1,2>[*bint*],*p*>[2,2]
- (b) 1 1' 2 2 *Relpos* <*axis*<1',2'>[3],4>[1,2]
- (c) 1'' 2''
1 1' 2 2' *Relpos* <*axis*<1'',2''>[3]4>[1,1]
- (d) 1''' 1'''' 2''' 2''''
1 1' 2 2' *Relpos* <*axis*<1''',2''''>[3],4>[0,1]
- (e) 1'''' 1'''' 2'''' 2''''
1 1' 2 2' *Relpos* <*axis*<1'''',2''''>[3],4>[0,0]
- (f) 1'' 4 2'' 2''''
1 1' 2 2' *axis*<4,2''''>[*bint*]

(18), (19) and (20) define position for the first quadrant only. A further version of (18) and of (19) is required dealing with negative magnitudes.

(18a) *Relpos* <*axis*<*p*,*p*[*bint*],*p*>[*bint*(*bint*,*t*),*bint*]

1 2 3 4 5 6 7 8

⇒ *Relpos* <*axis*<*p*,*p*>[3],4>[6,8],

9	1

and

(19a) *Relpos* <*axis*<*p*,*p*>[*bint*], *p*>[*bint*, *bint*(*bint*,*t*)]

1 2 3 4 5 6 7 8

⇒ *Relpos* <*axis*<*p*,*p*>[3],4>[5,7],

1
9

A corresponding version of (20) differs only in applying to the negative version of zero.

Notice that the same two pictorial relationships are used; we have merely changed the 'ordering' of the position pairs they relate. Thus, the definitions of *Relpos* express our intuition that if 'up' or 'left' is thought of as 'positive' then 'down' or 'right' is 'negative', where both 'positive' and 'negative' are given explicit definitions in terms of structural manipulations of bints.

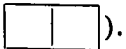
The representation of integer magnitude in a structural form¹ is thus

¹ Originally introduced by my colleague D. J. Langridge to provide a syntactic account of the arithmetic operations plus, multiply, etc.

associated with distance and measure. It is of course possible to formulate *Relpos* using not *bint* but more conventional integer symbolism, e.g. $n, n-1$, etc. We have adopted *bint* here in order to avoid the implication that our intuitions of position are based upon arithmetic concepts. Rather, we would like to suggest that the reverse is the case.

4.2 The accuracy of position judgements

Consider the following simple experiment. We provide a blank, square sheet of paper upon which we have marked a point. The subject (*S*) is invited to estimate the position of the point relative to the edges of the paper as axes, and to verbalise this estimate as an integer pair. The magnitude of these integers relates to an 'assignment', on the part of *S*, of an *interval scale* to the vertical and horizontal edges of the paper. It has been found (Klemmer and Frick 1953) that the accuracy of these integer estimates is about 20 per cent; that is, *S* can discriminate roughly 25 positions in this square. This limitation seems a fundamental one (Miller 1956, Clowes 1967); what does it imply for the structural description of position formulated here? The experimental observations are consistent with the view that the *bints* (suffixed 7, 8) in (9) both having a limiting value of 5. The magnitude of these integers is of course dependent upon the magnitude of the *nearest neighbour interval* (e.g.



We may, therefore, say that in judging relative position – recovering, that is, a *Relpos* such as (9) from its pictorial manifestation – *S* 'chooses' a nearest neighbour interval to define an axis. This interval is sufficiently large that each of the bints in 9 will not exceed 5 in magnitude.

4.3 Pairs of positions

Pairs of points may form an entity which is related to an axis. We may think of a pair as defining a line, i.e., as the positions of the ends of a straight line. Our grasp of the line as an entity involves a relation between the end points of the line which betray an inherent axis. Thus we might say the line is 'vertical', 'sloping' or 'long'. All of these epithets which apply to the relationship between the end points imply an axis and an interval on which that axis is defined. We shall designate this relationship *coord* expressed as

$$(23) \quad \underset{1}{\text{coord}} \langle p, p \rangle [\underset{2}{\text{Relpos}} \langle \text{axis} \langle 1, p \rangle [\text{bint}, 2 \rangle [\text{bint}, \text{bint}]]] \\ \Rightarrow \underset{4}{\text{Relpos}} \langle \text{axis} \langle p, p \rangle [\text{bint}, 1 \rangle [\text{bint}, \text{bint}], \\ \text{Relpos} \langle 4, 2 \rangle [\text{bint}, \text{bint}]$$

(23) shows how choice of an *axis* (4) commits us to a particular image of the relation between 1 and 2. If we rotated 4 we would grasp 1, 2 differently. (23) *fails*, however, to bring out the fact that the *axis* on the left-hand side is 'parallel' to that on the right hand side (see Postscript).

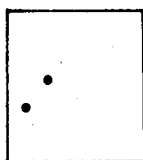
PATTERN RECOGNITION

4.3.1. *Near*. In describing a line as 'long' or 'short' we are implicitly relating its endpoints: the judgement appears to apply to positions, minimally to pairs of positions. If position is characterised by *Relpos*, and we are to seek a formal specification of position judgements in structural terms then it is natural to seek to characterise *near* as, say, an identity condition upon the independent structural characterisation of two positions. Thus we might argue that two *Relpos*'s (of the form (9)) having the same *axis* and the same *bint* (7,8) value would describe two positions which are near to each other. We have seen that the use of different nearest neighbour intervals in the definition of *axis* implies a labile metric for judgement of position. It follows, then, that in judging that two positions are near to one another we are assigning an axis – common to both *Relpos*'s – which makes them so. We may express this as a relation.

$$(24) \quad \text{near} \langle \underset{1}{p}, \underset{2}{p} \rangle [\text{Eq} \langle \underset{3}{bint}, \underset{4}{bint} \rangle, \text{Eq} \langle \underset{5}{bint}, \underset{6}{bint} \rangle] \Rightarrow \text{Relpos} \langle \underset{7}{axis}, \underset{8}{1} \rangle [\underset{9}{3,5}, \underset{10}{Relpos} \langle \underset{11}{7,2} \rangle [\underset{12}{4,6}]]$$

The crucial question, in making this judgement therefore rests upon the choice of axis.

(25)



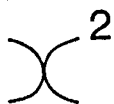
(a)



(b)

Thus, in (25a), relative to the rectangular frame (an *axis*) the pair of points appear close, but in (25b) they do not. Of course the axis may not be an external one.

(26)



(a)



(b)

For example, where we are judging the proximity not of positions but of complex pictorial forms as in (26), the *axis* may be provided by the forms themselves. Thus large characters (26a) appear nearer than small ones (26b).

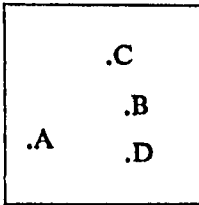
In introducing the metalanguage (§2) we made use of relationship *Coinc* (5) between a pair of positions. What is the distinction between 'coincident'

and near? We shall take it that coincident is an identity relationship upon a pair of positions, i.e., that they are the same positions.

4.4 Side

A weaker form of positional judgement than that involved in *Relpos* is the judgement as to which side of a pair of points a third point lies. For example in (27) the points C and D are on opposite of the line AB, while A and D are on the same side of the line CB.

(27)



In making this judgement it is intuitively apparent that a reorganisation of structure is involved. Thus, prior to the remark about 'side', we see the points as occupying unrelated positions in the rectangle. *It* acts as the frame of reference or origin of coordinates. In making the 'side' judgements we use first AB and then CB as the frame of reference. Thus (27) is an ambiguous picture and as such reveals the syntactic structure of the 'side' relationship.

The first organisation would be characterised by four distinct expressions of the form of (9) each taking A,B,C,D as the 'point' of the *Relpos*, i.e., as the item suffixed 6 in (9), all identifying the same positions as *axis*. Informally we may associate the latter with the base of the rectangular frame in (27). The 'side' judgement however takes a pair of these positions, say A,B as axis.

$$(28) \quad \underset{1}{side} \langle \underset{2}{p}, \underset{3}{p}, \underset{4}{p} \rangle [\underset{5}{bint}] \Rightarrow \underset{6}{Relpos} \langle \underset{1}{axis} \langle 1,2 \rangle [\underset{3}{bint}], \underset{4}{3} \rangle [\underset{5}{bint}, \underset{6}{4}]$$

Thus in (28) the positions suffixed 2, 3, 5 might be A,B,C. The derived structure of (28) is a *Relpos* relation between C and A,B as axis. (28) thus defines *side* as that relationship of relative position in which the 'x-co-ordinate' is undetermined.

The judgement 'D and C are on opposite sides of AB' is the result of a comparison of two relational structures of the form typified by the left-hand side of (28) both having the same *axis*. It follows, of course, that the *bint* (suffixed 6 in (28)) in one structure will be of opposite sign to that in the other – it is this which we will regard as underlying the judgement.

The judgement 'A and D are on the same side of CB' is another comparison of a pair of relational structures each involving CB as *axis*, with A,D as the positions.

We may treat 'opposite sides' and 'same side' as *relations* between two points and a pair of points. Thus *sside* (same side) might take the form

$$(29) \quad \underset{1}{sside} \langle \underset{2}{p}, \underset{3}{p}, \underset{4}{p} \rangle [\underset{5}{Ssign} \langle \underset{6}{bint}, \underset{6}{bint} \rangle] \Rightarrow \underset{5}{side} \langle 1,2,3 \rangle [\underset{6}{5}], \underset{6}{side} \langle 1,2,4 \rangle [\underset{6}{6}]$$

Ssign (same sign) is the relationship 'same structure type' on the pair of *bints* to which it applies (remembering that the distinction between positive and negative is exhibited in the structure of *bint*).

What (29) does of course is to specify formally what was earlier stated informally as the 'comparison of two relational structures of the form typified by the left-hand side of (28)'. That is *sside* is assigned to this set of four points just in case the separate *side* relationships on the right-hand side of (29) satisfy the *Ssign* relationship specified on the left-hand side.

4.5. Discussion

The foregoing has established that, with the notational system introduced in §2, we may characterise a variety of relationships between positions. The relationships to be characterised are made evident by considering various pictorial examples in accordance with the general methodological approach outlined in §3, that is, we are characterising our intuitions about picture structure, *not* erecting some arbitrary picture calculus.

Our perception of these varieties of pictorial organisation can be identified with the assignment of these functional descriptions, e.g. *Relpos*, *sside*, etc., to the primitive sensorily-manifest data. The process of assignment is essentially a *parsing* process. It will be evident that given say the illustration (27) there are very many (probably an infinite number of) relationships which could be assigned to this collection of points and the frame. We suggest that this is entirely consistent with observation: there *are* many ways of looking at (27). Significantly, however, we cannot hold these multiple views simultaneously – we switch between them. Formally, that is, we can only assign a single relational structure at a time, although this structure may relate a number of items, e.g. *sside*, in quite a complex manner.

There is one major respect in which the whole of the presentation to this point has been inadequate. In exhibiting various pictorial expressions manifesting varieties of position relationship, we have *assumed* that a position is denotable by a 'point' and that a 'straight line' has two salient positions associated with its two 'end points'. These positions are abstractions which underlie the *forms*, 'line', 'point', 'frame', etc. The whole apparatus is empty of empirical interest (i.e., has no application to picture interpretation) unless we can also characterise how these abstractions are possible, that is, characterise *form* with the same apparatus used to characterise *position*.

5. THE CHARACTERISATION OF FORM

The recovery of Form may be stated to be the discovery of the *significant* positional relationships exhibited in the picture. Thus, in our preceding account we have denoted position by a point and pairs of positions (as in *coord* for example) by a straight line. This denotation presumes that from these two types of form – point and line – the reader may easily recover the positional structure which underlies them. In picture interpretation, the raw

data is a very large number of possible relationships between positions (sampled by a scanner) having distinguishable colours, e.g. between all pairs of raster positions. A picture containing a line clearly has more than one significant positional relationship exhibited in it although there may be only one (the relationship between end points of the line) which we wish to utilise. The positional relationships between say black and white points which subsume the *edge* of the line (equivalently the edge of a point), are obviously necessary to the exhibition of the line itself and the recovery of these more primitive relationships must precede (in parsing the picture) the recovery of the relationship between the end points of the line.

The essential difference between the interpretation of computer graphics and their hard-copy equivalents lies precisely in the fact that these 'end point relationships' evident in both, are the raw data when input at the graphics console, but *extensively parsed* data when recovered from the hard-copy *via* a scanner.

The object of parsing is in some sense (one which we will progressively make sharper) the recovery of objects with which we can associate some 'position information', i.e., a line having as 'position information' its end points. We may regard an object to be defined as 'position information' upon which a variety of position relationships are specified. It will be convenient to utilise much the same notation as already deployed except that the name of the relationship, e.g. *side*, will be replaced by the object name, e.g. *LINE*.

5.1. Straight edges

We shall adopt the notation $p(\textit{colour } i)$ to designate a position having colour i . Clearly, objects or forms are ultimately dependent for their exhibition upon *distinctions* of colour between 'sets' of positions enjoying certain varieties of spatial relationship. In fact, we could regard $p(\textit{colour } i)$ as an abbreviation for an object definition involving two or more sets of positions having colours i, j respectively, enjoying the relationship $i \neq j$. Underlying the form 'straight edge' (abbreviated to *SE DGE*) we discern a pair of positions – the 'ends' of the edge – and a colour relationship between the opposite sides of the edge.

(30)

$$SE DGE \langle p, p, \textit{colour}, \textit{colour} \rangle [side \langle 1, 2, p(3) \rangle [0], side \langle 1, 2, p(4) \rangle [-1], Diff \langle 3, 4 \rangle]$$

1 2 3 4

(30) provides a partial formalisation of this form in terms of the *side* relationship. The values (0, -1) of the two *sides* employed restricts the scope of the colour contrast to be local to the edge in the y direction. However, it leaves unspecified the range of the contrast in the direction of the edge, i.e., in the x direction. This is a direct consequence of the formulation of *side*; note, however, that since isolated straight edges are pictorially anomalous, that is they can only occur in the context of some extended boundary, no parsing problems should arise from this imprecision. The relationship $Eq \langle \textit{colour},$

colour > may be said to have the value *true* when the two colours are discernibly different, otherwise *false*. The relationships predicated as underlying *SE DGE* are essentially those recovered by various types of ‘edge follower’ (Greanias 1963, Ledley 1964) in so far as these programs only examine some restricted neighbourhood of positions in the picture in order to assign an ‘edge’.

5.2. Convex boundaries

The simplest ‘context’ in which a straight edge can occur is that whose underlying ‘positional information’ is what we apprehend as a convex polygon. Let us call this form a convex boundary (*CBND*).

(31)

$$CBND \langle SE DGE \langle p, p, colour, colour \rangle [\underset{1_i}{}, \underset{2_i \ 3_i}{}, \underset{4_i}{}, \underset{5_i}{}, \dots \rangle \rangle [\underset{1_{i+1}}{SE DGE \langle p, p, colour, colour \rangle [\dots]}, \dots]$$

$$[Diff \langle 4_i, 4_{i+1} \rangle, Between \langle 2_i, 3_i, 2_j, 3_j, p(4_i) \rangle,$$

$$Coinc \langle 3_i, 3_{i+1} \rangle, side \langle 2_i, 3_i, 3_{i+1} \rangle [bint(bint, t)]$$

$$sside \langle 2_i, 3_i, 2_j, 3_j \rangle [] - \text{for all } i \neq j]$$

The formulation (31) is in terms of five types of relationship (*Diff*, *Between*, *Coinc*, *side* and *sside*) predicated upon an ordered set of *SE DGE*s. The ordering is determined by the values of *Coinc* and *side* which characterise continuity (or connectivity) and a clockwise (because *side* is negative) progression around the boundary. Convexity is specified by *sside*. The ‘fact’ that the spatially distributed colour relationships which support a form are not local to the edge (the assumption underlying *SE DGE*) is exhibited by the relationship *Between*.

(32)

$$Between \langle \underset{1}{p}, \underset{2}{p}, \underset{3}{p}, \underset{4}{p}, \underset{5}{p} \rangle [Ssign \langle \underset{6}{bint(t, bint)}, \underset{7}{bint(t, bint)} \rangle, sside \langle 1, 2, 4 \rangle]$$

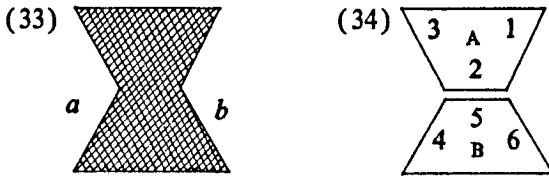
$$\Rightarrow side \langle 1, 2, 5 \rangle [6], side \langle 3, 4, 5 \rangle [7]$$

Of course such a form – one in which the ‘interior’ colour is uniform – is idealised. Naturally illuminated objects present interiors whose illuminance varies in a highly complex fashion. Thus the two-dimensional distribution of retinal illumination produced by a uniformly illuminated sphere, is dark at the edges (Lambert’s law), brightening uniformly towards the centre.

5.3. Compound forms

All forms do not of course have an underlying *CBND*. It is an essential part of this approach to picture syntax that we regard Forms containing concavities of boundary, as concatenations of Forms which *are* convex. Intuitively, the varieties of concatenation would be described as ‘join’, ‘overlap’,

and 'touch'. Such descriptors evidently apply to complete convex Forms in such a manner as to merge the two *CBNDs* deleting one or more *SEDGEs* and introducing concavities characterised by positive *side* relationships. Thus, if we take (34) to be the perceptual organisation underlying (33),



then the *BND* derived from joining the *CBNDs* of A and B would have 1 followed by 6, 4 followed by 3 and positive *sides* relating 1 and 6, 4 and 3. The *SEDGEs* 2 and 5 would have been 'deleted'.

The concatenation requires several relationships including an 'agreement' between the 'colour pairs' ($4_i, 5_i$ of (30)) for each *CBND*. In the event that A and B are of different colour *SEDGE* 2 will be 'merged' with *SEDGE* 5 not deleted, in the derived structure there will be a single *SEDGE* replacing 2 and 5, this *SEDGE* having an appropriately modified colour contrast. This form of join is appropriate to pictures such as maps: it will not be explored here.

Whether 2 and 5 are deleted or merged there will be two positions (a, b in 33) in the resultant *BND* where essentially new side relationships will be introduced. At least one of these positions a concavity (i.e. a negative *side*) must be introduced if we are to be able to recover the underlying pair A, B. [There is a weak sense in which any n -gon ($n > 3$) may be decomposed into $n - 2$ triangles and so on, even if the n -gon is convex. We do not consider this case here.]

The key relationship between A and B to which the term 'join' applies is of course the 'coincidence' of the *SEDGEs* underlying 2 and 5; that is, (33) is decomposable into the two parts A, B of (34) at which 'point' the relationship between A, B 'emerges'.

5.4. Discussion

In the foregoing sketch, we see how the *relations* defined in §4, mediate our apprehension of Form. From the standpoint of analysing pictures, we may say that the relationships underlying *CBND* (31), are predicates whose value must be *true* over the set of *SEDGEs* which constitute the arguments of these predicates. We may think of these arguments as the parts of *CBND*.

The formulation (31) is close to that developed by Evans (1968) and Guzman (1968). The crucial issues, however, are identified as being concerned with questions having to do with relationships not objects or forms, that is, 'How many relationships are there?'; 'How are they related and defined?', and so on. The answers given here to these questions may require revision in

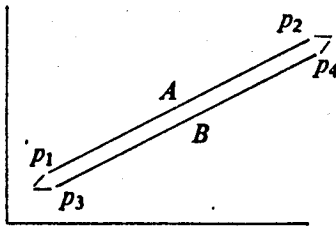
the light of further study; at present, however, the list of relationships is evidently small, perhaps 10-20, and they form a hierarchy in that more powerful relations are defined in terms of simpler ones (*see* Postscript).

It should now be evident, therefore, that a characterisation of Form is possible along these (relational) lines, and accordingly we can claim to have met the objection formulated in §4.5 concerning the status of observations about position which rely upon a grasp of Form for their statement. Specifically we may ask 'What is a straight line?'

6. SHAPE

A straight line is a Form having a Boundary whose underlying positional relationships are specified by a single *CBND*; that is, it is not a compound form. The *SEDE* set comprising this *CBND* is further characterised by having a pair of *SEDEs* between which the relationship *parallel* obtains and whose *ps* enjoy a *near* relationship. That is if *A, B* (in (35)) are the *coords* in question, then p_1 is *near* p_3 and p_2 *near* p_4 .

(35)



Thus we see that 'straight line' (and any other line for that matter) involves a judgement of positional relationships upon specified elements of a Form. We have seen (§ 4.2) that the accuracy of positional judgements is based upon the assignment of an *axis* which acts as a scale determiner. If we take this axis to be either *A* or *B* then we see that we are saying that a Form will appear line-like if, in addition to the requirements set out above, p_1 is *near* p_3 and p_2 *near* p_4 , taking *A* or *B* as axis. This will be the case if the form is, as it were, much 'narrower' than it is 'long'.

The crucial concept underlying this formulation of 'line' is that it 'involves a judgement of positional relationships upon specified elements of a Form'. We take this to be a general definition of Shape.

In this way we see that Shape involves Form but involves in 'addition' the 'recovery' of further position relationships whose accuracy is, of course, subject to the limitations discussed in §4.2. We may think of these additional relationships as the *metrical aspect of Shape*. In judging the similarity of Shape of two or more forms, e.g. as in (7), we first recover the *CBNDs* underlying the 'raw data' – and note we attempt to recover the same *CBNDs* – then we evaluate metrical relationships between the various *ps*.

Where there are alternative sets of metrical judgements of an essentially

different kind, e.g. *Relpos* as against *parallel*, we may see that a single Form has two or more alternative Shapes. This is the case in (6).

We noted in § 5 that 'the raw data is a very large number of possible relationships between positions (sampled by a scanner) having distinguishable colours'. The characterisation now given of Form and Shape suggests that the central problem in the assignment of structure to a picture (i.e., in Picture Interpretation) is the 'decision' as to what varieties of relationship are to be recovered, since as we have seen there is no 'S' (in the normal grammatical sense) from which all well-formed pictorial sentences are derived. This is a more or less direct consequence of espousing a wholly relational and transformational syntax. For the simple (whatever that means) pictures we have discussed thus far it appears counter-intuitive to suggest that there are many possible Forms and Shapes which are recoverable, i.e., visible in it. When faced, however, with a wholly novel picture, for example that produced in some esoteric experiment in physics, we may find that it takes some considerable time to adjust our view so as to recover the significant elements of Form and reject the insignificant. That we have a strong predisposition to see certain Forms and Shapes and not others is of course familiar to the psychologist. Boring's 'Young Girl/Mother in Law' (1930) is a classic example.

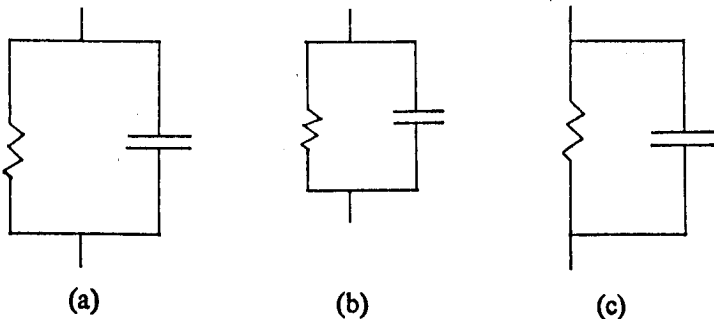
The conclusion we would draw from this is that the structure we assign to a picture is determined not solely by the 'raw data' of that picture but also by *a priori* decisions as to the varieties of relationship we expect to find there. The question therefore becomes 'can we formalise these *a priori* decisions?'

7. THE SEMANTICS OF PICTURES

We may summarise the foregoing argument as 'People see what they expect to see'. The essential rider is that *what* they want to see is *things* not pictorial relationships, that is, the *a priori* decisions reflect assumptions about the things and events which we expect to see exhibited in the picture. We shall argue that it is necessary and indeed possible to give a structural characterisation of things and events which is a mapping of the relational structure of the picture. This characterisation we call the *semantics* of the picture.

The case for something beyond the recovery of pictorial relationships is readily made by appeal to the methodology (§ 3).

(36)



Thus, while (36a) and (36b) are pictorial paraphrases, there is a variety of paraphrase, evident between both of them and (36c) which cannot be established on pictorial grounds. It is the underlying electrical relationships which are the same in all three pictures. We might expect that, corresponding to *Relpos*, we have *potential difference* and also *phase difference*. These relationships are, of course, purely abstract, that is, they cannot be recovered except *via* some sensory manifestation, usually pictorial. Just what the syntactic structure of a circuit might look like, utilising electrical relationships in the metalanguage, is not yet clear. The example parallels closely those given by Chomsky (1965, pp. 160-3) in discussing 'additional problems in semantic theory'. He suggests that anomalies such as '* the cut has a finger' (vs. 'the finger has a cut') are to be accounted for not in terms of language use but in terms of 'language independent constraints . . . in traditional terms, the system of possible concepts'. He remarks 'it is surely our ignorance of the relevant psychological and physiological facts that makes possible the widely held belief that there is little or no *a priori* structure to the system of "attainable concepts"'. The semantic structure we have been arguing for here is in our view identifiable with Chomsky's 'structured system of attainable concepts'.

We may note that developments in question-answering programs are placing increasing weight on the structure of the data base. From the standpoint espoused here we would regard the data base as exhibiting the relationships between *events*, e.g. games in Baseball (see Green *et al.* 1963), and entities, e.g. teams, places, scores. These relationships characterise our knowledge of league games in the same way that *Relpos* characterises our pictorial knowledge.

8. CONCLUSION

In this paper an approach to picture interpretation has been outlined. This views the process as one of assigning to the 'raw data' sampled by a picture scanner (equivalently the retina) a structure which makes explicit the varieties of pictorial relationship *visible* in that picture. These relationships are specified in a metalanguage (regarding the 'raw data' as an object language) having a strong similarity to that deployed in transformational grammar. In choosing to characterise *visible* structure we adopt a methodology which is intended to expose just those relationships which mediate *our* grasp of Form and Shape.

Among the many problems thrown up by this work we would single out the characterisation of the semantics of pictures in relational syntactic terms as crucial. The parallels drawn with current work in question-answering suggests that it may be profitable to consider not only event structures having a pictorial manifestation but events readily characterised in English too. Indeed we might consider situations like particle physics where we have bubble chamber photographs, English descriptions of particle interactions,

and an algebraic representation, e.g. $\mu^- + p - \varepsilon^0 + K^0$, as ideally suited to determine the semantic relationships.

A machine (or program) capable of mediating translations between these various languages would utilise the underlying semantic structure as the 'pivot' of the translational process. We could describe such a machine as 'informed' – 'informed', that is, about the varieties of relationship applicable in these various representations of an event. It would not, however, be intelligent. Such an appellation should be reserved for a machine (like us) capable of formulating and testing hypotheses about *new* relationships and ultimately about new systems of attainable concepts manifesting these relationships.

Acknowledgements

The approach to picture interpretation outlined here has emerged from the Verbigraphics Project. It is a pleasure to acknowledge my indebtedness to my colleagues D. J. Langridge and R. B. Stanton. I am grateful to Dr G. N. Lance for encouraging us and supporting us in this work.

REFERENCES

- Boring, E. G. (1930) A new ambiguous figure. *Am. J. of Psych.*, 42, 444–5.
- Chomsky, N. (1965) *Aspects of the theory of syntax*. Cambridge, Mass: MIT Press.
- Clowes, M. B. (1967) Perception, picture-processing and computers. *Machine Intelligence 1*, pp. 181–98 (eds Collins, N. L. & Michie, D.). Edinburgh: Oliver & Boyd.
- Clowes, M. B. (1968) Transformational grammars and the organisation of pictures. Seminar Paper No. 11. CSIRO Div Computing Research, Canberra. Presented to the Nato Summer School on Automatic Interpretation and Classification of Images, Pisa, Italy.
- Evans, T. G. (1968) Paper presented to the Nato Summer School on Automatic Interpretation and Classification of Images, Pisa, Italy.
- Greanias, E. C. & Meagher, P. F. (1963) The recognition of handwritten numerals by contour analysis. *IBM J. Res. & Dev.* 17, 1.
- Green, B. F., Wolf, A. R., Chomsky, C. & Laughery, K. (1963) Baseball: an automatic question answer. *Computers and thought*, pp. 207–16 (eds Feigenbaum, E. A. & Feldman, J.). New York: McGraw-Hill.
- Guzman, A. (1968) Paper presented to the Nato Summer School on Automatic Interpretation and Classification of Images, Pisa, Italy.
- Kirsch, R. A. (1964) Computer interpretation of English text and picture patterns. *I.E.E. Trans. on Electronic Computers*, 13, 363–76.
- Klemmer, E. T. & Frick, F. C. (1953) Assimilation of information from dot and matrix patterns. *J. Exp. Psychol.*, 45, 15–19.
- Ledley, R. S. (1964) High speed automatic analysis of biomedical pictures. *Science*, 146, 216–23.
- Miller, G. A. (1956) The magical number seven; plus or minus two: some limits on our capacity for processing information. *Psych. Rev.*, 63, 81–97.
- Minsky, M. (1961) Steps toward artificial intelligence. *Computers and thought*, pp. 406–50 (eds Feigenbaum, E. A. & Feldman, J.). New York: McGraw-Hill.
- Narasimhan, R. (1966) Syntax-directed interpretation of classes of pictures. *Commun. Ass. comput. mach.*, 9, 166–73.

PATTERN RECOGNITION

- Shaw, A. C. (1967) A proposed language for the formal description of pictures. *GSG Memo 28*, Stanford Linear Accelerator Centre.
- Sutherland, I. E. (1963) Sketchpad – a man-machine graphical communication system. MIT Lincoln Lab. Tech. Rept. No. 296.
- Woodward, P. M. (1966) LIST programming. *Advances in programming and non-numerical computation*, pp. 29–48 (ed. Fox, L.). Oxford: Pergamon Press.

POSTSCRIPT

During the process of putting the manuscript of this paper into typescript several points have emerged which help to place this work in better perspective.

Coord: There are two aspects of this entity which make it clear that it should be regarded as an object (cf. *SEDGE*) rather than a relationship: (i) The *Relpos* included in the L.H.S. of (23) references the elements suffixed 1,2. This is completely atypical of relation definitions but quite characteristic of object definitions. (ii) We have not utilised *coord* as a relationship in the formulation of any other relationship or object nor does it seem likely that we would want to. We therefore conclude that (23) is incorrect and that (23a) is more likely.

(23a) $COORD \langle p, p \rangle [Relpos \langle axis \langle 1, p \rangle [bint], 2 \rangle [bint, bint]]$
_{1 2}

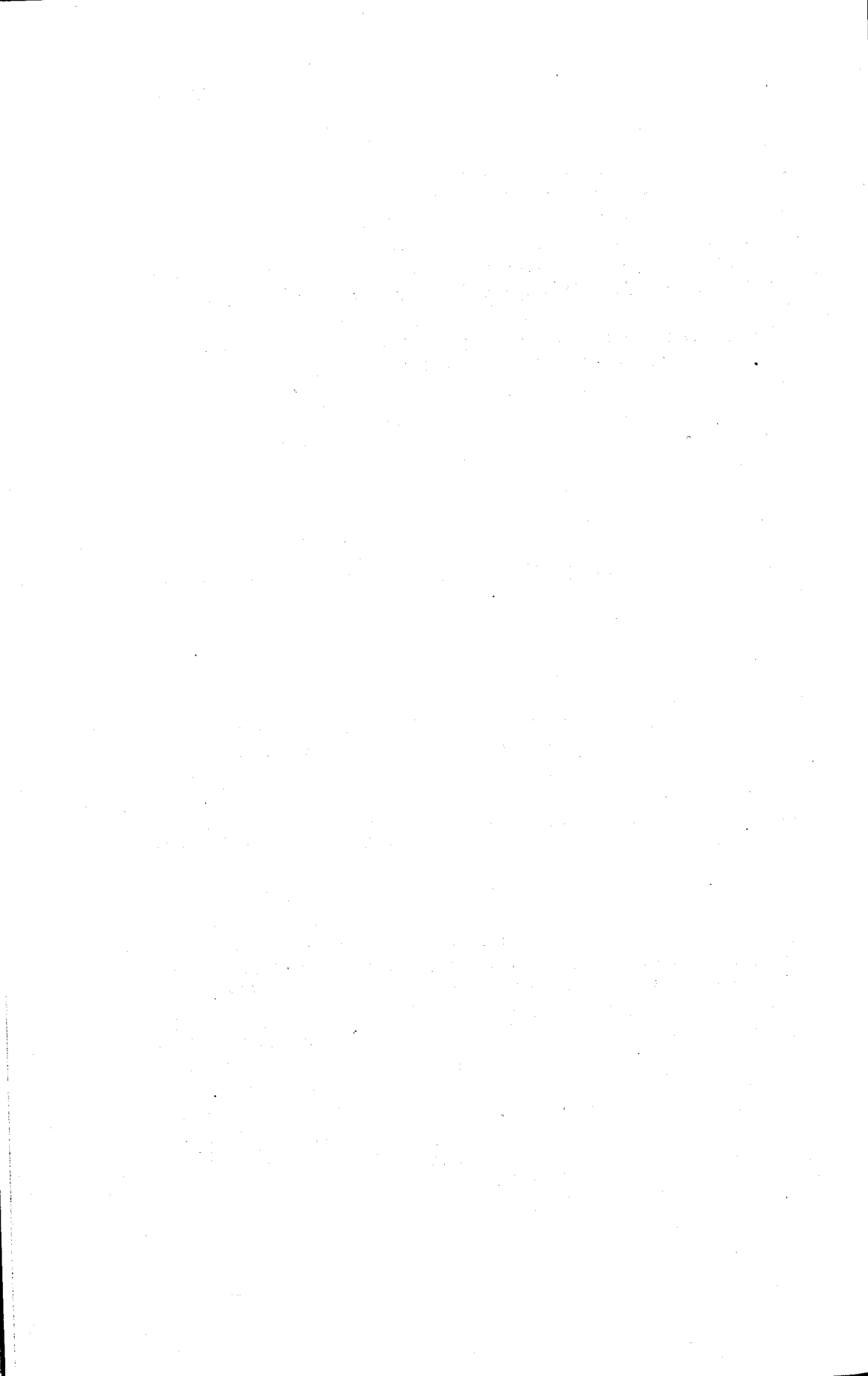
Objects and Relations: In defining *SEDGE*, *CBND* and (informally) *LINE* we have deployed only those relationships defined earlier in the paper (we could hardly have utilised any others). These relationships are characterised as being defined on, and in terms of, *positions*. If we regard position as an object (as suggested in § 5.1) then the relationships defined in this paper are those which involve just position (s) as the objects in terms of which they are defined. The informal discussion of compound forms (§ 5.3) makes frequent reference to *SEDGE*. Underlying every *SEDGE* we may presume a *COORD*, and it may be plausible, therefore, to consider *COORD* as the object necessary to the formulation of these ‘higher’ order relations ‘join’, ‘touch’, etc.

In terms of the theories of generative grammar espoused by Chomsky, it is tempting to identify these higher-order relations as those involved in co-ordinate constructions (Chomsky 1965, p. 134 and Note 7, p. 224), with the consequence that the lower-order relations (those defined on positions) might be identified with intra-sentential structure. Thus the inter-word and inter-phrase relationships, reflected in the use made of lexical substitution, might be regarded as *relations* defined on words and phrases as objects. The question which then arises is ‘why the distinction between the notational system for inter-sentential co-ordination (generalised T-rules) and intra-sentential co-ordination (strict subcategorization, selectional rules, and lexical substitution)?’

I would argue that the Aspects answer (which makes a distinction)

obscures a real uniformity (and therefore an economy) in the metalinguistic apparatus, and may well be obscuring our formal grasp of the linguistic significance of the word. Thus the analogy between picture points (positions) and words breaks down precisely because a word has a very complicated definition (its lexical entry) which from our standpoint would brand it as an *OBJECT*. To grasp an Object is to grasp the relationships which underly it. To regard words as sentences (complex objects) would perhaps be one way to tackle the anomaly underlying say '*phonophone' (see Chomsky 1965, p. 187).

These speculations may prove empty; what they point to, however – as does the rest of this paper – is the necessity for a clearer grasp of the distinction between relations and objects in the formulation of syntactic theories.



On the Construction of an Efficient Feature Space for Optical Character Recognition

A. W. M. Coombs

G.P.O. Research Department,
London NW2

INTRODUCTION

The particular form of character recognition we shall be dealing with here is that of identifying multi-font typed or machine-printed figures and letters (usually called alphanumerics), for the purpose of postal sorting. In such an application, special conditions apply: the characters (in a code group in our case) may be of very poor quality and in a wide range of styles, but the accuracy of identification is still to be high – about one error in 300 envelopes is tolerable, which, on the assumption of independence between characters of the code, means not more than 1 error per 2000 characters. As the number of characters in the code group has been reduced (for other reasons) to an absolute minimum, there is little if any redundancy; no help is to be gained from context. These are difficulties, and formidable ones at that. But the problem has its easier sides: the speed required is not high in electronic terms, being limited by mechanical handling difficulty to one envelope – about six characters – every 60 milliseconds, and although identification accuracy is to be maintained, it is permissible for the machine to reject characters as unreadable and divert the envelopes for hand-sorting in say 20 per cent of the cases (though naturally we wish to keep this figure as low as possible). There are thus two basic design limitations: 20 per cent of the envelopes may be rejected for hand-sorting, but of those not so rejected, less than 0.5 per cent are to be mis-read.

Other problems involved in the sorting operation, such as envelope handling (the 'hand' simulation), the extraction of the code from the envelope (the 'eye' simulation), the location of the code within the address, and the separation of MS from typed mail will not be discussed. It will be assumed that there exists within the machine a space-quantised and grey-level dichotomised pattern representing a probably degraded version of an alphanumeric character in some type font or other, that the character is correctly centred

on a test matrix (which can be a major problem in its own right) and may well have been subjected to a prior clean-up operation, and that the problem is to identify this character, or reject it, within the design limits specified above.

TEMPLATES

Type-written characters would appear, at first sight, to be of fixed size and shape – sufficiently so, at least, to lend themselves to a simple ‘template matching’ technique. In such an approach, the machine would have a set of templates or ‘masks’, and would look through the set for one which was ‘something like’ the character being examined. ‘Something like’ is, however, far too vague a condition to be implemented directly by machine; it is a human assessment, and one which is by no means understood. How do we measure the degree of ‘something likeness’ which something bears to something else? For MS characters, with their infinite variety, the thing simply cannot be done in this way, and in fact the fixed size and shape of typed characters are more apparent than real; there are scores of fonts, varying in thickness, angle of presentation, spacing, height and style.

A more reasonable approach is clearly to break down the patterns into sub-patterns or ‘features’, which are less variable within categories and to re-describe the patterns in terms of these features. The result is the 3-layer machine.

THE 3-LAYER MACHINE

It is a remarkable fact that research workers in the field of pattern recognition have almost without exception, and irrespective of where they started, eventually proposed a 3-layer machine, typified in figure 1. Note particularly the initial work of Taylor (1956) and the later development by Rosenblatt (1962). Such a machine possesses a sensory layer, or retina, on which the pattern to be recognised is implanted (possibly having undergone some degree of pre-processing), an association layer, or feature list, where the pattern is re-mapped or described in terms of its characteristic features, and a response layer where the verdict is finally given as to which category of patterns the unknown pattern belongs to. The feature extraction process takes place between the first two layers, and it may be highly elaborate, involving several layers of investigation in itself, or it may be simply an assessment of the activity of groups of retinal points, called n -tuples. Following the association layer there is a discriminator, usually a set or sets of weights, which operates on the association cells to give the requisite scoring differentials for the several categories in the response layer.

The discriminant weights are often obtained by an adaptive process which may conveniently be called ‘learning’. Indeed, there is a strong link between the 3-layer machine and the biological brain as neurologists are describing it, brought out by the expressions ‘retina’, ‘sensory layer’, ‘cell’ and so on, and illustrated in figure 1.

The greatest diversity in the machines designed to date has been in the nature of the features chosen, and it is towards the rationalisation and systematisation of this one process that the discussion here is directed. We shall be thinking specifically of retinal n -tuples, but the general principles evoked are of much wider application. Throughout, the approach will be that of linear vector spaces, wherein a set of N properties is represented by a vector in N -dimensional space. The sensory layer will have S cells, which may be thought

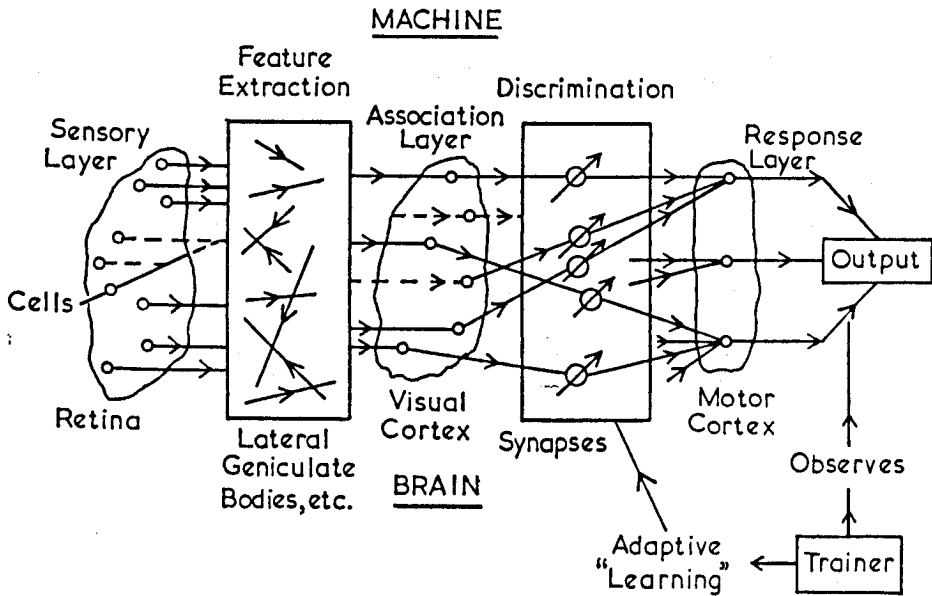


Figure 1. The 3-layer machine

of as the co-ordinates of an S -dimensional space, the association layer (or feature space) will have F cells (or F dimensions), and the response layer will have C cells, one per category. There will be a set of training characters, say E examples in each of the C categories and a set of test characters to assess the effectiveness of the machine designed around the training characters.

THE ASSOCIATION LAYER

Our discussion here is going to centre on the association layer, so we must be clear as to its function. On it, the characters which are to be separated are re-drawn in terms of their features, and the ideal choice of features would provide the following conditions:

1. Differences between characters in one category would be reduced.
2. Differences between characters in different categories would be enhanced.

PATTERN RECOGNITION

3. Accuracy would be improved by the use of as many different ways of describing the characters as might be found desirable.
4. The re-mapped characters would be in a form which made them easily separable by categories in the ensuing discrimination stage.

For the purposes of 3 above, a retina of S points will yield up to $(3^S/2)$ possible descriptors, each descriptor being an n -tuple of excitatory and inhibitory points. The denominator 2 reflects the fact that each n -tuple has a partner in phase-reverse, from which no extra useful information is derivable.

Let us now repeat the above conditions in terms of linear vector spaces, the patterns being considered as S -dimensional retinal vectors re-mapped into a feature space of F dimensions. So described, the conditions are:

1. The vectors of patterns in any one category should be a closely packed cluster in feature space, fewer in number than the original set in retina space.
2. The clusters of vectors representing different categories of patterns should be widely separated from each other.
3. The dimensionality of the feature space should be increased to give a large enough Hamming distance (HD) between typical category vectors.
4. Consider the tips of the pattern vectors as sets of points in feature space. Then the convex hull¹ of the points proper to any one category, and the convex hull of the points proper to all the other categories should be non-overlapping. This is the condition for linear separability.

It is the custom of designers of pattern recognition systems to start with the patterns on the retina, and to proceed forward from there; that, after all, is the order things happen in Nature. The requirements of an ideal feature space as set out above have not been used as a design criterion – not completely, that is. Many attempts have been made to satisfy the first requirement – clustering of within-category patterns – by the intuitive selection of features which seem to be useful, or by the use of vast numbers of random features, the less useful ones being then eliminated by some test procedure, for instance one involving an information theory measure (Kamentsky and Liu 1963). Some attempt has been made to satisfy requirements 2 and 3; Kamentsky and Liu made certain that typical category vectors were at an HD of at least 3, and Saraga *et al.* (1967) used a 45-dimensional feature space, divided into 10 overlapping sub-spaces (one per category) in each of which there was a guaranteed minimum HD of 1 between the relevant category and all others,

¹ The convex hull of a set of points in hyperspace is that minimal hypervolume enclosing them such that the straight line joining any pair of the points lies wholly within the hypervolume.

with a probable mean HD of perhaps 5. The third condition was heavily exploited by Uffelman (1962) in Conflex I; he used very large numbers of nearly random features at the sensory level. Apart from that, the tendency has been to use as *few* features as possible (the ultimate being of course $\log_2 C$). This is good mathematics but poor engineering, for no factor of safety remains. The fourth condition is not usually considered until the discriminant functions are being found; if it then transpires that the condition is not satisfied, other means (such as piece-wise linear separation) have to be used (Nilsson 1965), or more features found to provide the discrimination.

Human intuition is fallible, and there are so many possible n -tuple features that the best ones are unlikely to turn up by chance. It is the contention of this paper that starting with the retinal image is not the best way, however natural it may appear to be, and that since the feature space and the arrangement of the patterns in it are of supreme importance, then that space itself should be the starting point of design. We should begin in the middle and work both ways from there; that middle should be designed to satisfy the four feature space requirements given above.

It is of course true that we shall at a later stage require to find the n -tuples which divide the categories in the desired manner. This may not be as difficult as it sounds. At least, we shall know what the 'desired manner' is – and therefore what we are looking for.

THE FEATURE MATRIX

We may represent the patterns of feature space by a matrix, figure 2, in which the rows are the pattern vectors, one row per category, and any one column indicates the activity to be engendered by one feature among the various patterns. The pattern vectors are ideal, meaning that it will be our object by rule 1 above so to design the features in relation to the real patterns which have to be recognised, that all the patterns of a category shall on translation into feature space be represented by vectors that cluster closely around the ideal. Thus $a(ij)$ is a measure of the presence of the feature i in the ideal pattern of category j . For the purposes of this paper, the quantities a will be regarded as boolean – the feature is either there or it is not – though later work suggests that a third value 'Don't know' may be useful in centring the patterns. Further, the present two values of the a s will be $+1$ and -1 , rather than 1 and 0 ; under these conditions, the vectors of feature space will all be of the same length (\sqrt{F}) and that space itself will be the surface of a hypersphere centred at the origin of co-ordinates. In fact, the vectors will meet the hypersphere at points of contact with an inscribed hypercube of F dimensions.

For a pattern to be acceptable as belonging to a given category, its feature vector will be required to lie within a defining hypercircle (on the hypersphere surface) centred at the tip of the ideal feature vector for that category. If the feature vector lies in the no-man's-land between defining hypercircles, the pattern will be rejected as unreadable.

PATTERN RECOGNITION

The angles between ideal vectors taken in pairs are clearly to be large; the cosines of these angles are proportional to the scalar products of the vectors. In addition, the distances between ideal vector tips measured round the circumference of the hypercircle should be large relative to the radii of the category-defining hypercircles. In effect, these conditions imply large HDs between ideal vectors, and that implies a design constraint (in a minimal sense) on the dimensionality of the feature space. The ideal vectors will

Categories	Features									
	1	2	3	4	—	—	i	—	—	F
1	a_{11}	a_{21}	a_{31}	a_{41}	—	—	a_{i1}	—	—	a_{F1}
2	a_{12}	a_{22}	a_{32}	a_{42}	—	—	a_{i2}	—	—	a_{F2}
—	—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—	—
j	a_{1j}	a_{2j}	a_{3j}	a_{4j}	—	—	a_{ij}	—	—	a_{Fj}
—	—	—	—	—	—	—	—	—	—	—
—	—	—	—	—	—	—	—	—	—	—
C	a_{1C}	a_{2C}	a_{3C}	a_{4C}	—	—	a_{iC}	—	—	a_{FC}

Figure 2. The feature matrix

preferably be evenly and therefore symmetrically disposed in feature space, and the dimensionality will be a function of the desired accuracy, the probability of any one term $a(ij)$ being correct — that is, ideal — the permissible reject rate and the number of categories. The relevant formulae are exhibited in Appendix 1. The linear separability requirement is always satisfied by hypercircles on a hypersphere.

THE COMBINATORIAL MATRIX

If the number of categories is neither too large nor too small, say about 5, there is little difficulty in specifying a feature matrix to satisfy the conditions. Each column must be made to give a systematically different dichotomy among the categories. For instance, figure 3 illustrates a feature matrix for 5 categories, formed by making each column give a positive dichotomy for a different 2 from 5, 10 columns in all. Ideal vectors are separated by an angle $\arccos(-1/5)$ and an HD of 6; the category hypercircles would embrace all vectors at HD 2 or less from one of the ideal vectors (radius $2\sqrt{2}$), or for

greater accuracy in recognition, at the cost of higher rejection rate, all vectors at an HD of 1 or less.

The 10-dimensional feature space of figure 3 can be made to accommodate a sixth category of patterns, while still retaining the same angles and HDs between vectors, by the addition of a '10+' vector as shown in figure 4, or

Categories	Features									
	1	2	3	4	5	6	7	8	9	10
1	+	+	+	+	-	-	-	-	-	-
2	+	-	-	-	+	+	+	-	-	-
3	-	+	-	-	+	-	-	+	+	-
4	-	-	+	-	-	+	-	+	-	+
5	-	-	-	+	-	-	+	-	+	+

Figure 3. Feature matrix for 5 categories, 10 features. HD=6; mutual angle= $\cos^{-1}(-\frac{1}{6})$

Categories	Features									
	1	2	3	4	5	6	7	8	9	10
1	+	+	+	+	+	+	+	+	+	+
2	+	+	+	+	-	-	-	-	-	-
3	+	-	-	-	+	+	+	-	-	-
4	-	+	-	-	+	-	-	+	+	-
5	-	-	+	-	-	+	-	+	-	+
6	-	-	-	+	-	-	+	-	+	+

Figure 4. Feature matrix for 6 categories, 10 features. HD=6; mutual angle= $\cos^{-1}(-\frac{1}{6})$

Categories	Features														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	+	+	+	+	+	-	-	-	-	-	-	-	-	-	-
2	+	-	-	-	+	+	+	+	-	-	-	-	-	-	-
3	-	+	-	-	-	+	-	-	-	+	+	+	-	-	-
4	-	-	+	-	-	-	+	-	-	+	-	-	+	+	-
5	-	-	-	+	-	-	-	+	-	-	+	-	+	-	+
6	-	-	-	-	+	-	-	-	+	-	-	+	-	+	+

Figure 5. Feature matrix for 6 categories, 15 features. HD=8; mutual angle= $\cos^{-1}(-\frac{1}{15})$

six categories could be deployed in a 2-out-of-6 matrix as in figure 5. This has 15 features, a mutual angle of $\arccos(-1/15)$ and an HD of 8. Again, the lower rows of the matrix, figure 5, could be used to separate five categories with greater certainty, although with somewhat less efficiency, than is obtainable from the matrix figure 3, because of greater HD.

PATTERN RECOGNITION

And so on. The method for these 'combinatorial' matrices is obvious; it breaks down either when the number of categories is low, or when that number is high, on the one hand because it does not give enough features, and on the other because it gives too many. We shall now consider these two limitations.

THE REPEATED MATRIX

If the number of categories is low – say 3 – then there are only a few possible dichotomies of combinatorial type, and feature space is low dimensioned. This means low HD and hence loss of accuracy, although of course it is wholly possible (and likely) that the original retinal dichotomies themselves may be more reliable with a smaller number of categories, and so *require* less

Categories	Features								
	Field 1			Field 2			Field 3		
	1	2	3	4	5	6	7	8	9
1	+	-	-	+	-	-	+	-	-
2	-	+	-	-	+	-	-	+	-
3	-	-	+	-	-	+	-	-	+

Figure 6. Feature matrix for 3 categories, 9 features. HD=6; mutual angle= $\cos^{-1}(-\frac{1}{3})$

HD. If they are not, the inference is that the categories include some wide deviants, and these might well be classed as separate categories. Or dichotomies could be applied to a restricted field of the retina, and repeated for other fields, figure 6. An experiment something like this was described by Roberts (1960), though it was not then put forward in these terms.

We have not given much attention to this limitation, since it is not on the face of it our problem. We are much more concerned with the other extreme, where the number of categories is high.

THE CALTROP MATRIX

Information theory suggests that each feature should appear in about half the categories, and that each category should typically contain about half the features. The geometrical approach given in this paper confirms the point and shows why it is important; if it is not met, the hypersphere surface is not used efficiently, the ideal pattern vectors tending to bunch into a small part of it. A seventh category vector of 15 '+' units added to the matrix figure 5 makes an angle of $\arccos(-1/3)$ with each of the others, themselves mutually separated by the angle $\arccos(-1/15)$. But as the number of categories increases, matrices designed on the combinatorial plan and also

satisfying the efficiency condition rapidly become unmanageably large. Thus, with 11 categories and a 5-out-of-11 procedure, the required number of features is 462, with mutual angles $\arccos(-1/11)$ and HDs of 252; a twelfth 'all +' vector will make the same angle with all the others, and be at the same HD from them. With a 3-out-of-11 procedure, 165 features are needed, with mutual angle $\arccos(+7/55)$ and HD of 72; the twelfth vector is at $\arccos(-5/11)$ and HD 120. With a 1-out-of-11 procedure, only 11 features are needed, the mutual angle is $\arccos(+7/11)$ and the HD 2, but the twelfth vector is at $\arccos(-9/11)$ and HD 10. Clearly, the combinatorial approach is either impossibly uneconomic or highly inefficient, or both, as the categories increase. It would seem that there should be a way of selecting a sub-set of (in this case) the 462 features, which will still retain the angular separation. There is indeed a way, though it is of synthesis rather than analysis; such a sub-set may be derived from an orthogonal matrix of appropriate size.

+	+	+	+	+	+	+	+	+	+	+
-	+	-	+	+	+	-	-	-	+	-
-	-	+	-	+	+	+	-	-	-	+
+	-	-	+	-	+	+	+	-	-	-
-	+	-	-	+	-	+	+	+	-	-
-	-	+	-	-	+	-	+	+	+	-
-	-	-	+	-	-	+	-	+	+	+
+	-	-	-	+	-	-	+	-	+	+
+	+	-	-	-	+	-	-	+	-	+
+	+	+	-	-	-	+	-	-	+	-
-	+	+	+	-	-	-	+	-	-	+
+	-	+	+	+	-	-	-	+	-	-

Figure 7. Caltrop matrix for 12 categories, 11 features. HD = 6: mutual angle = $\cos^{-1}(-1/11)$

It is not difficult to show that a boolean orthogonal matrix must be of an order which is a multiple of 4 (Paley 1933); it is probable, though as far as we know not yet proved, that such a matrix can be constructed of *any* order which is a multiple of 4. It can certainly be done up to order 112, and with very few exceptions up to order 200. Now any such matrix is either already in a form where one complete column and one complete row are both 'all +', or it can be converted into this form by using the self-evident fact that a set of orthogonal vectors remains an orthogonal set if any number of the vectors be reversed in sign. The 'all +' column then represents a feature which possesses no power of discrimination between the categories, and it can be eliminated. The result is a matrix of $4m$ vectors in $(4m-1)$ space, the array of vectors possessing a mutual angle of $\arccos(-1/(4m-1))$ and a mutual HD of $2m$. For $m = 3$, the matrix is shown in figure 7, covering 12 categories with 11 features; the mutual angle is $\arccos(-1/11)$ and the HD 6, and this

PATTERN RECOGNITION

matrix clearly contains the required selected set from the 462 features specified using the combinatorial approach.

Vector sets of this sort are generalisations into n -space of the 4-spiked mediaeval 'caltrop' used to impede the advance of cavalry, and it is accordingly proposed that they shall receive this generic name. They give the most efficient possible way of filling the surface of a hypersphere with hypercircles. It is in fact possible to generate a caltrop of n spikes for any $(n-1)$ space, but the caltrop can be boolean only if the n -space will support a boolean orthogonal set (which probably means 'is of dimension $4m$ '). Thus, the 3-spiked caltrop in 2-space, figure 8, can never be boolean, whereas the 4-spiked caltrop in 3-space, figure 9, both can be and is.

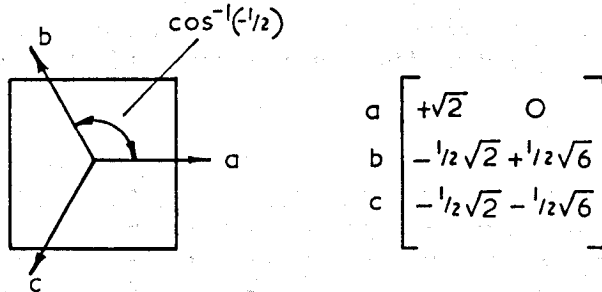


Figure 8. Caltrop and corresponding matrix in 2-space (non-boolean)

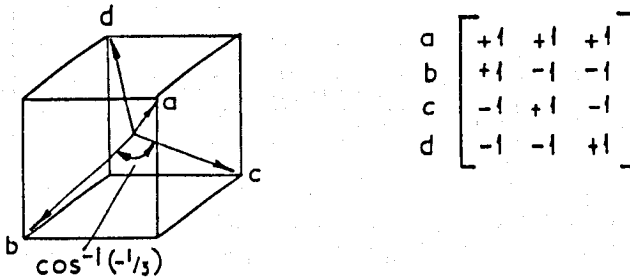


Figure 9. Caltrop and corresponding matrix in 3-space (boolean)

Consider the caltrop matrix, figure 7. In it, all the rows below the second are 'shifted' repeats of the second, and all pairs of rows have mutual scalar product (-1) . In fact, the typical row is a digital sequence with pseudonoise properties, and such sequences can be used to construct caltrops (see Everett 1966). This form of synthesis requires that the period of the sequence be p , where $p = 3 \pmod{4}$, and that either:

1. p is prime (e.g. $p = 11$)
- or 2. p is $2^k - 1$ (e.g. $p = 15$)
- or 3. p is lm , where l and m are primes differing by 2, and having a common primitive root (e.g. $p = 35 = 7 \times 5$, 7 and 5 possessing the common primitive root 3)

A great many combinations of these and other matrices are possible, but we cannot pursue the topic here. We should, however, refer to the fact, unknown to us when we derived the caltrop, that the properties of these very useful matrices have apparently been used in the construction of error-correcting codes (Peterson 1961).

THE DICHOTOMIES

Any one column of the feature matrix gives the dichotomy required to be effected by one feature among the categories. The aim will be to achieve that dichotomy by a linear separation with an n -tuple which does not contain too many points.

Now, each and every pattern of the input training set can be represented by a linear inequality of S variables, those variables being the weight to be attached to the retinal points, and the coefficients in the inequality being $+1$ or -1 depending on the activity of the retinal points. The problem reduces to that of solving a set of simultaneous linear inequalities, CE in number with S variables, with the added refinement that as many of the weights as possible are to be so close to zero that they can be ignored (thus reducing the size of the n -tuple).

Put like that, a solution method using the Linear Programming techniques of operational research is suggested; we are at present investigating this possibility. To obtain the results exhibited in Appendix 2, however, we used a computer simulation of 'Adaline' (Widrow 1962), which is a machine ideally suited for performing dichotomies of this very type. The principle was that we converged to a set of weights on the training set of characters, using the Adaline algorithm, eliminated those weights (i.e., retinal points) which were of low importance, re-converged, eliminated again, and so on. We expected that the ' n ' of the n -tuple with which we were eventually left would for each feature be rather larger than the number of categories, but considerably smaller than the number of patterns in the whole training set. And so it proved. For patterns on a retina of 200 points, with 40 samples of each of 10 categories, groups of 30 to 50 points were sufficient to give the dichotomies. These figures can undoubtedly be improved upon with more experience; it is important to observe that because of the built-in Hamming distance in the feature matrix, it is not necessary for the test dichotomies to be 100 per cent correct. The curves, figures 10 and 11 of Appendix 1, illustrate the point.

Acknowledgement

Acknowledgement is made to the Senior Director of Development of the General Post Office for permission to publish this paper.

REFERENCES

- Everett, D. (1966) Periodic digital sequences with pseudonoise properties. *G.E.C. Journal*, 33, no. 3, 115-26.
 Kamensky, L.A. & Liu, C.N. (1963) Computer-automated design of multifont recognition logic. *I.B.M. Journal*, 7, 2-13.

PATTERN RECOGNITION

- Nilsson, Nils J. (1965) *Learning machines*, chap. 7. New York: McGraw-Hill.
- Paley, R. E. A. (1933) On orthogonal matrices. *J. Maths. Phys.* **12**, 312-20
- Peterson, W. W. (1961) *Error-correcting codes*, chap 5. New York: MIT/Wiley
- Roberts, L. G. (1960) Pattern recognition with an adaptive network. *IRE International Convention Record*, **8**, pt. 2, 66-70.
- Rosenblatt, F. (1962) *Principles of neurodynamics*, pt. 2. Washington D. C.: Spartan Books.
- Saraga, P., Weaver, J. A., & Woollons, D. J. (1967) Optical character recognition. *Philips Technical Review*, **28**, no. 5/6/7, 197-202.
- Taylor, W. K., (1956) Electrical simulation of some nervous system functional activities. *Information Theory - 3rd London Symposium*, pp 314-28 (ed. Cherry, C.). London: Butterworth.
- Uffelman, M. R. (1962) Conflex I - a conditioned reflex system. *IRE International Convention Record*, **10**, pt. 4, 132-42.
- Widrow, B. (1962) Generalisation and information storage in networks of adaline neurons. *Self-Organising Systems - 1962*, pp. 435-61 (eds Yovitz, Jacobi and Goldstein). Washington D.C.: Spartan Books.

APPENDIX 1

Mis-sort and reject rates as functions of dichotomy reliability

Let the input characters belong to C categories, and let the number of features be F . For a fully effective caltrop matrix,

$$C = F + 1.$$

Let the feature space be used uniformly, so that every pair of ideal vectors possesses M common features and a mutual Hamming distance of H , where

$$F = M + H.$$

Let the probability of obtaining a correct dichotomy at any given point in the future matrix, with a test pattern, be p ; define k as

$$k = \frac{1-p}{p}.$$

Let the number of features permitted to be different from an ideal before a pattern is rejected be r , this specifying a hypercircle radius ($= 2\sqrt{r}$); the corollary is that any pattern at HD r or less from an ideal vector is allotted to the category of that vector. Then if the categories have equal likelihood of appearance, and a pattern is certainly being presented, the performance of the matrix can easily be derived by a Bayes analysis, as follows.

Consider one particular ideal pattern vector, A . The probability that the test pattern is in the A category is $1/C$, that being in that category it produces a vector within the prescribed hypercircle is

$$p^F \sum_0^r \binom{F}{x} k^x.$$

The possibility that the test pattern is not in the A category is $(C-1)/C$;

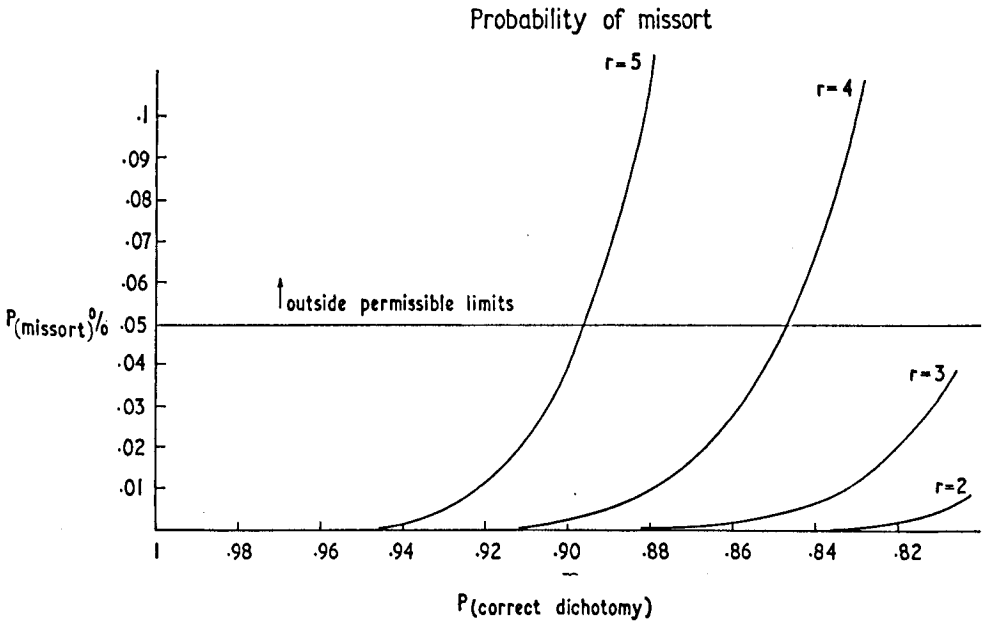


Figure 10. 24 categories. Probability of mis-sort

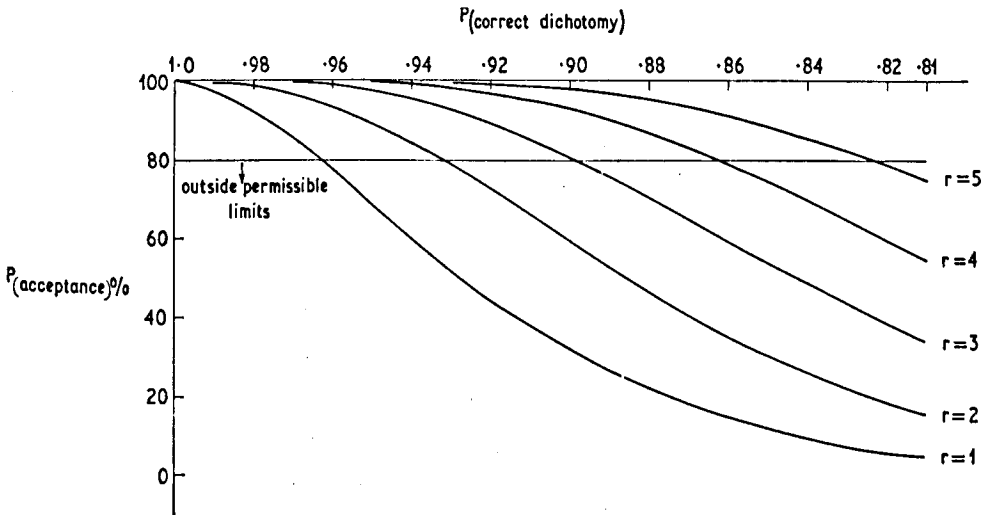


Figure 11. 24 categories. Probability of acceptance

PATTERN RECOGNITION

that not being in that category it still produces a vector within the A hyper-circle is

$$p^F \sum_0^r \binom{M}{y} \sum_0^{r-y} \binom{H}{z} k^{H+y-z}.$$

Hence, the probability that the test pattern is allotted to a category at all is Q , where

$$Q = p^F \left\{ \sum_0^r \binom{F}{x} k^x + (C-1) \sum_0^r \binom{M}{y} \sum_0^{r-y} \binom{H}{z} k^{H+y-z} \right\} \\ = (\text{say}) p^F (X+Y).$$

Given that the pattern is allocated, the probability that the allocation is wrong is P , where

$$P = \frac{Y}{X+Y}.$$

The graphs, figures 10 and 11, show mis-sort and rejection rates for a caltrop of 23 dimensions (24 categories) worked out according to the above formulae.

APPENDIX 2

The following experiments have now been carried out, testing the theory described in the paper.

Experiments

A set of 800 multi-font typed characters, 80 samples of each of the 10 numerals, taken with deliberately little regard for quality, was used as the input. Half the characters (40×10) constituted a 'Training' set, and the nearest suitable caltrop (11 features by 12 categories) was used, that is, 11 Adalines were converged according to the dichotomies shown in figure 7, but with no inputs labelled categories 11 and 12. Convergence was continued with the elimination of retinal points appearing to have little significance (Adaline weights approx. = 0) until the 11 Adalines had 50 inputs each. This number could have been reduced further had not time precluded it.

At this point, then, the feature layer consisted of 11 cells performing the dichotomies indicated in figure 7, with each of the categories 1 to 10 containing 40 training samples of the numerals 1 to 0 in multi-font, and categories 11 and 12 left arbitrary. The rest of the characters (40×10) were now shown to the machine as a 'Test' set; they were, of course, previously unseen by the machine. The results are given below.

Results

There is a 'trade-off' between reject rate and mis-sort rate in terms of the acceptable Hamming distance between a test vector and the ideal vector for the appropriate category. With the 11×12 caltrop, the HD between categories

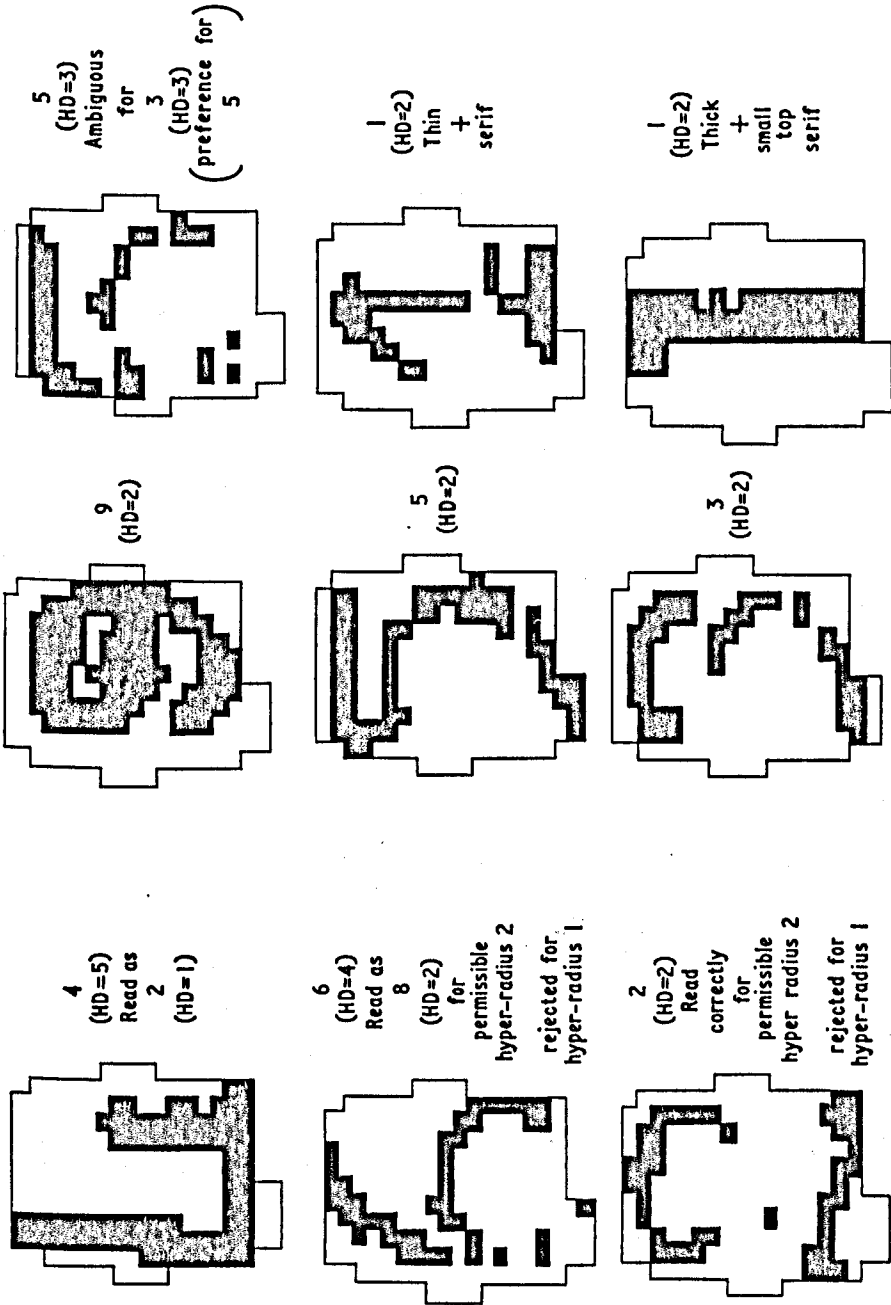


Figure 12. Assortment of characters tested and results obtained. Caltrop in 11-dimensional feature space.

PATTERN RECOGNITION

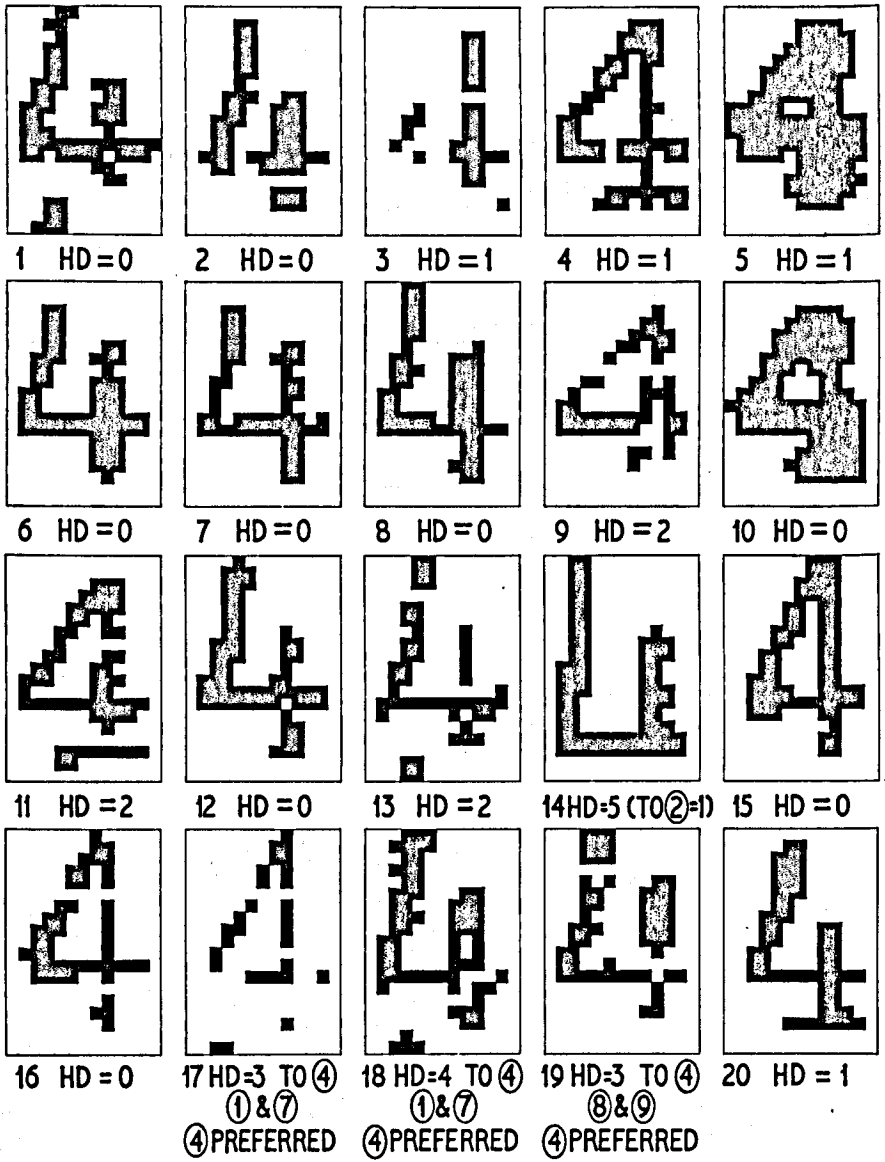


Figure 13. First 20 test set 4s-machine responses (HD from ideal 4)

is a uniform 6, so a distance of 3 from an ideal vector would in general give ambiguity, an allowable distance of 2 would give a low reject rate but medium mis-sort, while an allowable distance of 1 would give medium reject and low mis-sort.

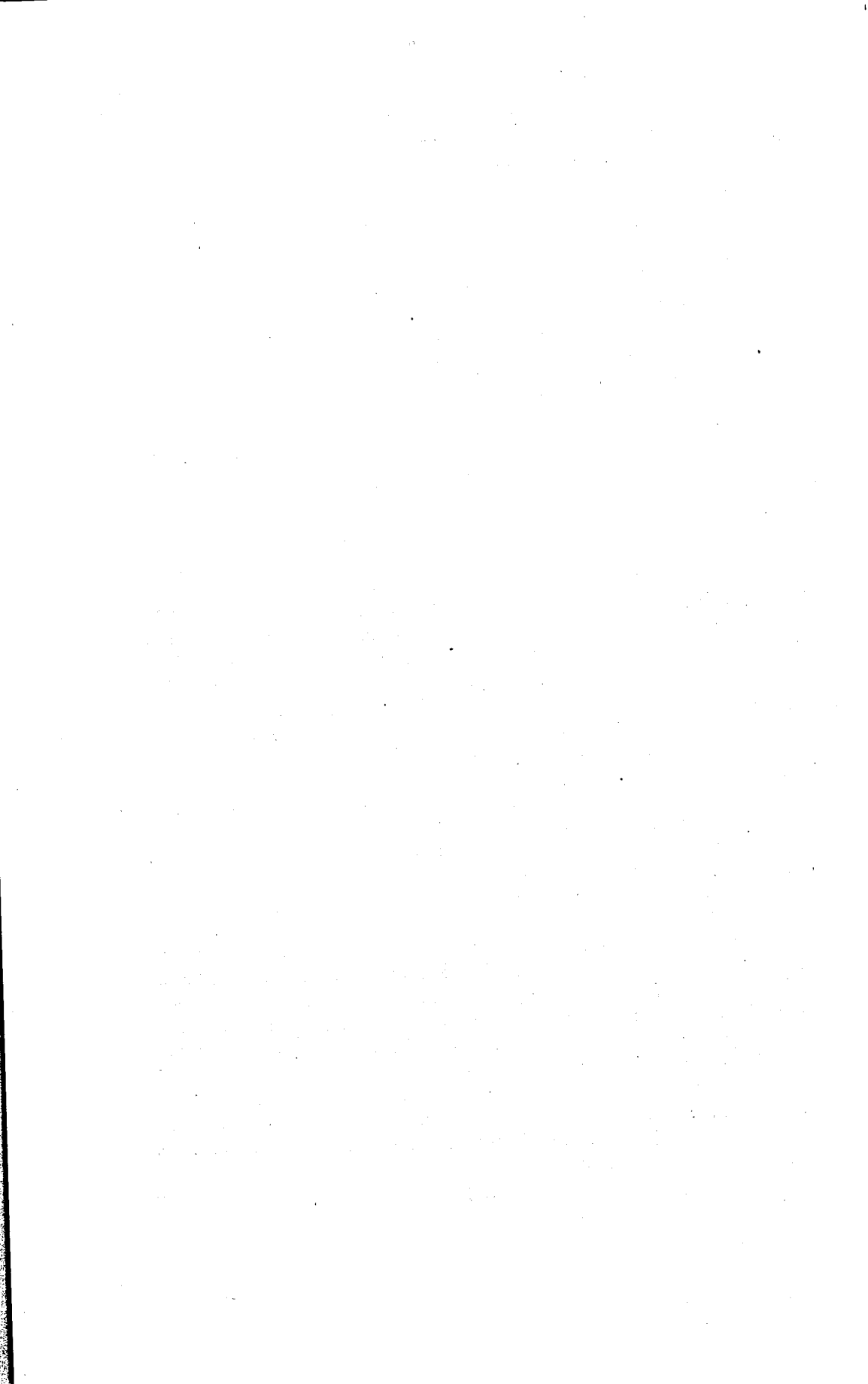
The figures actually obtained in the 'Test' phase with the unseen 400 characters were as follows:

1. Dichotomies performed correctly = 93%
(312 out of 4400 incorrect)
2. With HD allowed to be 2
 - Correct categorisation = 93.75%
 - Rejected = 4.25%
 - Mis-sorted = 2.0%
3. With HD allowed to be 1
 - Correct categorisation = 80.25%
 - Rejected = 19.50%
 - Mis-sorted = 0.25%
 - (i.e., one out of the 400 characters)

Notes

Figure 12 shows an assortment of the Test characters, chosen rather to illustrate the achievements and failures of the machine than to typify the set. Thus, the first '4' was the only character to fail on HD = 1; it was a sample in a very large font which had not appeared at all in the training set, and it overlapped the matrix. Even so, analysis of the result scores showed that a very small refinement in the technique of scoring would have resulted in its rejection rather than its mis-sorting. To the purist, the words 'hyper-radius 2' should be 'hyper-radius $2\sqrt{2}$ ' and similarly for 'hyper-radius 1'; the hyper-cubes in our boolean space are of edge 2 units, and any two of which edges are mutually perpendicular.

Figure 13 is more useful to show the typical quality of the test-set characters. It illustrates the first 20 samples of the numeral 4 in the test-set, including the only character to fail with HD = 1 (sample Number 14). Open and closed types, thick and thin, and highly defective samples are all shown, together with the HD from the ideal vector for category 4. The 14th sample was at HD = 5, and therefore mis-sorted. The 18th sample was at HD = 4, but in fact was at such a point of the hypersphere surface that it was no nearer any other ideal vector. The 17th and 19th samples were at HD = 3, and rejected as ambiguous; analysis of the scores showed that the preference in each case was for 4 - that is, if the association cell with the lowest score is deleted, then each time the HD from 4 is reduced, whereas the HDs from the alternative categories are unaffected. The suggestion is that a discriminating stage, with 'learned' weights, would have improved the performance. All the other samples were at HD 1, 2, or 0.



Linear Skeletons from Square Cupboards

C. Judith Hilditch

Medical Research Council,
Clinical and Population Cytogenetics
Research Unit

INTRODUCTION

The problem of reducing the line-like elements of a digitized picture to idealized thin lines is of general interest in pattern recognition. As early as 1957 the idea of obtaining a thin-line representation of certain patterns was suggested (Kirsch *et al.* 1957); recently McCormick (1963) and Narasimhan (1964) have described computer programs for doing this (for use in particular on bubble chamber photographs), and similar work has been done in character recognition, for example by Deutsch (1967). Blum (1964) has put forward an approach for dealing with more general shapes. In this the boundary of a shape is considered as being the source of a wavefront. The points at which wavefronts originating at different parts of the boundary first meet form a 'skeleton' which, with a function giving the time taken for the wavefront to reach each point of the skeleton, completely defines the original shape. Programs for generating this skeleton for digitized pictures have been described by Rosenfeld and Pfaltz (1966), and also by Philbrick (1966). A technique which we at the MRC have implemented for reducing line-like shapes to idealized thin lines is similar to the last of these, in that it involves working inwards from the boundary of the shape under consideration, removing all points except those which are considered part of the skeleton. It is also similar to a stripping routine described by Izzo and Coles (1962) and Preston (1961) in a rather different context.

AIMS

The problem with which we have been primarily concerned is the automatic analysis of chromosome spreads, a typical example of which is shown in figure 1.

The aim of the algorithm described in this paper is to reduce such a picture to a 'skeleton' of idealized thin lines which satisfy not only the obvious

requirement that they should lie approximately along the centre of each line-like part of the picture but also satisfy, as nearly as is possible in a discrete space, the definition of a line as 'that which has length without breadth' while still retaining the connectivity of the original.

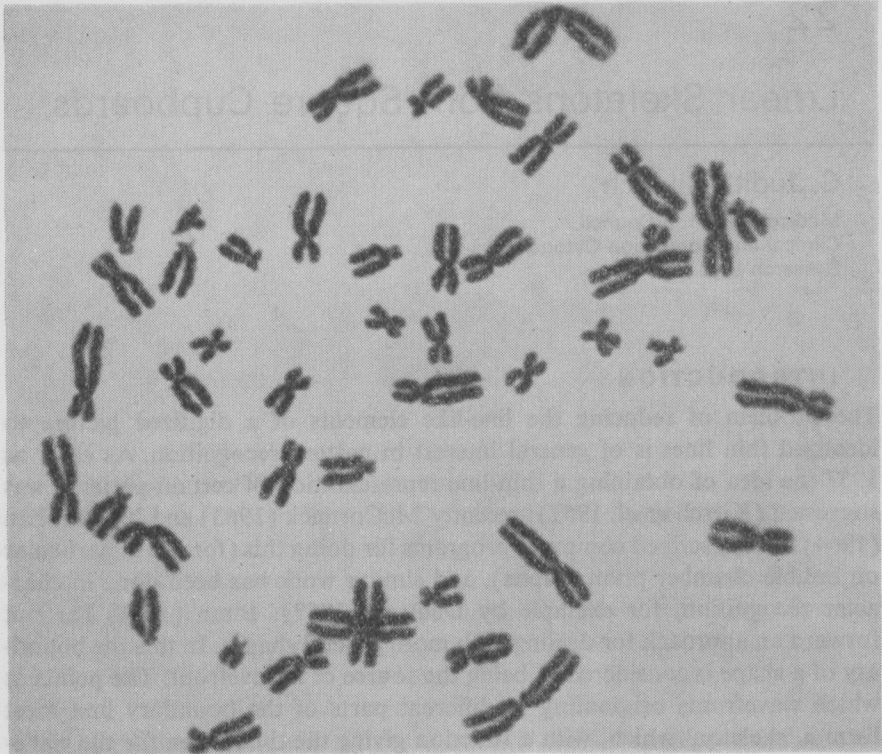


Figure 1. A typical human chromosome spread

THE PICTURE

Before describing how this has been done it is necessary to describe more fully the form which the picture takes. Let J be the set of all pairs of integers. Then a picture is defined as a function f on a subset P of J . In particular P is usually rectangular, i.e., $P = \{(i, j) \mid 1 \leq i < m, 1 \leq j < n\}$, and the elements of P can be regarded as the elements of a matrix, or as points with integer coordinates in the Euclidean plane. Usually the function f will initially take values such that each element or point represents in some way the darkness of that part of the picture. Subsequent transformations of the picture will, however, alter the values of the points and hence of course their significance.

The work described here is concerned not with whole pictures, which in our case consist of about a quarter of a million points, but with sub-pictures, that is restrictions of f to a subset Q of P ; formally we denote this by $f|_Q$. Q may for example be the set of all points of P with value greater than some threshold

or a connected component of such points. The problem which we are considering here is that of producing a skeleton for such a sub-picture which satisfies all the requirements for a skeleton listed above.

CONNECTIVITY

At this stage it is necessary to define what we mean by connectivity in the case of a digitized picture. Each point of a picture is considered to have eight neighbours, these being the eight points which differ from it by one in either or both co-ordinates. For convenience these neighbours are numbered n_1, n_2, \dots, n_8 as shown in figure 2. The numbers are taken modulo eight so that, for example, neighbour nine is the same point as neighbour one.

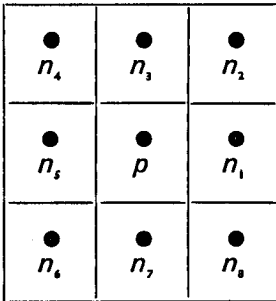


Figure 2. The eight neighbours n_1 to n_8 of a point p

Following Rosenfeld and Pfaltz (1966) we say that a subset of a digitized picture is *connected* if for any two points p and q of the subset there exists a sequence of points

$$p = p_0, p_1, p_2, p_3, \dots, p_{n-1}, p_n = q,$$

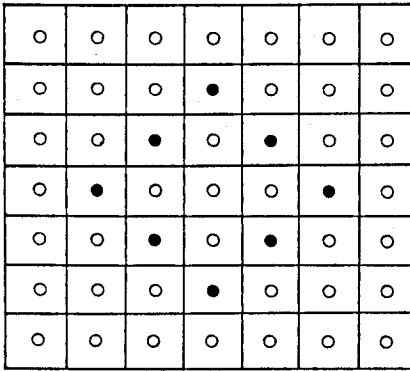
such that p_i is a neighbour of p_{i-1} , $1 \leq i \leq n$. This corresponds with the usual concept of connectivity in the Euclidean plane, if one considers the area defined by taking each point of the subset as the centre of a closed unit square.

However, as pointed out by Rosenfeld and Pfaltz (1966), if one then considers whether the complementary subset of the picture is connected, paradoxical situations can arise. For example, both the set of black points and the set of white points in figure 3a are connected, as is illustrated in figure 3b. This paradox is not resolved by considering a point as having only four neighbours, namely n_1, n_3, n_5 and n_7 . In this case we simply have the reverse situation that neither the set of black points nor the set of white points is connected.

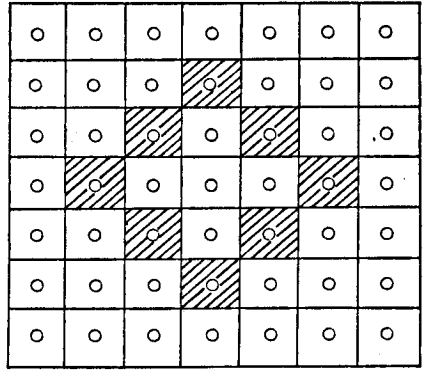
The paradox arises because the connectivity of a subset of the picture and the connectivity of its complementary subset are both being considered at the same time. In practice one is usually interested only in the connectivity of a

PATTERN RECOGNITION

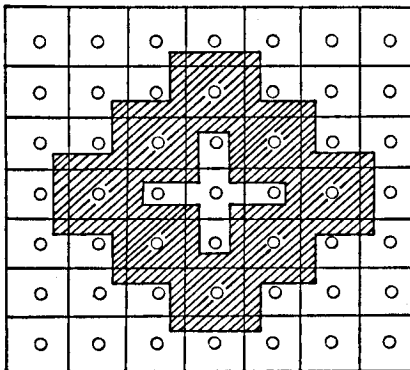
given subset, say the black points in figure 3a. In this case confusion can be avoided by considering the connectivity of the area defined by this subset in the Euclidean plane and the connectivity of the complementary area, rather than the area defined by the complementary subset.



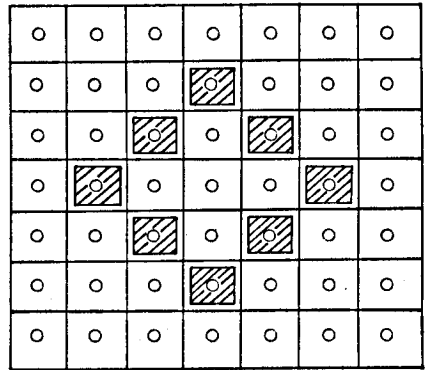
3a



3b



3c



3d

Figure 3. A paradoxical situation in which both the set of black points and the set of white points are connected. For a full description see text

This is made clearer by taking each point of the subset as the centre of a square with side slightly greater than the distance between adjacent points; the area defined by the set of black points is then given by the shaded part of figure 3c and clearly is connected, whereas its complement clearly is not. On the other hand the area defined by the set of white points (given by the unshaded part of figure 3d) is also connected and its complement is not. We choose to

consider each point of a given subset of the points of a picture as representing a square of slightly greater than unit area, in this way, and define the subset as connected if the area which it represents is connected in the usual sense in the Euclidean plane.

This is equivalent to defining connectivity on the neighbourhood basis with the points of the given subset each having eight neighbours n_1 to n_8 , but allowing the points of the complementary subset to have only four neighbours n_1, n_3, n_5 and n_7 .

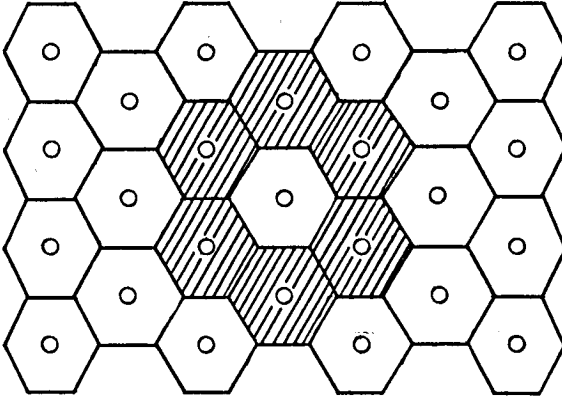


Figure 4. In the hexagonal case each point has six neighbours, and the paradoxical situations cannot arise

It may be interesting to note an alternative approach which avoids this difficulty altogether. This requires alternate lines of the picture to be displaced by half a unit, and each point taken to have six neighbours, as shown in figure 4. This is equivalent to considering each point as the centre of a hexagon, and in this case it is clear that the paradoxical situations do not arise.

REQUIREMENTS TO BE MET BY THE SKELETON

We are now in a position to consider in greater detail the requirements to be met by the skeleton and how these are satisfied. The requirements are:

1. Thinness

First, the skeleton is required to consist of thin lines. This is achieved simply by eroding away the subset by successively removing points which lie on its edge until all that remains is lines which are one point wide. The algorithm is intended for use on a general purpose digital computer in which the points must be treated sequentially and not in parallel. However, if the points are removed sequentially – a point being removed from the subset as soon as it has been found to lie on the edge of the subset – then subsequent nearby points will appear to lie on the edge and will be deleted in their turn. The

PATTERN RECOGNITION

result is that the skeleton will tend to be biased towards one side or the other of the original subset, depending on the order in which the points are taken.

2. Position

The second condition is that the idealized thin lines should lie along the centres of the line-like parts of the subset. To achieve this the process of removing points on the edge of the subset is made essentially parallel. A point is removed only if it lies on the edge of the initial subset, regardless of which other points have been removed. This means that several passes through the points of the subset are required. At each pass the outer layer of points is removed to give a 'thinner' subset for the next pass.

However, once this thinning process has reached a stage where some or all of the subset has been reduced to thin lines of points, these lines must not be thinned away to nothing. This is ensured by the next two conditions.

3. Connectivity

The third requirement is that the process should not alter the connectivity of the subset. This is achieved by testing each point that is to be removed to find whether its removal will alter connectivity. If it does, then the point is retained even though it lies on the edge.

However, if this is performed in parallel, a difficulty arises in the case where the subset has been reduced to a line that is two points wide, such as is shown in figure 5.

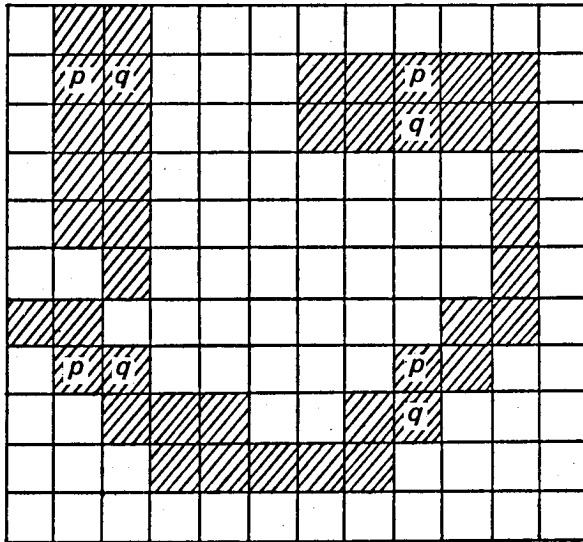


Figure 5. In each of the cases shown above neither the deletion of point p nor the deletion of point q will disconnect the area of shaded points; however, in each case the deletion of both p and q will,

In this case the removal of point p would not alter the connectivity of the subset, neither would the removal of point q . Unfortunately, the removal of both together most certainly will. This difficulty is overcome by taking advantage of the fact that the points are really being dealt with sequentially. Suppose, for example, that point p in figure 5 is tested before point q . Since the removal of p does not alter the connectivity of the subset, this point is removed. Subsequently, when point q is tested, the fact that p has been removed can be taken into consideration and the point q retained. This means that for a point to be removed, not only must its removal alone not alter connectivity, but if any one of its neighbours has been removed, then the removal of the two points together must not alter connectivity either.

4. Stability

Finally, it is necessary that as soon as a satisfactory skeleton, or part of a skeleton, is obtained, this should be stable and not be eroded away by subsequent passes. For most points of the skeleton this is ensured by the connectivity condition. However, a point which lies at the tip of a thin line can be removed without altering connectivity. An additional condition is therefore required to the effect that such a point may not be removed, otherwise the lines of the skeleton will gradually become shorter and shorter. Also, in order that an approximately circular subset should not disappear entirely, we need a condition which ensures that the last remaining point of such a subset is not removed.

THE ALGORITHMS

To meet these requirements the various algorithms that we have implemented each require several passes. At the start of a pass a subset Q of some sub-picture $f_0 | P$ is given¹; at the end of the pass a subset Q' of Q , which is one layer 'thinner', has been defined. Initially, Q is the subset to be reduced to idealized thin lines; the resulting Q' then becomes Q for the second pass, and so on; eventually a Q' is obtained which is a skeleton of the required form. Thereafter, no further points can be removed, and this fact is used to recognize when no more passes are required.

We have found it convenient to assume that the subset Q is defined as those points of P for which f_0 takes one of a given set of values, I , say, and that all other points of P take one of a set of values N . In particular, the picture usually takes the form of either a characteristic function with points in Q having value one and all other points zero, or f_0 takes values on the non-negative integers, points in Q having values greater than some threshold and other points less. However, this assumption causes no loss in generality since it can be extended to a subset Q consisting of points having *any* given property simply by making f_0 the characteristic function for these points.

¹ Since the algorithm requires the values of the neighbours of all points of Q to be defined, P should contain at least all neighbours of points of Q .

Throughout a pass through the points of the picture a record of the initial subset Q must be maintained. However, it has been found convenient, rather than retaining the original picture $f_0 | P$ as well as constructing point by point a new picture $f_1 | P$ which will define Q' , to reserve a third set of values R for points which have been removed ($R \cap I = \phi$ and $R \cap N = \phi$). Only one picture is then required, and as each point $q \in P$ is tested the picture is changed by setting that point to a new value $f_1(q)$, where $f_1(q) \in R$ if the point is removed, or, if it is not, $f_1(q) \in I$ or $f_1(q) \in N$, depending whether the point was in Q in the first place or not.

At the stage when point p of the picture has just been set the value of any other point q is denoted by $f^{(p)}(q)$ (or $f(q)$ for short), where $f^{(p)}(q) = f_0(q)$, if q follows p , and $f^{(p)}(q) = f_1(q)$, if q precedes or is p . At this stage we have a *partially thinned picture* $f^{(p)} | P$; the initial subset Q is given by

$$Q = \{q \in P \mid f^{(p)}(q) \in I \cup R\}$$

and the *partially thinned subset* $Q^{(p)}$ is defined by

$$Q^{(p)} = \{q \in P \mid f^{(p)}(q) \in I\}$$

When all points of the picture have been tested

$$f^{(p)} = f_1 \text{ and } Q^{(p)} = Q' = \{q \in P \mid f_1(q) \in I\}$$

The picture is then ready for the next pass to begin, except that all points with values in R must first be reset to have values in N , or R and N must be redefined to the same effect.

It is sometimes useful, also, to reserve a subset U of I for the values of points which for some reason may not be removed at the current pass.

The crucial part of the algorithm is the determination of whether or not a point should be removed. This is independent of the choice of sets I, N, R and U and is described in full below.

CONDITIONS FOR THE REMOVAL OF A POINT

A point p will be removed, that is, it will be set to a new value $f_1(p)$ in R , if and only if it satisfies all the following conditions:

1. It belongs to Q and its removal is allowed, i.e.,

$$f(p) \in I - U;$$

2. It lies on the edge of Q , that is, at least one of its axially adjacent neighbours n_1, n_3, n_5 and n_7 does not belong to Q ; i.e., if

$$\mu(p) = a_1 + a_3 + a_5 + a_7 \text{ (where } a_i = 1 \text{ if } f(n_i) \in N, a_i = 0 \text{ otherwise)}$$

then we have the edge condition

$$\mu(p) \geq 1;$$

3. It is not the tip of a thin line, that is, it has more than one neighbour which belongs to Q , i.e., if

$$v(p) = \sum_{i=1}^8 (1 - a_i) \quad (a_i \text{ defined as above})$$

then we have the 'not tip' condition

$$v(p) \geq 2$$

4. It is not the last remaining point of small 'circular' subset, that is, it has at least one neighbour in Q which has not been removed; i.e., if

$$w(p) = \sum_{i=1}^8 c_i \quad (\text{where } c_i = 1 \text{ if } f(n_i) \in I, c_i = 0 \text{ otherwise})$$

then we have the condition

$$w(p) \geq 1$$

5. Its removal does not alter connectivity, that is, the area of the Euclidean plane defined by the subset Q can be continuously deformed to give the area defined by the subset $Q - \{p\}$. This will be so if and only if the set comprising those neighbours of p which are in Q consists of one connected component. (The case where the removal of p would produce a 'hole' is prevented by condition 2 above).

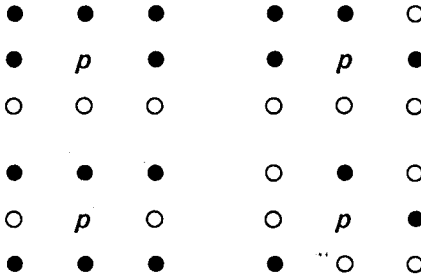


Figure 6. Crossing number. The crossing number at point p with respect to the set of black points is one in the two upper cases and two in the two lower cases. Note in the two right-hand cases that neighbour one is connected to neighbour three, so the fact that neighbour two is white does not cause an increase in crossing number.

The number of components is calculated by considering the number of times a 'bug' taking a walk around p by way of its neighbours would have to cross from outside to inside the subset. This number is called the *crossing number* of p with respect to the subset, and is generally equal to the number of components, but would be zero in the case where the 'bug' did not need to leave the subset at all. Note that to conform with our definition of connectivity the 'bug' must go from one axially adjacent neighbour of p to the next axially adjacent one, by-passing the diagonal if by so doing it avoids leaving the subset. Some examples are given in figure 6.

PATTERN RECOGNITION

This crossing number $X(p)$ is calculated as follows:

$$X(p) = \sum_{i=1}^4 b_i \text{ (where } b_i = 1 \text{ if } f(n_{2i-1}) \in N \text{ and either } f(n_{2i}) \in I \cup R \text{ or } f(n_{2i+1}) \in I \cup R \text{ and } b_i = 0 \text{ otherwise).}$$

The connectivity condition is then given by

$$X(p) = 1$$

6. Its removal in conjunction with any one of its neighbours that has been removed does not alter the connectivity of Q ; that is, if neighbour n_i of p has been removed, i.e., $f(n_i) \in R$, then if the value of this neighbour is temporarily altered so that $f(n_i) \in N$ and it thus appears that n_i is not in Q , then the new crossing number at p , $X_i(p)$, say, is still one. This gives us the additional 'two thick line' condition

$$f(n_i) \notin R \text{ or } X_i(p) = 1 \text{ (} i = 1, \dots, 8 \text{)}$$

In general, the sequence in which the points are being tested is known, and it is therefore only necessary to test this last condition for those neighbours which precede p , since f can only take a value in R for a point *after* it has been tested. For example, if we start at the top of the picture and work from left to right along each line, it is only necessary to test the condition for $i=2,3,4$ and 5. In addition, if i is even and p satisfies all the preceding conditions, then the condition that either $f(n_i) \notin R$ or $X_i(p) = 1$ is automatically satisfied and need not be tested. For, if $f(n_i) \in R$, then, since p has crossing number equal to one and has more than one neighbour in Q , and n_i belongs to Q , it follows that either n_{i-1} belongs to Q , or n_{i+1} belongs to Q , or both. If both belong to Q then deleting neighbour i does not alter the crossing number at p since the 'bug' can in this case 'cut the corner'. If on the other hand only one of these is in Q then deleting neighbour i cannot possibly alter the crossing number, so in each case we have $X_i(p) = 1$. It is thus only necessary to test this last condition for $i=3$ and $i=5$, i.e.,

$$f(n_i) \notin R \text{ or } X_i(p) = 1, \text{ for } i=3 \text{ and } i=5$$

APPLICATION TO BINARY PICTURES

The algorithm was initially implemented for use on pictures taking the form of a characteristic function. In this case points belonging to the subset Q have value one, all other points have value zero, and when a point is removed its value is set to minus one, i.e., the algorithm is applied with

$$\begin{aligned} I &= \{1\} \\ N &= \{0\} \\ \text{and } R &= \{-1\} \end{aligned}$$

Between each pass it is necessary to set those points which have been removed and so have value -1 to zero ready for the next pass.

This can be made more efficient in the following way. Consider a point p of the picture and suppose that at one pass neither this point nor any of its neighbours is removed. Then this point cannot possibly be removed at the next pass since its situation with regard to the conditions for removal has in no way changed. If all such points are set to two, therefore, and we define

$$U = \{2\}$$

this will avoid the unnecessary testing of the neighbours, crossing number, etc., of these points at the next pass. This is achieved by setting the value of *all* points of Q which are not removed to two; then at the end of the pass, when all points which have been removed are reset from -1 to zero, their neighbours with value two are reset to one.

Figure 7 shows the results obtained by applying this algorithm to several pictures.

An alternative method of improving the efficiency of the algorithm is by avoiding the need to reset the values of all points which have been removed, between passes, by re-defining the sets N and R instead. This is done by setting the value of those points which are removed at the m th pass to $-m$ instead of -1 and defining I , N and R as follows:

$$\begin{aligned} I &= \{1\} \\ N &= \{f \mid 0 \geq f > -m\} \\ R &= \{-m\} \end{aligned}$$

Thus, after each pass, N and R are re-defined by incrementing m , and the next pass can take place immediately.

SHORTCOMINGS OF THESE IMPLEMENTATIONS

With chromosome images we found that the type of approach described above had two disadvantages. First, the outlines of chromosomes tend to be rather noisy, with the adjacent arms of a chromosome not completely separate but touching in several places, and this results in a skeleton with many spurious branches, as illustrated in figure 8b. This can usually be overcome without much difficulty by a simple smoothing of the boundary and by expanding any 'holes' produced by touching arms until they are connected with the exterior of the object. Application of the algorithm then usually produces a skeleton with few or no spurious branches, as in figure 8c.

A second more serious shortcoming is that the resulting skeleton is completely defined by the original outline of the object. If the picture is essentially black and white and its outline well defined – as it frequently is in character recognition – this is exactly what is required. However, with chromosomes the density drops off gradually from the central ridge of each arm outwards, and the choice of boundary is somewhat arbitrary. In this case what is

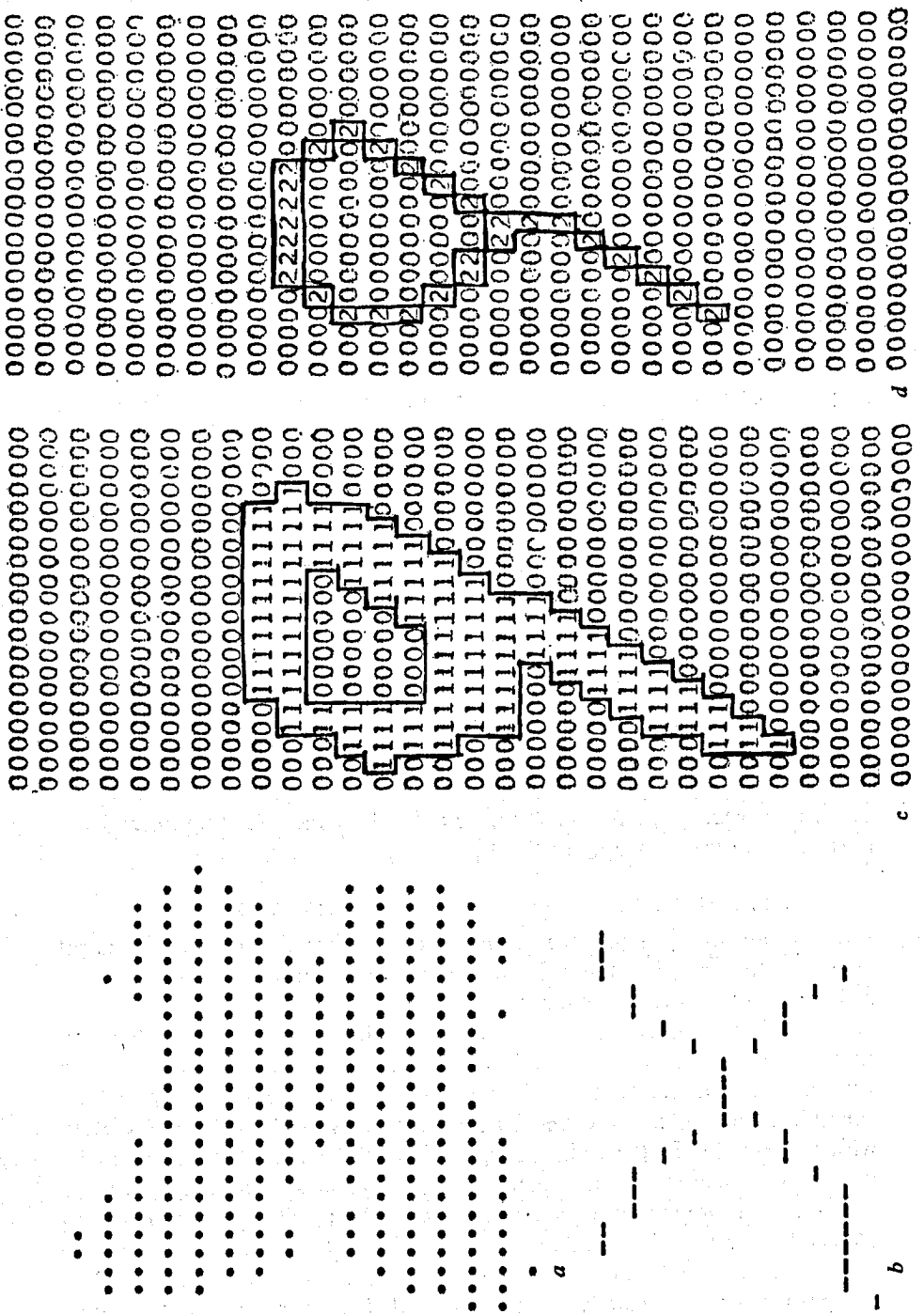


Figure 7. Binary pictures of a chromosome and of a hand-written digit, and the skeletons obtained from them.

required is that the skeleton should lie along the higher density ridges of the picture rather than in a central position determined by outline alone. An example illustrating this is shown in figure 9.

The way in which the algorithm can be varied in order to achieve this, for pictures in which f takes values that are positive integers representing the densities of the points, is given below.

IMPLEMENTATION FOR PICTURES WITH A RANGE OF DENSITY VALUES

It is possible to obtain a skeleton which tends to lie along ridges of higher density in a picture by removing only points at low density in the first few passes and allowing points with higher density to be removed later. This is done by restricting removal in any one pass to points with a given value. Thus, if d is the value of points which may be removed in the current pass, the algorithm is applied with I, N, R and U defined as follows:

$$\begin{aligned} I &= \{ \text{positive integers} \} \\ N &= \{ 0 \} \\ U &= \{ \text{all positive integers except } d \} \\ R &= \{ -1 \} \end{aligned}$$

Initially d is set to one. After each pass those points which have been removed are set to zero and the algorithm is re-applied until a pass is made in which no points are removed. At this stage any troughs of points with value one, which are not completely surrounded by points with a higher value, will have been deleted. d is now increased to two and the process repeated. In this way points with a low value lying between ridges of higher value points are deleted first, thus, for example, separating touching arms. When no further points with value two can be removed the program starts on value three, and so on. However, whenever the deletion of a point with value d uncovers a point with a lower value (i.e., a neighbour of a deleted point has value $g, 0 < g < d$) d is reset to g at the next pass. Nevertheless, it may be desirable to consider that whenever there is an area of points with value d_1 completely surrounded by points with a higher value, d_2 say, then this represents a closed curve surrounding the d_1 points. To achieve a skeleton which reflects this, all that is necessary is that when all points with value less than or equal to d_2 which can be removed, have been, then any remaining points at level d_1 are deleted thus producing 'holes'. Thinning then continues at level $d_1 + 1$.

Unfortunately, application of the algorithm is very time-expensive since it requires a complete pass to be made through *all* the points of the picture even though there may be only very few points remaining with the current value d (and in general the greater the number of values the less the efficiency). This has been overcome by initially making a list of all points of the picture with each value. Therefore, when removing points with a given value, it is necessary to consider only the points given in the list for that value.

	292	302	312	322	332	342	352	362
	I	I	I	I	I	I	I	I
96								
97								
98								
99								
100								
101								
102								
103								
104								
105								
106								
107								
108								
109								
110								
111								
112								
113								
114								
115								
116								
117								
118								
119								
120								
121								
122								
123								
124								
125								
126								
127								
128								
129								
130								
131								
132								
133								
134								
135								
136								

	291	301	311	321	331	341	351
--	-----	-----	-----	-----	-----	-----	-----

95							
96							
97							
98							
99							
100							
101							
102							
103							
104							
105							
106							
107							
108							
109							
110							
111							
112							
113							
114							
115							
116							
117							
118							
119							
120							
121							
122							
123							
124							
125							
126							
127							
128							
129							
130							
131							
132							
133							
134							
135							
136							
137							

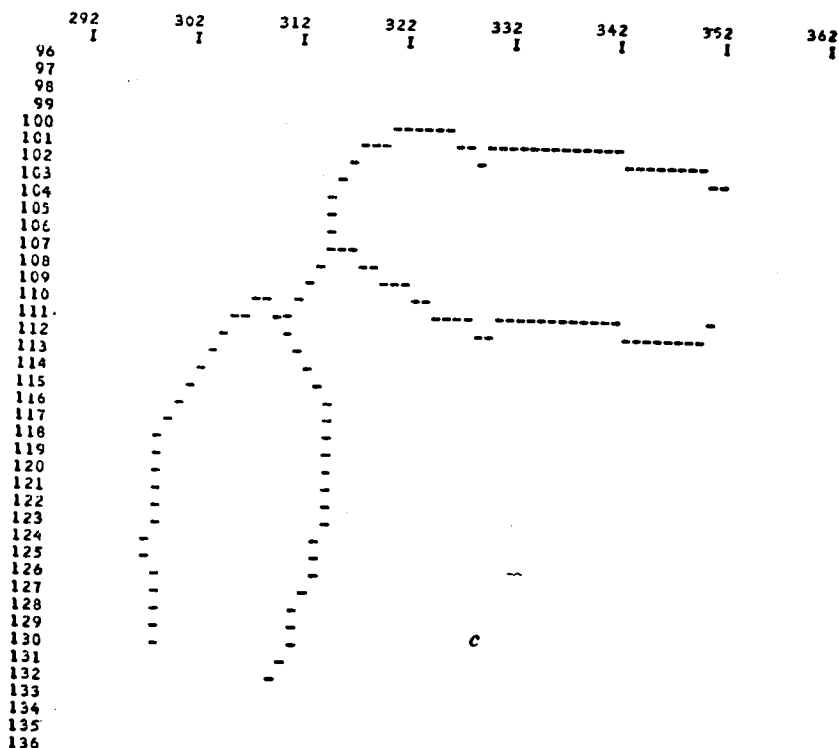


Figure 8. (a) A digitization of a chromosome. The chromosome is digitized on a seven-level density scale, and in the line printer representation shown here the characters are chosen so that the density at each point corresponds approximately to the overall darkness of the printed character.
 (b) The skeleton obtained from this chromosome by using the binary application of the algorithm on its characteristic function.
 (c) The skeleton obtained by pre-processing the picture before taking the characteristic function.

PATTERN RECOGNITION

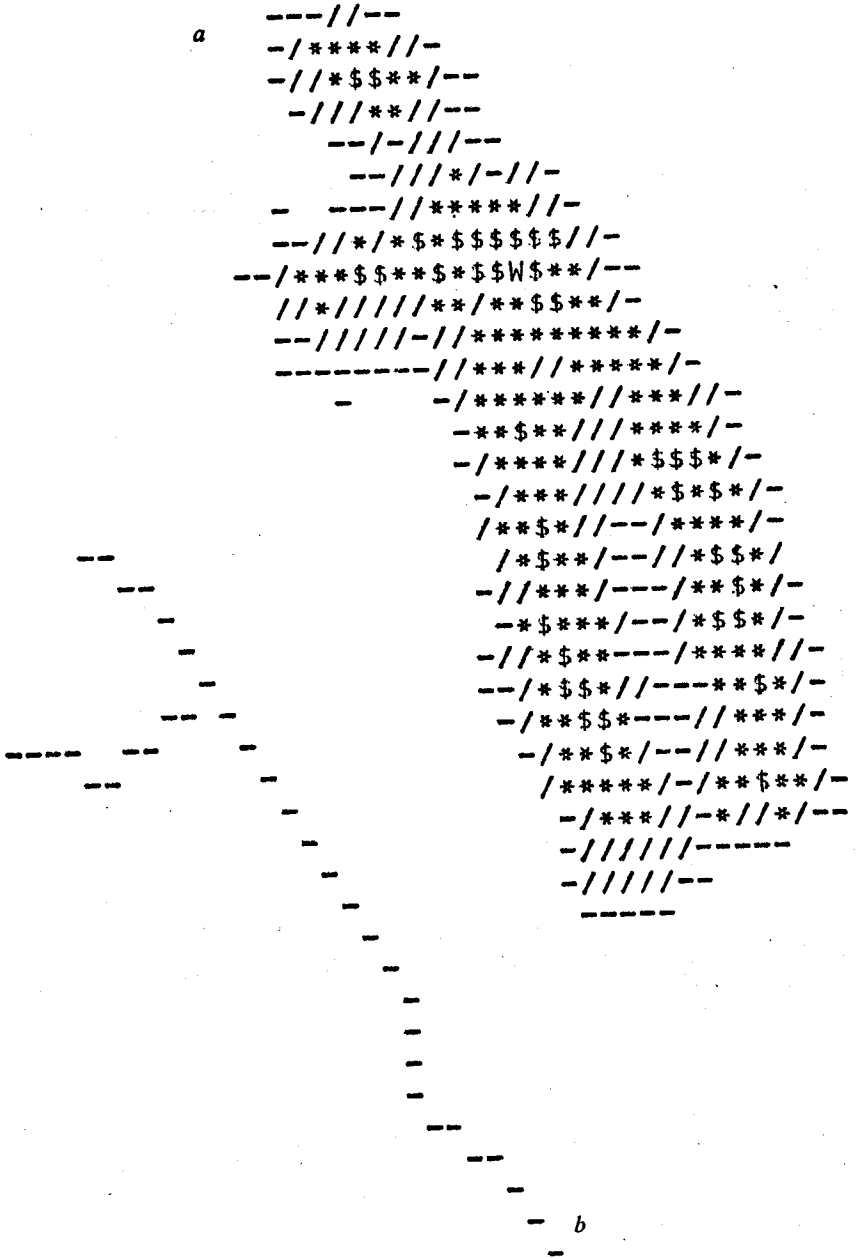


Figure 9. An example of a chromosome and its skeleton obtained by using the binary application of the algorithm. In this case it would be better if the skeleton were to lie along the high density ridges of the picture, rather than in a central position with respect to area alone.

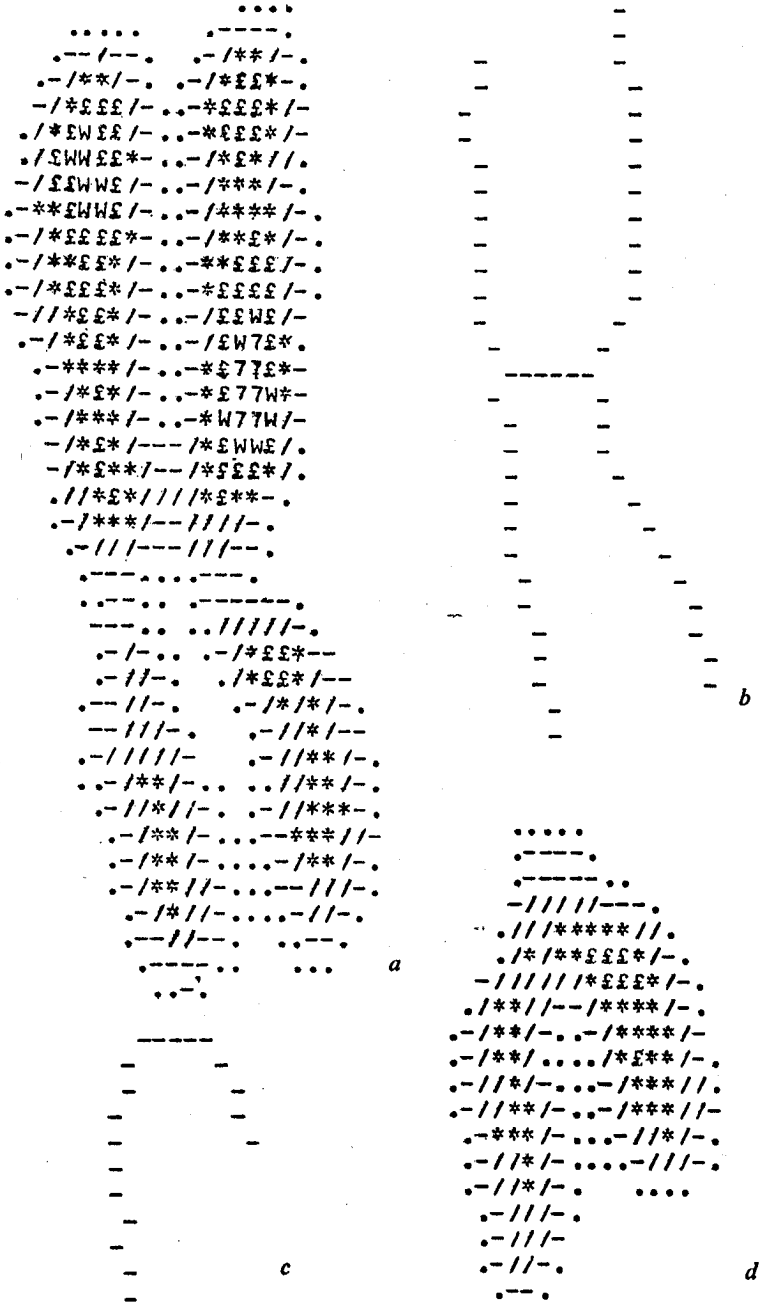


Figure 10. Some chromosome pictures and skeletons obtained using the version of the algorithm intended for pictures covering a range of density values.

PATTERN RECOGNITION

The success of this algorithm is illustrated in figure 10. As well as being less sensitive to the choice of boundary for the picture it is also found to be much less sensitive to noise – the only pre-processing required to obtain satisfactory skeletons being a simple smoothing, by setting each point to a weighted average of itself and its neighbours.

Acknowledgements

Thanks are due to all members of the Clinical and Population Cytogenetics Research Unit, Pattern Recognition and Automation Section under the direction of Dr Denis Rutovitz.

The computing was done using the IBM 7090 computer at Imperial College in conjunction with a Fidac Scanner.

REFERENCES

- Blum, H. (1964) *A Transformation for Extracting New Descriptors of Shape*. Boston: Air Force Cambridge Research Laboratories.
- Deutsch, E.S. (1967) Computer simulation of a character recognition machine. *The Post Office Electrical Engineering Journal*, 60, 39–44 and 104–9.
- Izzo, N. & Coles, W. (1962) Blood-cell scanner identifies rare cells. *Electronics* 52.
- Kirsch, R.A., Cahn, L., Ray, C. & Urban, G.J. (1957) Experiments in processing pictorial information with a digital computer. *Proc. Eastern Computer Conference*, 221–9.
- McCormick, B.H. (1963) The Illinois pattern recognition computer—Illiac III. *IRE Trans. Electronic Comp.* EC-12, 5.
- Narasimhan, R. (1964) Labelling schemata and syntactic descriptions of pictures. *Information and Control*, 7, 151.
- Philbrick, O. (1966) *A Study of Shape Recognition Using the Medial Axis Transform*. Boston: Air Force Cambridge Research Laboratories.
- Preston, K. (1961) The Cell Scan system—a leukocyte pattern analyser. *Proc. Western Computer Conference*, 173.
- Rosenfeld, A. & Pfaltz, J.L. (1966) Sequential operations in digital picture processing. *Journal of the ACM*, 13, 471–94.

**PROBLEM - ORIENTED
LANGUAGES**



Absys 1: an Incremental Compiler for Assertions; an Introduction

J.M. Foster

and

E.W. Elcock

Computer Research Group
University of Aberdeen

1. INTRODUCTION

Earlier papers (Elcock 1968, and Foster 1968) in this Workshop series introduced design aims and motivation for an assertional language. Briefly, in many problems concerned with complex but well-structured data, it is advantageous and certainly nearer to mathematical practice to be able to assert things *about* the structure of data, instead of being constrained to sequences of imperative statements to *construct* particular data.

Absys 1 (Absys standing for *Aberdeen System*) is a working on-line incremental compiler written by the Computer Research Group at Aberdeen for the Elliott 4120.

The aim of this paper is to present, in an informal way, some of the main features of the implemented language, and to try to exhibit assertional programming by means of a few examples. The paper is not meant to be a complete description of the language, which will appear elsewhere.

2. THE 'AND' CONNECTIVE

An individual assertion asserts a relation about data objects. Thus the assertion

$$x=y$$

asserts that x and y satisfy an equality relation.

The operator '=' is to be interpreted in the sense 'substitutable for', and as such has the expected properties of reflexivity, transitivity, etc. In the same spirit, the arithmetic operators have their expected properties so that, for example, the assertions $a=b+1$, $a=1+b$, $1+b=a$, $b+1=a$ are equivalent.

PROBLEM-ORIENTED LANGUAGES

A written program consists of assertions. The individual assertions of a program have an implicit (in that it is not written) 'and' connective between them, with properties similar to its logical counterpart.

The system acts to construct data satisfying the conjunction of the assertions. Thus, the trivial program

$$x=y \quad y=2$$

would make both x and y the datum 2.

If the conjunction of the assertions of a program is found to be unsatisfiable then the program terminates unsuccessfully.

The occurrence in a program of the assertions $y=x$, $x=2$ and $3=y$ would make that program terminate unsuccessfully, since no data x , y can be constructed to satisfy what has been asserted about x , y , 2 and 3.

There are no type declarations in the language. Types are determined progressively and dynamically. Thus, after the assertion,

$$a=b+c$$

the types of a , b and c are constrained only to the set of types meaningful with the asserted '+' and '=' relations.

3. DATA DIRECTED CONTROL

A written assertional program places no explicit constraints on the order in which particular operations are performed. In particular, if data can be constructed to satisfy all the assertions about them, then these data are independent of the order in which the operations are performed.

The on-line system is incremental in that, as assertions are accepted by the system, whatever processing can be done on the basis of data already present in the system is done.

This lack of unnecessary concern with control in assertional programs is sufficiently novel to be worth elaborating in the context of a trivial example of list processing.

Consider the following constructions in a conventional list processing language with assignment:

(i) the constructor operation

$$z \leftarrow \text{cons}(x,y)$$

which the programmer must ensure is only processed in a data environment in which x and y have appropriate values and the current value of z is no longer needed.

(ii) the selector operation

$$x \leftarrow \text{hd}(z)$$

which the programmer must ensure is only processed in a data environment in which z has an appropriate value and the current value of x is no longer needed.

(iii) a test such as

$$\text{equal}(x, \text{hd}(z))$$

which the programmer must ensure is only processed in a data environment in which both x and z have values.

In Absys 1

$$z = [x \ \& \ y]$$

simply asserts that z is a list whose head is x and whose tail is y .

Whether it acts to construct z , or to select x and y , or to test whether x , y and z satisfy the asserted relation, depends solely on the data in the system at the time this assertion is processed.

If, at the time of processing the assertion, z is a known datum and it is a list, then the assertion acts as a selector in that x will be asserted to be equal to the head of the list and y will be asserted to be equal to its tail. If, on the other hand, at the time of processing the assertion z is an unknown datum, then the effect is to make z a list and again assert that x is equal to its head and y is equal to its tail, that is, to act as a constructor.

In a less trivial example: the rather opaque assignment statement

$$z2 \leftarrow \text{cons}(\text{cons}(\text{hd}(z1), \text{cons}(\text{hd}(\text{tl}(z1)), \phi)), \text{cons}(\text{cons}(\text{hd}(z1), \text{cons}(\text{hd}(\text{tl}(z1))), \phi), \phi))$$

expresses only one facet of the assertions

$$z1 = [a; [b; c]]$$

$$z2 = [[a; b]; [a; c]]$$

which asserts a simple relationship over the lists $z1$ and $z2$.

4. FUNCTIONS

A lambda construction allows the assertion of functions other than primitive functions of the system.

Lambda expressions are sufficiently well known for the features of the Absys 1 implementation to be discussed only briefly and by example.

Thus the assertion

$$f' = \text{lambda } x, y \Rightarrow z \text{ begin } x = [z \ \& \ q'] \ y = [z \ \& \ r'] \ \text{end}$$

introduces a new function f such that

$$f(m, n) = b$$

is equivalent to the compound assertion

$$\text{begin } m = [b \ \& \ q'] \ n = [b \ \& \ r'] \ \text{end}$$

that is, an assertion that m and n are lists with the same first item b .

The primes in the above serve to introduce new names, the textual scope of which are delimited by `begin` and `end` brackets.

PROBLEM-ORIENTED LANGUAGES

Functions may be introduced by partial application of other functions. Thus:

$$g' = f'(1)$$

$$f = \text{lambda } x, y \Rightarrow z \text{ begin } z = x + y \text{ end}$$

asserts that g is the function $\text{lambda } y \Rightarrow z \text{ begin } z = 1 + y \text{ end}$

5. THE 'OR' CONNECTIVE

Before going on to discuss the use of lambda constructions for the assertion of recursive functions it is necessary to introduce the assertion of alternatives. Alternatives can be asserted by the construction

$$\{A1 \text{ or } A2\}$$

where $A1$ and $A2$ are conjunctions of assertions.

The assertional (implicit) **and** and **or** distribute so that, e.g.,

$$A1 \{A2 \text{ or } A3\} A4$$

is equivalent to

$$\{A1 A2 A4 \text{ or } A1 A3 A4\}$$

The system attempts to construct distinct data to satisfy each conjunction of assertions. Unsatisfiable conjunctions disappear when unsatisfiability is detected.

To make this clear with a completely artificial example: after

$$p = [1; 2]$$

$$\{p = [a; b] \text{ or } p = [b; a]\}$$

there are two sets of distinct data. In one set a and b are the data 1 and 2 respectively whilst in the other set a and b are the data 2 and 1 respectively.

If we were now to assert, for example,

$$a = b + 1$$

one of the conjunctions will be unsatisfiable and will disappear from the system together with its associated data.

By distribution, an assertional program can be transformed into a normal form of a disjunction of conjunctions of elementary assertions. In this form the conjunctions in effect constitute parallel non-interacting programs.

Absys 1 distributes the **and**, **or** connectives in a way which attempts to minimise unnecessary duplication of processing.

6. RECURSIVE FUNCTIONS AND KEYS

Let us use the **or** construction to introduce a function analogous to the recursive list-processing function *map*. The intention is that $\text{map}(p, f)$, where p is a list and f a function, should have the effect of asserting f applied to each item of the list.

Consider the assertion

$$\begin{aligned} \text{map}' &= \text{lambda } p, f \\ &\quad \text{begin } \{ \text{null}(p) \text{ or } p = [r' \ \& \ s'] f(r) \text{ map}(s, f) \} \text{ end} \end{aligned}$$

The two alternatives at each call of *map* are incompatible in that either the list is null or not and if this incompatibility is detected then only one, the intended, conjunction of assertions will be satisfiable. However, since there is no explicit ordering of processing it is possible that processing of *map* might give rise to further processing of *map* in the second of the or alternatives before processing the assertion $p = [r' \ \& \ s']$ in that alternative. In this case the construction leads to processing which does not terminate.

It is clearly not sensible to try to process $\text{map}(p, f)$ unless the datum p is already present. Similar considerations apply to many other functions. The lambda construction above does not however express this information.

The correct assertion for *map* is:

$$\begin{aligned} \text{map}' &= \text{lambda } p, f \text{ key } p \\ &\quad \text{begin } \{ \text{null}(p) \text{ or } p = [r' \ \& \ s'] f(r) \text{ map}(s, f) \} \text{ end} \end{aligned}$$

The key statement indicates the data that must be present before evaluation of the function may take place.

With this construction it is clear that there is no difficulty about termination, since the $\text{map}(s, f)$ in the second alternative of the or will not be evaluated until the datum s is present. This datum is constructed only as a result of processing the assertion $p = [r' \ \& \ s']$, which is the assertion incompatible with the assertion $\text{null}(p)$ in the first alternative of the or.

In the example of *map* above the or acts like a conditional. The following example makes fuller use of the possibilities of the or construction.

First, a preliminary assertion:

$$\begin{aligned} \text{item}' &= \text{lambda } x \Rightarrow z \text{ key } x \\ &\quad \text{begin } x = [p' \ \& \ q'] \{ z = p \text{ or } z = \text{item}(q) \} \text{ end} \end{aligned}$$

This function *item* is such that, for example, the assertion

$$i = \text{item}([1;2;3])$$

is equivalent to asserting

$$\{ i = 1 \text{ or } i = 2 \text{ or } i = 3 \}$$

Consider now the following problem. We are given three lists p_1 , p_2 and p_3 , and we wish to assert that the triples r , s , t , such that r is an *item* in p_1 , s in an *item* in p_2 and t is an *item* in p_3 , satisfy the relation $r + s = t$. The required assertion is simply

$$r = \text{item}(p_1) \quad s = \text{item}(p_2) \quad t = \text{item}(p_3) \quad r + s = t$$

For example, if the data p_1 , p_2 , p_3 already exist and p_1 , p_2 , p_3 are respectively the lists [1;2;5;6], [3;7;1], [10;2;7;4], then the result would be three

distinct conjunctions of assertions with associated data r, s, t 1, 3, 4, 1, 1, 2 and 6, 1, 7 respectively. Each of these conjunctions would of course be subject to any further assertions made.

7. A FINAL EXAMPLE

The function $f' = \text{lambda } x \text{ begin } g(x) \text{ end}$ parallels in an assertional language what is rendered in predicate calculus by $\forall(x)\{f(x) \rightarrow g(x)\}$, in that if, for any a , $f(a)$ is asserted, then $g(a)$ is also asserted.

It is interesting to examine how one might parallel more general if-then statements such as the statement $\forall(x, y)\{f(x) \wedge g(y) \rightarrow h(x, y)\}$. We want to arrange that if we assert, for example,

$$f(a) \quad f(b) \quad g(c) \quad f(d)$$

then $h(a, c)$, $h(b, c)$ and $h(d, c)$ are also asserted.

To do this we might arrange that the function f is defined so that an assertion $f(a)$ extends the function g by the partially applied function $h(a)$, so that an assertion $g(b)$ results in the further application of $h(a)$ to b to produce $h(a, b)$. The extension of the function g generated by any particular assertion of f must of course operate on all asserted arguments of g , irrespective of the particular temporal sequence of processing asserted f s and g s.

Call a list of the form $[a; b \ \& \ q]$, where q is an as yet undetermined datum, an extendable list. The significance of this name is obvious at the non-functional level in that after, e.g.,

$$p' = [1 \ \& \ q']$$

the assertion

$$q = [2 \ \& \ r']$$

extends p to be $[1; 2 \ \& \ r]$, when the assertion

$$r = [3 \ \& \ s']$$

extends p again to be $[1; 2; 3 \ \& \ s]$, etc.

Let *extension* be a system primitive function such that the assertion

$$\text{extension}(i, l)$$

extends an extendable list l by the item i leaving l still further extendable.

Assert

$$\text{applist}' = \text{lambda } l, x \text{ key } l$$

$$\text{begin } l = [r' \ \& \ s'] \ r(x) \ \text{applist}(s, x) \ \text{end}$$

Applist takes as first argument an extendable list of functions and asserts the conjunction of these functions applied to the other argument.

Returning to $f(x) \wedge g(y) \rightarrow h(x, y)$, the required effect is now obtained by asserting:

new *glist*

$f' = \text{lambda } x \text{ begin extension}(h(x), \textit{glist}) \text{ end}$

$g' = \text{lambda } y \text{ begin applist}(\textit{glist}, y) \text{ end}$

It is revealing to follow through the effect of some assertions of f and g with particular arguments, as if these assertions were made at the on-line teletype.

$g(b)$

The *glist* is as yet undetermined and so *applist(glist,b)* is held up awaiting extension of the *glist*.

$f(a)$

The *glist* is now extended by the partially applied function $h(a)$ and becomes [$h(a)$ & s'], say. The waiting *applist(glist,b)* now results in $h(a)$ being applied to b , asserting $h(a,b)$, leaving *applist(s,b)* awaiting further extension of the *glist*.

$g(c)$

This asserts *applist(glist,c)* where *glist*=[$h(a)$ & s] and gives rise to the assertion $h(a,c)$, leaving *applist(s,c)* awaiting further extension of the *glist*.

$f(d)$

The *glist* is extended further by $h(d)$ to become [$h(a)$; $h(d)$ & r'] and the waiting *applists* now assert $h(d,b)$ and $h(d,c)$, etc., etc.

Acknowledgements

The work reported in this paper is sponsored by the Science Research Council. The authors wish to thank Dr A. M. Murray for help in implementing the system.

REFERENCES

- Elcock, E.W. (1968) Descriptions. *Machine Intelligence 3*, pp. 173-80 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
- Foster, J.M. (1968) Assertions: programs written without specifying unnecessary order. *Machine Intelligence 3*, pp. 387-92 (ed. Michie, D.). Edinburgh: Edinburgh University Press.



**PRINCIPLES
FOR DESIGNING
INTELLIGENT
ROBOTS**

THE
LIBRARY
OF THE
MUSEUM OF
ART AND HISTORY
OF THE
CITY OF
NEW YORK

Planning and Generalisation in an Automaton/Environment System

J. E. Doran

Department of Machine Intelligence and Perception
University of Edinburgh

INTRODUCTION

This paper describes recent progress with a computer program which simulates a general heuristic controller learning to perform a rather special control task. An early version of the program was described in Doran (1968).

The general control situation is represented in figure 1. There is a 'black box' which we may imagine to have on it only a set of 'windows' and a set of 'buttons'. Through each window can be seen a symbol from some arbitrary set. The black box contains unknown mechanisms which in some way relate the symbols displayed to the pressing of buttons. We suppose that the box operates in real time and that buttons can be pressed without restriction.

The controller has, roughly, the task of keeping certain symbols visible through certain of the windows for as much of the time as possible. Necessarily it has built into it some general assumptions about the nature of the box, but it is expected to collect and remember detailed information about the way the box reacts to button-pressing, and to use this information to improve its performance in the control task. In general there will be restrictions on the components and logical processes which can be built into the controller.

Similar control tasks have been considered by Andreae (1968) and Michie and Chambers (1968).

It is illuminating to consider the set B of all possible black boxes with associated control tasks, and the set A of all possible controllers, and to consider the relationship between these two sets. In particular, given a subset B' of B together with a probability distribution over this subset, we can suppose that there will be an optimal controller $a(B')$ in A , given that the task is to control some box b from the set B' , where b is to be selected randomly according to the probability distribution and therefore cannot be predicted in advance. In an important sense, B' defines a . The most interesting members

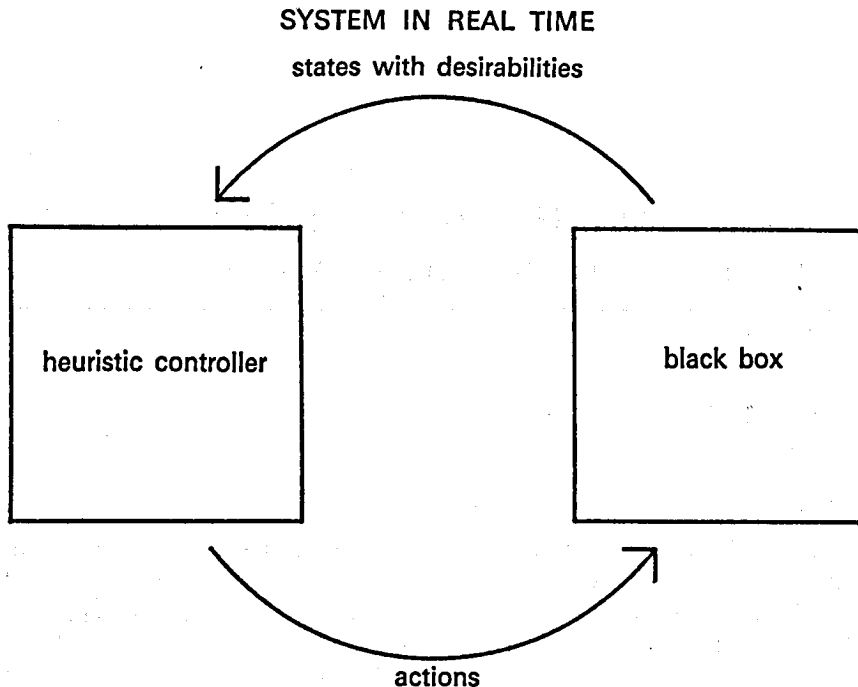


Figure 1. Schematic representation of the control task discussed in the text

of the set A are those controllers which are optimal for large and varied subsets of B .

Let us now call each window of the black box a *variable*, each symbol a *value*, and the set of variable values on display at any time the *state*. Let us also call the pressing of a button an *action*. The controller to be described in this paper, henceforth called the *automaton*, is most efficiently applied to the subset of B of which the following are the main characterising properties:

1. The state normally only changes after an action, but occasionally spontaneous changes occur.
2. The effect of an action depends upon the state at the time of the action, and possibly on a random variable, but not on the previous states or actions.
3. The control task is defined in terms of the desirability of each possible value of each variable.
4. The control task can be performed by dealing with each variable independently, possibly in some particular order.
5. Nothing especially simple can be said about the effects of particular sequences of actions.

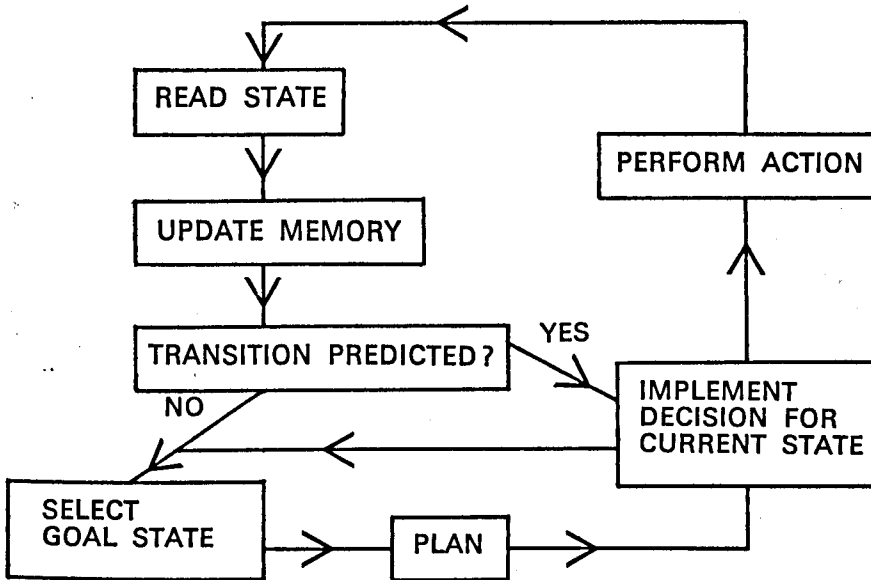


Figure 2. Outline flow chart for the heuristic controller (the *automaton*)

Notice that, unlike normal function maximisation or 'hill-climbing', this situation is one where it is sensible to store information from one search for control to the next, and where the desirability of states is in no sense a continuous function.

When the automaton is applied to a control task not possessing some of these properties, then we must expect its performance to be correspondingly poor, though not necessarily disastrous. In fact, the special, but particularly interesting, control task defined in this paper contradicts properties 2, 5 and, to some extent, 4. It is a greatly simplified representation of the task which the brain of a small animal (or mobile robot) faces when living in a spatial environment which has to be explored and understood if the necessities of life are to be obtained. Comparatively poor performance is acceptable since we learn about the ways in which a general controller can cope with such a spatial environment.

It is worth emphasising that this work is not intended as a model of animal learning or other behaviour, nor as a serious robot simulation. It does, however, have some relevance both to animals and to robots. It is intended as an investigation of how different heuristic processes can be integrated into an effective general controller.

THE AUTOMATON DESIGN

Figure 2 is an outline flow chart for the heuristic controller, that is, for the automaton. The main elements of the automaton design are the following:

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

1. A process for selecting top level goals.
2. A planning process which predicts the effects of sequences of actions and which decides which action should be used in the various states predicted to occur. This planning process has many features in common with the tree-searching components of typical problem-solving and board-game playing programs.
3. A memory network of complex nodes each of which contains the descriptions of a state together with useful information.
4. A generalization process which is used in planning.

After explaining these processes in detail I shall describe the automaton's performance in the spatial environment mentioned in the last section. I shall then end the paper with a discussion of recent improvements in the generalisation and planning processes. I shall describe the design of the automaton independently of particular control tasks.

Goal selection

Before describing the goal selection process I shall make the definition of the control task a little more precise. There is associated with each state variable a function, which when applied to the variable value yields a measure of the *desirability* of the value. The measure always lies between 0 and 1. The *total desirability* of a state is defined as $\sum wd$, where the summation is over all the variables, and where d is the value desirability and w a weight, also between 0 and 1, which defines the *importance* of the variable. The task is then to maximise the mean total desirability over whatever period of time is specified.

In this simulation 'time' is simply a measure of the amount of computation carried out by the automaton. The automaton does not perceive time directly.

When the total desirability exceeds a chosen threshold, the automaton ceases selecting actions until the desirability falls again. Notice that unlike the desirability functions and the importance weights, which are logically part of the control task, this threshold can be regarded as an adjustable part of the automaton's control strategy.

Throughout the planning and decision-making processes to be described, the automaton has a particular state, its *goal*, which it is trying to reach. More precisely, it is trying to increase the desirability of one particular variable, and to this end is trying to reach again a state encountered in the past at which this variable had a high desirability. The automaton reconsiders its goal whenever it is about to plan, and always assumes that variables can be controlled independently. It will never try to control two variables jointly.

In more detail, the situation is as follows. For each variable the automaton keeps a *potential goal state*. This is the state in which the variable had its highest observed desirability score. When selecting a goal, the automaton calculates for each variable the difference between the desirability of the

variable value in the potential goal state and the desirability of the variable value in the current state. The result multiplied by the importance weight, I shall call the *benefit*. The automaton also calculates the *match* between the current state and the potential goal state. This is the number of variables having the same value in each state divided by the total number of variables. The final merit figure for a potential goal state is then calculated as

$$B(Mk+1-k),$$

where B is the benefit, M the match, and k an adjustable parameter. The potential goal state with the highest merit score is adopted as the goal.

Notice that, by incorporating an admittedly crude estimate of difficulty of achievement into the selection process, the automaton is caused to abandon trying to bring one variable under control not only if another which is more important goes out of control, but also if another which is less important seems easier to bring under control.

The potential goal state for a particular variable is not fixed. It will change whenever the automaton discovers, or is shown, a better value for the variable.

Decisions, plans, and plan implementation —

In view of its initial total ignorance, the automaton's general strategy must be to try actions, to observe the consequences, and to keep the acquired information available for future reference.

The assumption that the state changes only in direct and immediate response to an action means that the automaton need only inspect the state just after it has implemented an action, that is, after a *transition*. Its past experience can therefore be represented as a state/action sequence as in figure 3(i). Given the additional assumption that the effect of an action depends only upon the current state, then the past experience can be represented without loss as a network as in figure 3(ii), with different occurrences of the same state no longer distinguished.

Let us now consider the concept of a *decision*. Roughly, this is a state together with an *operation* which will be implemented whenever it is encountered, given a particular goal state. The operation will usually be an action but may be more complicated. Other components of a decision are introduced below.

Ideally the automaton must, for each goal state, form a correct decision for all the states which it may encounter. How and when should decisions be made? In general, they can be made just when they are needed, that is, when some action has to be chosen for immediate implementation; or they can be made in anticipation of a later need.

The automaton forms decisions in advance of need during *planning phases*. During a planning phase the automaton tries to predict the effect of implementing alternative sequences of actions, so that it can select the most

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

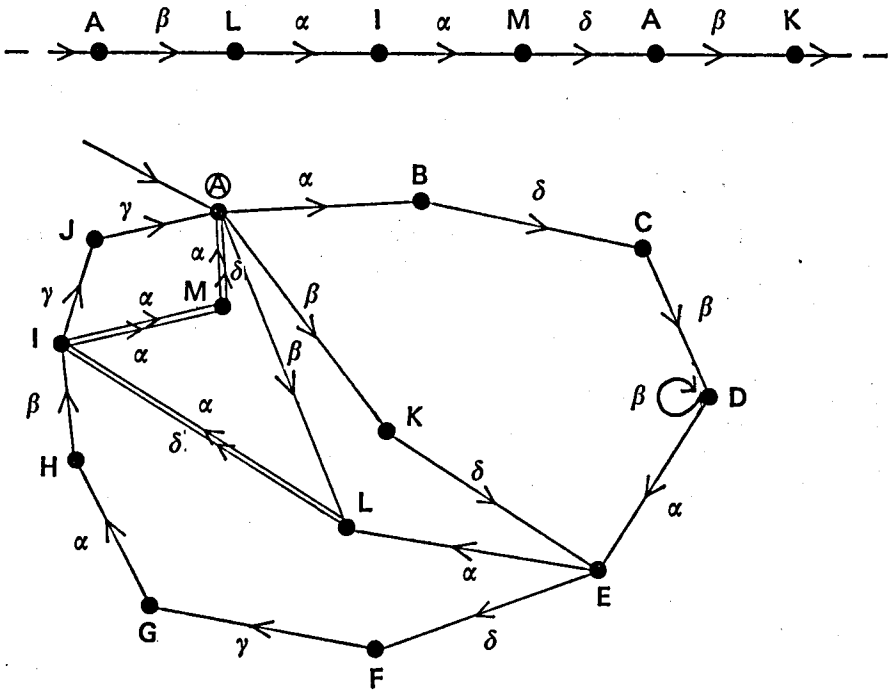


Figure 3. (i) The automaton's observations represented as a sequence of states (capital letters) each followed by an action (Greek letter). (ii) The automaton's observations represented as a network. Each node corresponds to a state and each arc to a transition caused by an action. The network representation does not permit the original state/action sequence to be reconstructed

promising actions with respect to its current goal state. Therefore it must be able to predict what will be the result of implementing a given action, α say, in a given state, A say. In the terminology of Michie and Popplestone (see Michie 1968) it needs *predictor functions*. Let us assume at this stage that it makes a prediction by finding all the previous occasions on which α has been applied to state A , and forming a list αA of all the different *consequent states* which have been obtained. It can then assume that one of these consequent states will be obtained again, the probability of obtaining any particular state being estimated by the frequency with which it has been obtained in the past. In fact the automaton uses a rather more complex prediction process, to be described in a later section, and also makes some allowance for the possibility of a quite new state being generated. If α has never in the past been applied to A then, on our present assumption, no prediction can be made.

The automaton predicts for its current state the effect of applying each of its actions, where possible, and then does the same for each of the consequent states obtained. The planning 'tree' so obtained will branch rapidly. It is

scanned by a 'depth first' procedure (figure 4). Each branch of the tree must be terminated and there are the following ways in which this can occur:

1. By a state being predicted for which no predictions can be made.
2. By a state being predicted which has already been dealt with. For this to happen two branches of the tree must coalesce or one branch must loop back on itself.
3. By a state being predicted for which a decision formed before this planning phase is available.
4. The probability that a state will actually be reached, if the actions along the branch leading to it are implemented, will be lower the further along the branch the state is. Following Sandewall (1968) I call the limit below which this probability must not fall the *pruning level*. Once the pruning level is passed the branch is terminated.
5. By a 'long-stop' limit to the length of a branch.

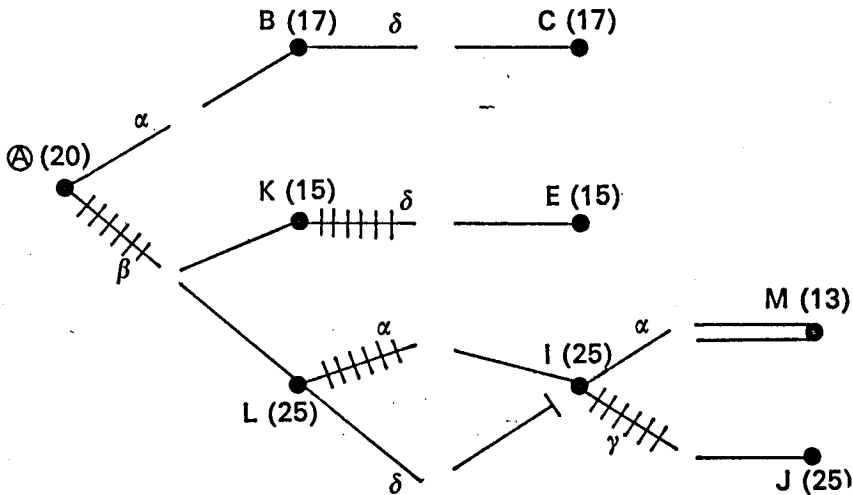


Figure 4. Simple planning tree based on the network of figure 3 (ii). The tree is scanned ('depth first') from top to bottom, and terminal and other values are assigned as described in the text. The decisions made are indicated by the cross hatching. Not shown are the *EXPLORE* and *MAKEPLAN* options at each node, nor the provision made for unpredicted consequences of actions. The coalescence of two branches at state I is detected and suitable action taken

As the tree is scanned the automaton forms decisions for all the states predicted. It can select as the operation of the decision either one of the actions or one of the complex operations *EXPLORE* and *MAKEPLAN*. When *EXPLORE* is found in a decision to be implemented, the automaton selects an action whose effect cannot be predicted. A random choice is made, possibly with a fixed bias. When *MAKEPLAN* is to be implemented, then the automaton makes a new decision for the state in question, going through a

complete planning phase to do so. Thus, to incorporate MAKEPLAN in a decision during the scanning of the planning tree is to say 'if I meet this situation I will think again'.

In order to decide which operation to incorporate in a decision the automaton assigns a numerical value to each of the alternatives open to it and selects the alternative with the highest value. If the effect of an action is unpredictable, then it is considered only as part of the option EXPLORE. The *value* of the decision is the value of the alternative selected. These decision values should not be confused with the values of state variables, which are not necessarily numbers.

How are the values assigned? To assign a value to an action in a particular state, the automaton takes the mean of the values of the decisions made for the predicted consequent states of the action, weighted by the associated probabilities. The value assigned to the EXPLORE option depends on the match between the state and the goal state, and on the actions remaining to be tried. A value is assigned to the MAKEPLAN option by weighing, very crudely, the long term benefit likely to be obtained by 'rethinking' against the short term loss of time.

It follows from what has been said that a decision cannot be made for a state before decisions have been made for all its predicted consequent states. The decision-making process must therefore work back from the tips of the branches towards the root of the tree. How are values assigned at the tips?

If a branch is terminated because a state is predicted which has a usable decision attached to it, then there is no problem. The decision value is just what is needed. The decision may have been made either earlier in the current planning phase, implying that two branches have coalesced, or at some earlier stage. A decision is not usable unless a quantity called its *reliability* is sufficiently high. The reliability of a decision is a measure of how much more the decision can be used before it should be remade. Reliabilities are set initially by the amount of computation used to form the decision, and are decreased whenever the decision is used in any way. Thus any decision which is in steady use is bound to be remade from time to time.

If a branch is terminated for any other reason, then EXPLORE or MAKEPLAN will be the operation selected, and a value will be assigned as for a non-terminal state. MAKEPLAN is normally selected.

When a planning phase has been completed, decisions with operations, and reliabilities, will have been formed for all the states encountered which did not already have an attached usable decision. In particular, a decision will have been attached to the current state. This decision the automaton will now implement. Normally there will then occur a predicted transition to a new state which will also have an attached decision. This the automaton will also implement, and so on, until either a decision with operation EXPLORE or MAKEPLAN is implemented, or something unpredicted happens. I shall say that the automaton is following a *plan* when it implements a sequence of

decisions in this way. However a plan ends, it is always followed by a planning phase.

When a plan goes wrong in some sense, for example if an unpredicted transition occurs, then the reliabilities of the decisions used in the plan are decreased.

The planning process involves several parameters which need careful adjustment. The most important choice is between planning trees which are large (to depth 10, say) and tedious to process but which produce good decisions, and trees which are small (to depth 4, say), fast, and possibly misleading. In particular, there are parameters which set the pruning level, the 'long-stop' cutoff, and the degree to which the possibility of unpredicted transitions is taken into account. If this last parameter is set so that the automaton 'expects the unexpected', then the effect will be to inhibit planning.

Of course, the mean depth of the tree will also depend greatly on the extent to which previously made decisions are incorporated into it. There is another important choice here between using past decisions frequently, with an increase in speed but an attendant risk of repeatedly using the same wrong action, and using past decisions rarely, with slower processing but more reliable results. There are parameters to control the amount by which a decision reliability goes down when the decision is used, and to adjust the threshold which the reliability of a decision must exceed if it is to be incorporated into the planning tree.

One other parameter controls the extent to which MAKEPLAN is used. In practice, too free a use of MAKEPLAN can lead to the automaton failing to explore when it should. Thus, this parameter also needs to be set with some care.

The memory structure

As already made clear, the memory of the automaton is a network. Each node of the network is a complicated affair which we may call a *record*. Records have the following components:

- (a) a state
- (b) a list of actions tried with (pointers to) their observed consequent states (to be precise, to the records containing the consequent states), and a note of how often each consequent state has been obtained.
- (c) a list of decisions each keyed to a goal state, and each containing an operation, a value, a reliability, and a list of (pointers to) expected consequent states.

Virtually all processing by the automaton is a matter of updating this network or drawing information from it. In particular, planning is carried out entirely in terms of this memory structure. The nodes of the planning trees are nodes of the network which are 'visited' as the planning process demands.

I have so far said nothing about how the automaton locates a record in its memory given the appropriate state but no direct pointer. This is a very common need. It is met by making each record of the network the tip of a branch of a special sorting tree, the tests at the branch points being simple tests on the values of successive variables of the state. The reason for using a sorting tree rather than, say, simply a list is speed of access. The greater the number of records the greater the benefit.

There are reasons why it is sometimes efficient to delete records from the memory network. The most obvious of these is in order to make room for newly created records, the supposition being that the total memory storage capacity is limited. A second reason is because the records in question, belonging to the distant past, seem either irrelevant to, or actually misleading in, the current situation. (This remark will have more point in the context of 'generalisations'.)

Some simple and rather arbitrary 'forgetting' processes have been built into the automaton on an experimental basis. For a record to be fully forgotten, it must be separated from the memory network, so that no operation over the network ever encounters it. This means detaching it from the sorting tree *and* independently from the transition links, that is, the actual network links.

Each network link has a *strength* associated with it which depends on the number of occasions on which the transition has actually occurred, and which decays with time. Any attempt to make use of a link whose strength is below a specified threshold merely results in its deletion.

Detachment from the sorting tree is achieved by limiting the amount of branching which can occur at the branch points immediately prior to the tips. The final 'twigs' are arranged in order of occurrence, with the record corresponding to the last state encountered heading the list. When a state is actually observed its record always goes to the head of the appropriate list. If it was not already present on the list then, assuming that the list has reached its full length, the last record on the list will be discarded.

Generalisation

Now let us return to the task the automaton faces when it has to predict the effect of implementing a given action in a given state. We have previously assumed that it merely referred to previous applications of the action to that state. However, this would be quite unsatisfactory, for it would mean that a difference in one variable value could prevent a past state being recognised as functionally equivalent to the current state even though that difference were quite irrelevant to the effect of the action being considered.

States must be recognised as equivalent for purposes of prediction even though they are not identical. The criterion for equivalence must itself be dependent upon the action under consideration. In the terminology of Michie and Popplestone, we must have a non-trivial *equiv function* (see Michie 1968).

The automaton uses the following system for defining equivalence. With each action is kept a record of those variables of the state which influence the effect of the action. Also kept with each action is a record of those variables whose values are never altered by it. These two sets of *significant* variables need not be the same. When the effect of action α on state A must be predicted, all states which agree with A in the significant variable values are found, together with all the states which have been obtained by applying α to them. For each such consequent state, the variables whose values are known not to be affected by the action α are given the values which they have in A (figure 5). This last step will greatly reduce the number of distinct consequent states, and those remaining form the actual prediction list αA . As before, estimated probabilities can be attached to the states of αA based upon their frequencies of occurrence in the past.

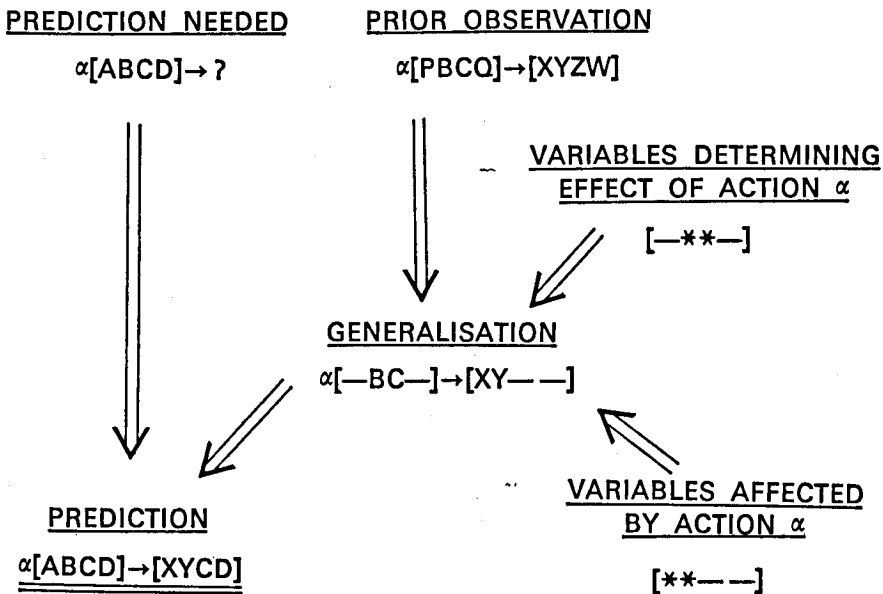


Figure 5. The stages in the generalisation process. An asterisk indicates a significant variable. The generalisation itself may or may not be permanently stored. Normally more than one previous observation is taken into account

This process for obtaining αA can be time-consuming when the number of states in the memory which are equivalent to A is large. It is therefore wasteful to recompute αA every time it is required, and provision is made for inserting into the memory the outcome of such a computation so that it stands as a 'generalisation' to be quickly available whenever useful. Thus the generalised state A' , which is A with 'irrelevant' markers attached to the variables which do not influence the effect of α , is inserted into the memory network and points to the consequent states αA obtained in the generalisation process.

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

Whenever a prediction is required for a state, then these generalisations are checked to see if the relevant one has already been made.

The generalisation process has one property which is worth noting, the 'pink elephant' property. The automaton can correctly predict the existence of possible states which it has never actually observed, although it must have experienced, each of the variable values which form them. In human terms, I can visualise a pink elephant even though I have never seen one. I have, however, seen both elephants and pink objects – it is the combination which is new. New states of this 'imaginary' type are added to the memory network in order that decisions can be associated with them, but they are more quickly discarded than 'real' states.

Although the significant variables for each action are given to the automaton, it will still sometimes make predictions which are wrong. It may even predict states which the black box cannot possibly generate. If a faulty prediction is made into a permanent generalisation, then the automaton's performance may suffer greatly. Thus the criterion which determines when a prediction should be permanently preserved is very important. In practice, the criterion is that αA shall consist of a single state that has been obtained at least a specified number of times. There is a choice here between a low threshold leading to many generalisations, fast processing and mistakes, and a high threshold which is slow but reliable.

Automatic parameter control

Each of the heuristic processes described can be adjusted by at least one parameter. In practice, it turns out to be very difficult to so adjust all these parameters that good performance is obtained in all of the situations which may be encountered. In particular, when the automaton's plans are being successful the parameter settings need to be different from when its plans are failing. The automaton therefore monitors its own performance at a higher level than that at which individual actions are selected, and adjusts its parameters accordingly.

Specifically, it notices when an unexpected transition occurs, or when a planned sequence of actions returns it to a state previously encountered, or when the decision values are uncomfortably low; and in each of these cases increases the value of an 'alarm' parameter which determines other parameters linearly dependent on it. The value of the alarm parameter falls quickly to zero when things are going well.

THE CONTROL TASK

The environment

Having completed the description of the basic version of the automaton, I shall now describe the special control task to which it has been applied.

The automaton 'lives' in an $n \times m$ chequer-board environment, occupying

at any moment one of its squares. Typically, both n and m are ten or less. Some of the other squares of the environment, including all those at the edge, are occupied by letters of the alphabet (see figure 6). The analogy of a small animal living on the floor of an enclosure which has boundary and interior walls is not too misleading.

E1

A	D	B	F	A	A	A	A
R				E	A	H	A
P		↑		S	T		J
O							K
A	N	M	G	U	V	L	A

E2

A	D	B	F	A	A	A	A
G				I	A	H	A
G				I	J		C
G	←						C
A	K	K	K	K	K	K	A

E3

A	D	K	F	A	A	A	A
G				C	A	H	A
E				L	K		J
E				→			K
A	C	J	B	L	B	G	A

Figure 6. The three 'chequer-board' environments used in the experiments. The automaton (indicated by the arrow) must be adjacent to and facing the letters *F*, *D* for the actions *EAT*, *DRINK*, respectively, to be effective. In mode 2 the automaton is warm only when in the square adjacent to the letter *H* (its 'nest')

The automaton is oriented to the 'top', 'left', 'right' or 'bottom' of the environment, and can move by stepping (action *STEP*) into the square immediately before it, provided that that square is not occupied by a letter – if it is, then the automaton's attempt to step has no effect. It can also turn through a right angle to its left or right (actions *LEFT*, *RIGHT*). It can move at any time.

The automaton can perceive very little of its surroundings at any particular moment. It can detect only which letter lies directly before it (state variable *LETTER*), and how many empty squares there are between it and the letter (state variable *DISTANCE*). It cannot detect what lies to its right and left.

Initially the automaton does not know how its movements will affect its perception. It does not know, for example, that the action sequence LEFT RIGHT will leave its perceptions unchanged nor, generally, does it know that it is moving around on a flat surface.

The automaton is quite deliberately denied this information, which would permit powerful but *ad hoc* search strategies to be used. We must accept that, with this added burden of ignorance, performance will be unimpressive compared with an *ad hoc* design, at least in the initial stages of an 'incarnation'.

The automaton is motivated by adding three state variables whose values it is important it should control. Without being too misleading, these may be described as measures of the automaton's 'hunger', of its 'thirst', and of the 'temperature' it is experiencing (state variables HUNGER, THIRST, TEMPERATURE).

The automaton is equipped with two special actions EAT and DRINK. When EAT is implemented at one particular point in the environment, then the variable HUNGER takes a highly desirable value. As time passes, however, HUNGER spontaneously slips back to an undesirable value. EAT has no effect elsewhere in the environment. DRINK has similar properties. The value of TEMPERATURE depends on the automaton's location in the environment. In general some areas of the environment are 'warmer' than others. The interactions between these three motivational variables make the control task both interesting and difficult.

To sum up, the automaton's state is made up thus

[LETTER DISTANCE HUNGER THIRST TEMPERATURE],

with the last three variables having large importance weights, such that HUNGER is more important than THIRST, and THIRST than TEMPERATURE. As already stated, if the total desirability exceeds a threshold then the automaton stops acting (sleeps) until it falls again. A natural criterion of performance for the automaton within this environment is the proportion of time for which it is asleep. The automaton's actions are STEP, LEFT, RIGHT, EAT, DRINK.

Even this very restricted type of environment is capable of systematic variation in two important ways. Firstly, the variation through time and space of the HUNGER, THIRST and TEMPERATURE variables can be made simple or complex. In the simple environments used so far all three are merely boolean variables. Secondly, *ambiguity* can be incorporated into the environment by repeating some of the letters in it. If there are two occurrences of *K*, say, then the LETTER/DISTANCE pair *K0* is ambiguous in the sense that the automaton will perceive it at two different positions in the environment.

Performance

The POP-2 (Burstall and Popplestone 1968) program which simulates the automaton/environment system is fully debugged and has no obvious logical

errors. This proves that the various ideas and heuristics incorporated into it *can* be made detailed and precise in a consistent way. However, it is no proof that these ideas and heuristics are genuinely useful for some or all control tasks. Ideally we wish to know under what circumstances, if any, each heuristic is useful, and then how useful. Further, different heuristics will interact and we wish to know how.

As yet only a small amount of experimentation to these ends has been performed. However, enough has been done to establish the general level of performance and to identify the main limitations of the present design.

As mentioned at the end of the previous section, it is useful to distinguish two dimensions of variation for the environment. The degree of ambiguity which is built in can be varied, as can be the complexity of the variation of the HUNGER, THIRST and TEMPERATURE values. Experimentation has been restricted to three environments (see figure 6) each of which has two modes. The environments are:

- E1. All letters different. Therefore no ambiguity.
- E2. Each 'wall' formed of a single letter, but different letters for different walls. Therefore ambiguity, but ambiguity which will often be helpful rather than confusing.
- E3. An allocation of letters with various awkward duplications. Therefore unhelpful ambiguity.

In each of these environments, the letters F and D occur uniquely in the same positions and indicate where 'food' and 'drink' can be obtained, that is, where the actions EAT and DRINK are effective. Where the automaton can 'see' a letter from more than one direction, special steps are taken to avoid any unwanted ambiguity.

The two modes for each environment, M1 and M2, are as follows:

- M1. Temperature high throughout the enclosure. After eating hunger disappears for 10,000 time units, and then returns at full strength. After drinking, thirst disappears for 5,000 time units and then returns at full strength.
- M2. Temperature low throughout the enclosure except at the top of the right hand passage – the automaton's 'nest'. Hunger and thirst as before but with the time intervals tripled.

To make the goal-selection mechanism operate sensibly, especially desirable values are assigned to HUNGER and THIRST for a brief period immediately after eating and drinking, respectively.

A typical incarnation starts with a 'training' session in which the automaton is shown the desirable effect of eating and drinking at the appropriate points. We can regard this as giving it basic 'reflexes'. Without this help the automaton's search task would be impossibly difficult. This training takes 30,000 time units – an artificial figure which in this case does not indicate the amount of computation performed by the automaton.

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

The automaton is then placed in its 'nest', facing the letter H, and left to its own devices. It is expected to explore sensibly, to find its way to food and drink, and to return to its nest for warmth (M2 only). When all its needs are satisfied it should sleep until hunger or thirst wakes it up. Ultimately it should settle into a fairly steady pattern of activity.

The standard length of an incarnation is 150,000 time units for M1, and 250,000 for M2: roughly 75 minutes and 120 minutes of real time on an Elliott 4130 computing system. A planning phase may take from 100 to 10,000 time units.

INCARNATION	<i>environ- ment</i>	<i>time not hungry</i>	<i>time not thirsty</i>	<i>time not cold</i>	<i>time asleep</i>
A	E1/M1	50.1	28.4	—	19.2
B	E1/M1	57.6	44.0	—	29.2
C	E1/M1	46.9	30.0	—	16.1
D	E2/M1	40.0	35.3	—	21.2
E	E2/M1	26.7	10.0	—	3.3
F	E2/M1	54.4	31.6	—	19.3
G	E3/M1	4.4	6.7	—	0.0
H	E3/M1	18.2	10.1	—	0.0
I	E3/M1	48.0	17.7	—	11.7
J	E2/M2	24.0	30.0	18.4	0.0
K	E2/M2	72.7	51.9	36.7	13.0
L	E2/M2	12.0	6.2	56.8	0.0

Table 1. Summary of the automaton's performance in three incarnations in each of four variants (E1/M1, E2/M1, E3/M1, E2/M2) of the basic chequer-board environment. The incarnations are independent, and the automaton starts each with no knowledge of its task except EAT and DRINK reflexes. TEMPERATURE is a variable to be controlled only in J, K and L. The figures given are percentages of the total lifetime, which is 250,000 time units in variant E2/M2 and 150,000 time units otherwise. Details of the environment variants and the automaton parameter settings are given in the text.

Table 1 gives a summary of the automaton's performance over twelve incarnations, three in each of E1/M1, E2/M1, E3/M1, E2/M2. The main features of its performance are as follows:

1. Learning does occur, and the automaton often does make a good job of its control task.
2. However, performance is erratic depending quite heavily on key random choices, and on the automaton not getting into situations such that it repeats long action sequences inappropriately. This can happen when there are ambiguities in the environment, or where there are opportunities for faulty generalisation. The root cause is the

automaton's assumption that only the current state is relevant to prediction.

3. Predictably, the introduction of 'unhelpful' ambiguities and of TEMPERATURE as a relevant variable impairs performance.

Figure 7 is a record of the first part of the most successful incarnation in E2/M2. To the onlooker, the automaton's behaviour appears a mixture of motionless 'thinking' or 'sleeping', and sudden bursts of movement.

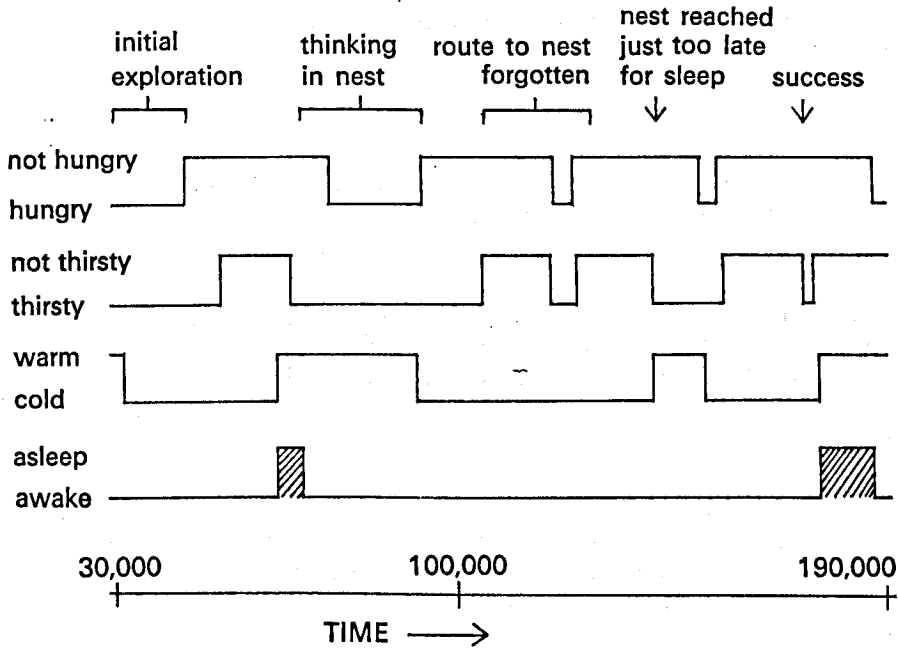


Figure 7. Part of incarnation k. The automaton sleeps only when it is warm and neither hungry nor thirsty

As already explained, the performance of the automaton may be adjusted by a considerable number of parameters. The parameter settings used during these experiments were obtained after a considerable amount of trial and error. They can be summarised as follows:

- (a) Large rather than small planning trees, except when the 'alarm' level is very low.
- (b) No allowance made during planning for unpredicted transitions.
- (c) Past decisions not used if alarm level at all high.
- (d) The operation MAKEPLAN only used when the alarm level is low.
- (e) 'Generalisations' formed as quickly as possible in E1 but only after three or four consistent transitions have been observed in E2 and E3.

During these trials the memory record deletion mechanisms were switched off.

As the performance of the automaton is clearly limited in major ways, I shall say no more about details of performance, and turn instead to a more general discussion of ways in which the design can be improved, and of the preliminary experiments which have been carried out to this end.

DISCUSSION

Prediction and generalisation

Two critical remarks may immediately be made about the automaton's present prediction mechanism. Firstly, it should be able to detect for itself the significant variables of the state for each action; secondly, it should be much more general and, for the chequer-board environment at least, should not assume that only the current state is relevant to a prediction.

To meet the first point an extension to the automaton has been made. Recall that a variable can be significant in either or both of two ways with respect to a particular action. The variable can be one of those which help determine the effect of the action or it can be one whose value is liable to be changed by the action.

The automaton typically has stored in its memory many examples of the effects of each action. It can therefore 'ruminate' occasionally and simply look to see, for each action, which variables usually have their values carried over unchanged, and which variables can usefully be taken into account by the prediction mechanism, in the sense that knowing the value of the variable enables the prediction to be made more precise.

This capability has been given to the automaton, and first trials indicate that this essentially inductive process will work both in the sense that, given a large and varied memory network, it will select as significant the correct variables, and also in the stronger sense, that starting a control task with all variables regarded as significant, and with a null memory network, the process will come to select the correct variables as the automaton explores. A similar inductive process was proposed to permit the Graph Traverser program to improve its problem state evaluation function (Doran 1967), and unpublished work by R. Ross provides additional evidence that the approach is sound (personal communication).

The second criticism of the prediction mechanism is a more complex one. The present mechanism can establish, store and use deductively generalisations of the following class

$$\alpha(x_j=a \wedge \dots \wedge x_k=b) \rightarrow (x'_m=p \wedge \dots \wedge x'_n=q)$$

where the Greek letter denotes an action, the unprimed x s refer to variables of the state prior to the action, and the primed x s refer to variables of the consequent state. I ignore the possibility of several alternative consequent states. It is implied in the notation that the value of a variable is unaltered by an action if the primed variable is not mentioned. The set of variables which appear for a particular action must always be the same.

That this set of possible generalisations is very restricted is obvious at a glance. For example, the generalisation

$$\alpha(\neg x_i = a) \rightarrow (x_i = b)$$

cannot be formulated. On the other hand this limited set does have the virtue that it arises fairly naturally from the way states are observed and stored. How far does it impose a limitation on the automaton's performance?

In the context of the chequer-board environment, consider the following generalisations

$$\text{EAT} (\text{DISTANCE} = 0 \wedge \text{LETTER} = \text{F}) \rightarrow (\text{HUNGER} = 0)$$

$$\text{EAT} (\neg (\text{DISTANCE} = 0 \wedge \text{LETTER} = \text{F})) \rightarrow ()$$

These exactly describe the effect of the action EAT. The closest the automaton can get to this is to establish all generalisations of the form

$$\text{EAT} (\text{DISTANCE} = a \wedge \text{LETTER} = b \wedge \text{HUNGER} = c) \rightarrow (\text{HUNGER} = d)$$

for observed values of a, b, c, d .

The automaton should be able to predict how the temperature will change as it moves about its environment. At present it formulates all generalisations of the type

$$\text{STEP} (\text{DISTANCE} = a \wedge \text{TEMPERATURE} = b) \rightarrow (\text{DISTANCE} = c \wedge \text{TEMPERATURE} = d)$$

for observed values of a, b, c, d , and thus tries to express the fact that stepping 'changes' the temperature. However, in a situation with a fairly complex temperature function, this is very clumsy and at best a crude approximation to the true state of affairs, which is much better described by the following types of generalisation:

$$\text{STEP} (\text{DISTANCE} = a) \rightarrow (\text{DISTANCE} = b)$$

together with

$$(\text{DISTANCE} = a \wedge \text{LETTER} = b) \rightarrow (\text{TEMPERATURE} = c)$$

Notice that the second class of generalisation makes no reference to an action.

Thus in the chequer-board environment the prediction mechanism is far from adequate, even if we could retain the assumption that it need take into account only the current state.

Ideally, the automaton would formulate an efficient copy or 'model' of that part of the mechanism generating its environment which is relevant to its control task.

It does not seem too difficult to enable the automaton to handle a much wider class of generalisations than it does now. The STELLA learning machine, for example, has an appreciably more general prediction mechanism (Andreae 1968.) A more complex prediction process would combine a number of individual generalisations in order to 'build up' the consequent state, and branches of the planning tree would be terminated more when the significant

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

features of the states could not be predicted than for any other reason. Of course, a very general prediction mechanism is inappropriate if the control tasks envisaged do not require it.

At present the automaton's planning is directly in terms of its experience in the following sense. Every state which appears on the planning tree is either already in the memory or is at once placed there, and is just the same kind of object as any other state in the memory. Thus it is not a great misuse of words to say that the automaton predicts the effects of actions by imagining actually doing them. This must in itself impose limitations on the power of the planning process, and some capability to use symbolic reasoning must be introduced sooner or later. To this end the automaton could be provided with a formal language in which to carry out all or part of its reasoning, in the tradition of the Advice Taker (McCarthy 1959), and in the way in which Green has applied an automatic theorem-proving program to the task of controlling a robot (Green 1969). A more remote possibility is to expect the automaton to learn to use one class of its perceptions as a source of symbols with which to represent all or part of its knowledge. The symbols would be manipulated 'mentally' by the actions appropriate to that class of perceptions. Something of this type seems to take place in human thinking. Some first steps have been made in this direction in the context of the STELLA learning machine (Andreae and Cashin 1969).

Complex planning

So far in this paper the word 'planning' has been used to refer to the process of predicting in detail the consequences of actions. However, this is rather a weak use of the word. It seems more appropriately used for a process which outlines a course of action, leaving the details to be filled in as the action is being carried out. Minsky (1961) discussed the value of some planning process which would break up a problem into a set of smaller subproblems by placing 'islands' in the search space. Planning has also been discussed by, among others, Newell, Shaw and Simon (1959), Travis (1964) and Hormann (1965). More recently, Sandewall (1968) has discussed in detail one type of planning and much of his analysis is relevant to the work described in this paper.

In the spirit of these discussions, some first experiments have been made with a version of the automaton which has the following properties:

1. Not only are the basic state transitions caused by a single action stored in the memory, but also the transitions which result from the implementation of a plan in the old sense, that is, which result from the implementation of a sequence of actions.
2. During a planning search, these complex transitions are manipulated just as if they were basic transitions, and a complex transition may be adopted as the operation of a decision. Only the fact that the

consequent state has been reached from the initial state is stored, not the actions which were used.

3. When a decision which has such a complex transition as its operation comes to be implemented, then the consequent state of the transition is set as a temporary 'subgoal', and a search directed towards this subgoal is initiated. This search may well involve the formation of plans involving further complex transitions, and so on.
4. When such a subgoal is achieved (that is, the state is reached) then the search is automatically resumed within which the subgoal was set up.

First trials with this system have demonstrated that the approach is sound, but indicate two problems. Firstly, the subgoals have a rather different genesis from the main goals which is an unattractive state of affairs. This is probably more because the main goal selection process is rather arbitrary than because the planning process is faulty. The second and deeper problem centres on the fact that complex transitions, like basic transitions (corresponding to basic actions), really act only from one subset of the state variables to another. The system implemented takes no account of this and is in consequence too rigid to be really useful. In some coherent way, a record of the relevant variables must be formed and kept with each complex transition.

The reader may wonder why the automaton should not store with each complex transition the sequence of actions which generated it. There seem two possible arguments against this, whose force will depend upon the class of control task envisaged. They are that the amount of storage of detail required may be too great, and that sequences of actions will not always be repeatable in detail, or have repeatable effects.

The reasons for dealing with complex transitions at all are to permit 'long-range' planning without impossibly large planning trees, and to escape from the hazards of planning in detail in a control situation which is too complex or too random to be predicted in detail.

CONCLUDING REMARKS

The approach to machine intelligence which I have followed in the work described in this paper can be seen as lying roughly midway between two extremes. At one extreme is the approach which would advocate equipping the automaton with the most general and precise inductive and deductive systems which we know how to program, however 'unnatural' this may in some sense seem to be. At the other extreme is the approach which is primarily impressed by the abilities of the human brain, and which is therefore attracted by large networks of similar elements operating in parallel (consider the paper in this volume by Kiss). Just as we can ask how far the present automaton's capabilities could be encompassed within some formal system, so also we can ask to what extent they could be performed by a network of elements operating in parallel.

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

At first sight these two extremes seem far apart. However, it may be that each will find its place in a general theory relating the design of intelligent machines to the tasks which they must perform and to the constraints under which they must perform them.

Acknowledgements

The research described in this paper is financed by the Science Research Council under the supervision of Professor Donald Michie. I have also benefited from discussions with Professor J.H. Andreae of the University of Canterbury, New Zealand, and with my colleague Dr R. M. Burstall.

REFERENCES

- Andreae, J.H. (1968) Learning machines: a unified view. *Pergamon Encyclopaedia on Linguistics, Information and Control*. London: Pergamon Press.
- Andreae, J.H. & Cashin, P.M. (1969) A learning machine with monologue. *Journal of man-machine Studies* 1 (in press).
- Burstall, R.M. & Popplestone, R.J. (1968) POP-2 reference manual. *POP-2 Papers*. Edinburgh: Edinburgh University Press.
- Doran, J.E. (1967) An approach to automatic problem-solving. *Machine Intelligence 1*, pp. 65-87 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.
- Doran, J.E. (1968) Experiments with a pleasure-seeking automaton. *Machine Intelligence 3*, pp. 195-216 (ed. Michie, D.). Edinburgh: Edinburgh University Press.
- Green, Cordell (1969) Theorem-proving by resolution as a basis for question-answering systems. *Machine Intelligence 4*, pp. 183-205 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.
- Hormann, A. (1965) Gaku: an artificial student. *Behav. Sci.*, 10, 88-107.
- McCarthy, J. (1959) Programs with common sense. *Proceedings of a Symposium on the Mechanisation of Thought Processes*, pp. 75-91. London: HMSO.
- Michie, D. (1968) 'Memo' functions and machine learning. *Nature*, 218, no. 5136, 19-22.
- Michie, D. & Chambers, R.A. (1968) BOXES: an experiment in adaptive control. *Machine Intelligence 2* (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.
- Minsky, M. (1961) Steps towards artificial intelligence. *Proc. Inst. Radio Engrs*, 49, 8-30.
- Newell, A., Shaw, J.C. & Simon, H.A. (1959) Report on a general problem-solving program. *Proceedings of the International Conference on Information Processing*, pp. 256-64. Paris: UNESCO.
- Sandewall, E. (1968) A planning problem solver based on look-ahead in stochastic game-trees. Uppsala University, Department of Computer Sciences, *Report no. 13*.
- Travis, L.E. (1964) Experiments with a theorem-utilizing program. *AFIPS*, 25, 339-58. Spring J.C.C. Baltimore: Spartan Books.

Freddy in Toyland

R. J. Popplestone

Department of Machine Intelligence and Perception
University of Edinburgh

Introduction

This paper is about how Freddy, a robot that we in Edinburgh are building, might structure the input from his sense organs, and build an internal model of his toy universe. Freddy can then use this model to decide how to do things like fetching an object when asked to do so.

1. SELF ORGANISING SYSTEM

Let us consider how Freddy might make sense of his environment given a set of primitive operations.

Suppose that Freddy has one input which is a teletype on which a stream of characters is being typed. These happen to form sentences of the English language, but Freddy doesn't know this. Freddy will first try to see structure in this stream and then relate this structure to the inputs from his other sense organs. How could the first process of structuring the input be done?

A phrase structure grammar (PSG) can be used as a means of describing languages. It is a useful first step in describing English. Suppose Freddy had the wherewithal to construct a PSG. Could he build a grammar to describe his input? I am going to describe rather informally how I think he might.

A PSG can be built out of symbols. The *terminal symbols* T of the grammar are simply the set of characters of the input. A *symbol* s of the grammar is either a terminal symbol, or a sequence of symbols, or a set of symbols. The set S of all symbols which can be built from T is called the set of symbols over T . A sequence of symbols (s_i) is said to be parsed into a sequence (t_i) of symbols if (t_i) is obtained from (s_i) by replacing a subsequence of (s_i) by a single symbol, or by replacing one member of (s_i) by a set containing it together with an indication of which member of the set was involved.

Thus '*the cat sat on the mat*' parses to '*(the) cat sat on the mat*'. Where *(the)* is that symbol which is the sequence of terminal symbols 't' 'h' and 'e' and '*(the) (cat) (sat) (on) (the) (mat)*' parses to '*(the)[1, {(cat) (dog) (pig)}] (sat) (on) (the) (mat)*' where $[1, \{(cat)(dog) (pig)\}]$ indicates that member no. 1 of the set $\{(cat) (dog) (pig)\}$ was involved.

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

A grammar G is then just a subset of the set S of all symbols over T . G specifies a language because only those symbols that are in G may be used for parsing. How then do we 'grow' a grammar? That is, how do we decide that one set of symbols should be used for making substitutions?

Let us notice first that English is highly redundant. That is to say that of all possible sequences of n characters, only a few will be sentences of English.

Let the symbol s have a probability $p(s)$. Let us assume that we have a meaningful way of defining and measuring the probabilities of non-terminal symbols. The *score* of a sequence (s_i) is defined to be $\log(p(s_i))$. It is of course the log of the probability of the occurrence of the sequence if there are no interactions between the members. If one of the s_i is a set member $[n,c]$ then $p([n,c])$ is defined to be the product of the probability of the occurrence of the set c and the probability of the occurrence of the n th item of c . A parsing can be regarded as a *coding* of one sequence into another. Freddy will try to parse his input into a sequence of minimal score. I will try to show that it is plausible that such a sequence will reflect the grammatical structure of English.

Consider first the replacement of a subsequence of a symbol. If the subsequence is (s_i, \dots, s_j) and it is replaced by s , then the change in score is $\log(p(s)) - (\log(p(s_i)) + \dots + \log(p(s_j)))$.

Thus there is a decrease in score in those cases for which s is more likely to occur than if the s_i were independent.

Thus the replacement of *the* by *(the)* will lead to a greater reduction in score than the replacement of *e c* by *(e c)*.

There is no immediate advantage in score in replacing a symbol by a set containing it, since both the probability of the set occurring and the probability of the symbol occurring within the set are used in calculating the score. However, there can be a delayed advantage because the set may itself be a member of a high-probability sequence. Thus if many sentences had the form

(the) \langle noun \rangle \langle verb \rangle \langle preposition \rangle *(the)* \langle noun \rangle \langle S1 \rangle

the advantage to be gained from substituting the set \langle noun \rangle for *(cat)* in '*the cat sat on the mat*' lies in the ability to parse this further into form \langle S1 \rangle . The score is then that required to specify \langle S1 \rangle together with the scores required to specify *(cat)* as a member of \langle noun \rangle , *(sat)* as a \langle verb \rangle , *(on)* as a \langle preposition \rangle and *(mat)* as a \langle noun \rangle , which is less than that required to specify the words of the sentence as independent symbols.

However, there is a very large space of possible parsings to be searched. How should this be organised? One could try some sort of Graph Traverser search (Doran 1968) for a minimal score using only those symbols whose occurrence was high enough for their probability to be estimated accurately.

In choosing sets of symbols to group together, the methods of numerical taxonomy (Sokal & Sneath 1963) could be used. A measure of the likeness of two symbols s and t would be the number of sequences actually occurring

in text which on substituting t for s give a sequence which also occurs. Thus, substituting (*dog*) for (*cat*) in (*the*) (*cat*), gives a sequence which can occur.

2. ON SEARCH STRATEGIES, AND SETS

One often has to search a large set for an optimal member. In such a situation one wishes to avoid an explicit listing of the whole set, but instead to use a method which will 'climb uphill'. The Graph Traverser (Doran 1968) provides one standard method by which a set can be represented by a connected graph and explored by moving along the arcs.

I want to suggest an alternative representation for sets. I will illustrate this by an example. If I want to choose one of the set of all boats I may go to a naval architect. The conversation might run as follows:

Architect: Do you want a sailing or power boat?

Me: Sailing

Architect: Sloop, cutter, ketch, schooner or yawl-rigged?

Me: Sloop

Architect: What about the length?

Me: 25 feet

Architect: I now know just what you want.

The thing to notice is that the later questions depend on the answers to the earlier ones. Thus, one would not be asked about the rig of a power-boat. One can regard the possible questions as forming a tree, with *OR*-nodes at the points where choices are made. There will be *AND*-nodes at some places, for instance in specifying the areas of the sails, because these can be specified in parallel.

How could one represent this process in a language like POP-2 (Burstall and Popplestone 1968)? A set S is a function

$S: \text{choice-tree} \rightarrow \text{item, choice-tree}$

that is, S takes a choice-tree and produces an item and a new choice-tree. If the choice-tree is a complete specification of a member of the set, then the item will be that member. Otherwise the item will be *UNDEF* (undefined) and the choice-tree produced as result of the function will be extended to the next *OR*-nodes.

In passing, it is worth noting that this representation is a generalisation of representing a set as an array, while the graph-traverser is a generalisation of the list-representation.

3. MEMBERSHIP FUNCTIONS AND SET-OPERATIONS

The usefulness of any representation of sets depends on how easy it is to implement the standard set operations like union and intersection.

First, let us suppose that each set S has a membership function *MEM-FUN*(S). The *FNPROPS* facility of POP-2 permits us to attach attributes

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

to functions in this manner. Clearly any operation involving sets must produce a new membership function.

The simplest set operation is *UNITSET* which takes a POP-2 item and produces a set containing that item. The choice-tree required to specify that item is just a terminal-node *TERMIN*.

The set of real numbers, *REALS* has as members all POP-2 numbers. A real is specified by a 1-node choice-tree

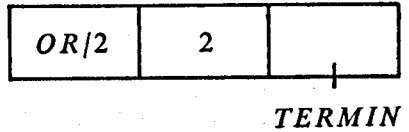
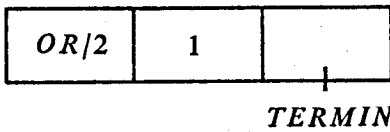
<i>REAL</i>		3.14
-------------	--	------

.

If *S, T, . . . , W* are sets then let *U* be the union of them. Then the choice-tree for *U* will have an *OR*-node that specifies which of the component sets is being referred to. Thus one might write in POP-2

UNION([%UNITSET('DOG'), UNITSET('CAT')%]) → ANIMALS;

The trees for *DOG* and *CAT* would then be:



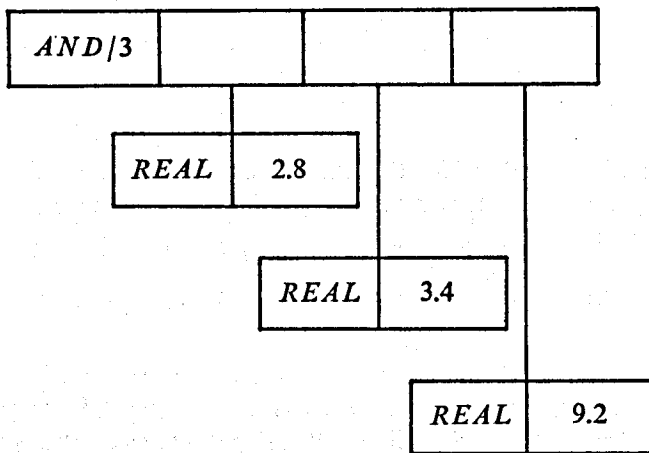
The only other set operation that is easy to implement is the direct product. If as before *S, T, . . . , W* are sets, and *L* is a word, then we can form the labelled direct product *D* of the sets by

DIRPRODL(*L*, [%*S, T, U, V, W*%]) → *D*;

The label *L* is desirable because it permits conceptual distinctions to be made which seem natural. Thus one might say

DIRPROD('POINT3D', [%*REALS, REALS, REALS*%]) → *SPACE3D*;

The choice trees for members of a direct product have an *AND*-node connecting the choice trees for the components. Thus a point of *SPACE3D* could be specified by:



The actual members of a direct product would be POP-2 records. Hence the membership function would just have to check the *DATAWORD*, which is the label of the direct product. It would be desirable to keep a record of the selector functions of the record class somewhere in the set, so that additional structural dependent operations such as vector addition could be defined.

The last of the set constructions that I am going to describe is the operation which takes a set *S* and produces the set *P* of all finite sequences of members of *S*. A member of *P* will be specified by a chain of *OR*- and *AND*-nodes as in figure 1, where the final *OR*-node is marked by a choice 2.

The POP-2 would be:

FINSEQ(*S*) → *P*;

The members of *P* will be lists of members of *S*.

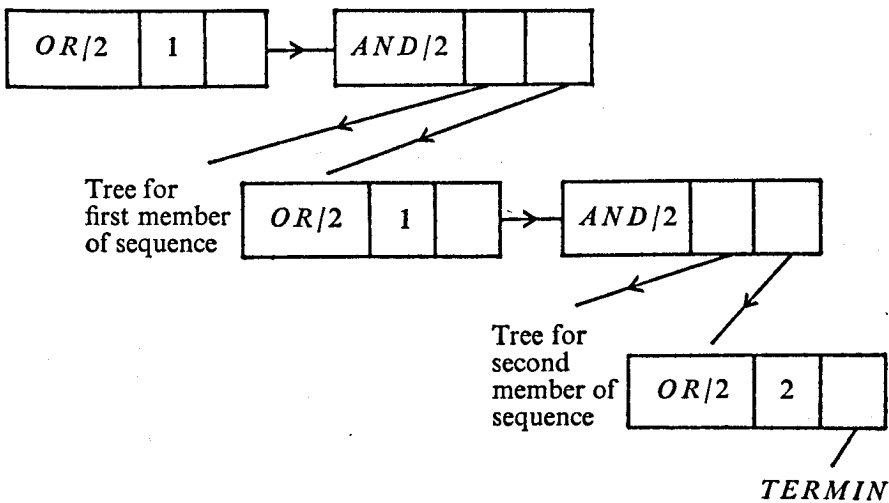


Figure 1. The choice-tree for a finite sequence

Let us brush aside Hamlet's comment that 'There are more things in heaven and earth, Horatio, Than are dreamt of in your philosophy', and suppose that Freddy lives in a universe of blocks, cylinders and cones.

A rectangular block can be specified by two 3-vectors giving the position of a corner and the length and direction of a side, together with two numbers which give the lengths of the other two undetermined sides. So, in POP-2

DIRPROD("BLOCK", [%SPACE3D,SPACE3D,REALS,REALS%]) → BLOCKS;

Likewise:

DIRPROD("CYLINDER", [%SPACE3D,SPACE3D,REALS%]) → CYLINDERS;

DIRPROD("CONE", [%SPACE3D,SPACE3D,REALS%]) → CONES;

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

Objects can be cones, cylinders or blocks. Hence:

$\text{UNION}([\% \text{ CONES, CYLINDERS, BLOCKS } \%]) \rightarrow \text{OBJECTS};$

Finally, a universe is a sequence of objects.

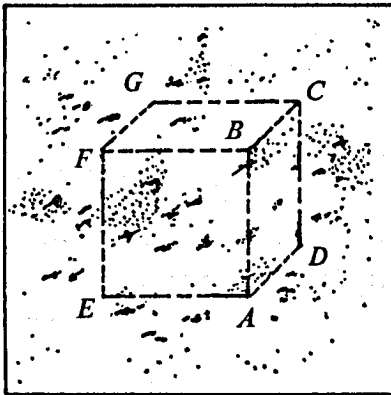
$\text{FINSEQ}(\text{OBJECTS}) \rightarrow \text{UNIVERSES};$

4. VISION

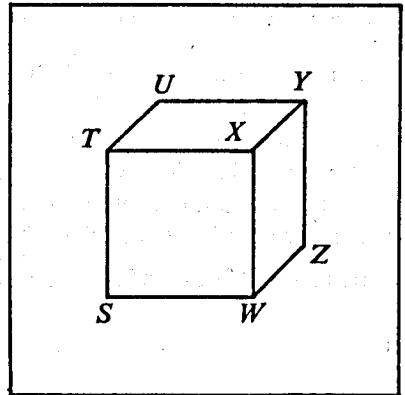
How can Freddy select one of this set of universes and say 'That is where I live'? Consider what his eyes tell him. The raw information produced by a TV camera can be regarded as a retinal function, which is a function which takes a point in 2-space to a brightness level. One could, with varying degrees of sophistication, write an eye-simulator, which would take an 'imagined' universe and produce a retinal function, which is what the universe would be expected to look like. If the imagined universe were like the real universe then there would be some sort of match between the retinal functions produced by the real eye and the 'mind's eye'.

There are two linked problems to be solved before the promise inherent in the last paragraph can be realised. The first is whether there is any possibility of converging on an imagined universe that is a faithful representation of the real one. The second is how to compare two retinal functions bearing in mind that the subtle variations in light and shade of the real world are unlikely to be simulated.

Let us look first at the problem of matching retinal functions. Edges, where there are sharp transitions between light and shade seem important, and indeed line drawings are a common way of conveying visual information in a much condensed form. Thus one would first form the *derived retinal functions* by dividing the retina up into squares and defining the derived function to be 1 only on those squares whose light-level differed from that of one of their immediate neighbours by more than a threshold.



A



B

Figure 2

If this differencing and thresholding operation were applied to an input from a real camera then the result would be something like figure 2A. From the imagined universe one could get figure 2B. One can compare 2A and 2B by saying that every point on 2A that is black (i.e., has a derived function = 1) must be accounted for by a point on 2B, and conversely. That is, the dissimilarity of the two pictures is measured by adding together the distances of each black point on 2A from the nearest black point on 2B and conversely, and then taking the weighted sum of the totals.

Let us now recall that objects of the imagined universe are specified by a choice-tree, which is a structure containing real numbers among other things. Suppose that as in 2B we have specified one cube. Changing the real numbers in the choice-tree will then move the image of the curve over 2B. As it moves so the value of the matching function will change. There will be maximum matching when the two images coincide. It seems plausible that a function minimisation algorithm would find this minimum, especially if the lines in 2B which were not accounted for in 2A were given small weight. The problem of getting stuck in local minima is rather serious. Thus, matching S, T, U of 2B against A, B, C of 2A would give a near minimum that could only be resolved by moving the face $WXYZ$ through and past ABC .

In general a set-searching function is needed which will attempt to maximise the value of some function defined over the set. The searching function does not know anything about the set. It simply tries to make a choice-tree for which the corresponding set member is optimal. The possible changes to a choice-tree are to change a real terminal node (which is a continuous change) or to change the choice made at an *OR*-node (which is a discrete change). Such a set-searching function could in principle search the set of all universes to find one which would account for quite a complicated scene. The problem of one body partially occluding another would be dealt with by the eye-simulator.

It seems that for complex scenes some form of pre-processing to look for clues is desirable. Buchanan *et al.* (1969) found that this certainly paid off in the case of *HEURISTIC DENDRAL*. Thus the meeting of three straight lines would suggest the existence of a block.

In the case of patterns that are describable by linear functions, the work of Roberts (1963) shows that the approach outlined in this paper is feasible.

5. CONCLUSION

The ideas outlined in this paper are untested. The future will show how workable they are. However, I feel that both the principle of reducing information stated in the first part, and the use of a model to explain sensory input as in the second part, must form part of an integrated intelligent machine. Certainly there is an information reduction in converting the retinal image of a cup into the word 'cup'. Since one can draw or paint imaginary objects quite convincingly, it seems that we have quite a good internal

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

model of visual processes like occlusion of objects and variation of light and shade.

Acknowledgements

Many of the ideas in this paper arose in discussion with several people, especially J.M. Foster and D.B. Vigor. Also, it has come to my notice that some work by M.G. Notley (1968) has been on lines parallel to the first part of this paper. Ideas relevant to the present discussion are developed elsewhere in this volume by my colleague J. E. Doran.

REFERENCES

- Buchanan, B., Sutherland, G. & Feigenbaum, E.A. (1969) Heuristic Dendral: A program for generating explanatory hypotheses in organic chemistry. *Machine Intelligence 4*, pp. 209-54 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.
- Burstall, R.M. & Popplestone, R.J. (1968) POP-2 Reference Manual, in *POP-2 Papers*, Edinburgh: Oliver and Boyd.
- Doran, J.E. (1968) New developments of the Graph Traverser. *Machine Intelligence 2*, pp. 119-35 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.
- Notley, M.G. (1968) The cumulative recurrence library. ICI Research Division, Report no. 31,047.
- Roberts, L.G. (1963) Machine perception of three dimensional solids. *Technical Report No. 315*, Lincoln Laboratory, MIT.
- Sokal, R.R. & Sneath, P.H.A. (1963) *Principles of Numerical Taxonomy*, London: W.H. Freeman and Company.

Some Philosophical Problems from the Standpoint of Artificial Intelligence

J. McCarthy

Computer Science Department
Stanford University

P. J. Hayes

Metamathematics Unit
University of Edinburgh

Abstract

A computer program capable of acting intelligently in the world must have a general representation of the world in terms of which its inputs are interpreted. Designing such a program requires commitments about what knowledge is and how it is obtained. Thus, some of the major traditional problems of philosophy arise in artificial intelligence.

More specifically, we want a computer program that decides what to do by inferring in a formal language that a certain strategy will achieve its assigned goal. This requires formalizing concepts of causality, ability, and knowledge. Such formalisms are also considered in philosophical logic.

The first part of the paper begins with a philosophical point of view that seems to arise naturally once we take seriously the idea of actually making an intelligent machine. We go on to the notions of metaphysically and epistemologically adequate representations of the world and then to an explanation of *can*, *causes*, and *knows* in terms of a representation of the world by a system of interacting automata. A proposed resolution of the problem of freewill in a deterministic universe and of counterfactual conditional sentences is presented.

The second part is mainly concerned with formalisms within which it can be proved that a strategy will achieve a goal. Concepts of situation, fluent, future operator, action, strategy, result of a strategy and knowledge are formalized. A method is given of constructing a sentence of first-order logic which will be true in all models of certain axioms if and only if a certain strategy will achieve a certain goal.

The formalism of this paper represents an advance over McCarthy (1963) and Green (1969) in that it permits proof of the correctness of strategies

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

that contain loops and strategies that involve the acquisition of knowledge; and it is also somewhat more concise.

The third part discusses open problems in extending the formalism of part 2.

The fourth part is a review of work in philosophical logic in relation to problems of artificial intelligence and a discussion of previous efforts to program 'general intelligence' from the point of view of this paper.

1. PHILOSOPHICAL QUESTIONS

Why artificial intelligence needs philosophy

The idea of an intelligent machine is old, but serious work on the problem of artificial intelligence or even serious understanding of what the problem is awaited the stored-program computer. We may regard the subject of artificial intelligence as beginning with Turing's article 'Computing Machinery and Intelligence' (Turing 1950) and with Shannon's (1950) discussion of how a machine might be programmed to play chess.

Since that time, progress in artificial intelligence has been mainly along the following lines. Programs have been written to solve a class of problems that give humans intellectual difficulty: examples are playing chess or checkers, proving mathematical theorems, transforming one symbolic expression into another by given rules, integrating expressions composed of elementary functions, determining chemical compounds consistent with mass-spectrographic and other data. In the course of designing these programs intellectual mechanisms of greater or lesser generality are identified sometimes by introspection, sometimes by mathematical analysis, and sometimes by experiments with human subjects. Testing the programs sometimes leads to better understanding of the intellectual mechanisms and the identification of new ones.

An alternative approach is to start with the intellectual mechanisms (for example, memory, decision-making by comparisons of scores made up of weighted sums of sub-criteria, learning, tree search, extrapolation) and make up problems that exercise these mechanisms.

In our opinion the best of this work has led to increased understanding of intellectual mechanisms and this is essential for the development of artificial intelligence even though few investigators have tried to place their particular mechanism in the general context of artificial intelligence. Sometimes this is because the investigator identifies his particular problem with the field as a whole; he thinks he sees the woods when in fact he is looking at a tree. An old but not yet superseded discussion on intellectual mechanisms is in Minsky (1961); see also Newell's (1965) review of the state of artificial intelligence.

There have been several attempts to design a general intelligence with the same kind of flexibility as that of a human. This has meant different things to different investigators, but none has met with much success even in the sense of general intelligence used by the investigator in question. Since our criticism of this work will be that it does not face the philosophical problems discussed in this paper we shall postpone discussing it until a concluding section.

However, we are obliged at this point to present our notion of general intelligence.

It is not difficult to give sufficient conditions for general intelligence. Turing's idea that the machine should successfully pretend to a sophisticated observer to be a human being for half an hour will do. However, if we direct our efforts towards such a goal our attention is distracted by certain superficial aspects of human behaviour that have to be imitated. Turing excluded some of these by specifying that the human to be imitated is at the end of a teletype line, so that voice, appearance, smell, etc., do not have to be considered. Turing did allow himself to be distracted into discussing the imitation of human fallibility in arithmetic, laziness, and the ability to use the English language.

However, work on artificial intelligence, especially general intelligence, will be improved by a clearer idea of what intelligence is. One way is to give a purely behavioural or black-box definition. In this case we have to say that a machine is intelligent if it solves certain classes of problems requiring intelligence in humans, or survives in an intellectually demanding environment. This definition seems vague; perhaps it can be made somewhat more precise without departing from behavioural terms, but we shall not try to do so.

Instead, we shall use in our definition certain structures apparent to introspection, such as knowledge of facts. The risk is twofold: in the first place we might be mistaken in our introspective views of our own mental structure; we may only think we use facts. In the second place there might be entities which satisfy behaviourist criteria of intelligence but are not organized in this way. However, we regard the construction of intelligent machines as fact manipulators as being the best bet both for constructing artificial intelligence and understanding natural intelligence.

We shall, therefore, be interested in an intelligent entity that is equipped with a representation or model of the world. On the basis of this representation a certain class of internally posed questions can be answered, not always correctly. Such questions are

1. What will happen next in a certain aspect of the situation?
2. What will happen if I do a certain action?
3. What is $3 + 3$?
4. What does he want?
5. Can I figure out how to do this or must I get information from someone else or something else?

The above are not a fully representative set of questions and we do not have such a set yet.

On this basis we shall say that an entity is intelligent if it has an adequate model of the world (including the intellectual world of mathematics, understanding of its own goals and other mental processes), if it is clever enough to answer a wide variety of questions on the basis of this model, if it can get

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

additional information from the external world when required, and can perform such tasks in the external world as its goals demand and its physical abilities permit.

According to this definition intelligence has two parts, which we shall call the epistemological and the heuristic. The epistemological part is the representation of the world in such a form that the solution of problems follows from the facts expressed in the representation. The heuristic part is the mechanism that on the basis of the information solves the problem and decides what to do. Most of the work in artificial intelligence so far can be regarded as devoted to the heuristic part of the problem. This paper, however, is entirely devoted to the epistemological part.

Given this notion of intelligence the following kinds of problems arise in constructing the epistemological part of an artificial intelligence:

1. What kind of general representation of the world will allow the incorporation of specific observations and new scientific laws as they are discovered?
2. Besides the representation of the physical world what other kind of entities have to be provided for? For example, mathematical systems, goals, states of knowledge.
3. How are observations to be used to get knowledge about the world, and how are the other kinds of knowledge to be obtained? In particular what kinds of knowledge about the system's own state of mind are to be provided for?
4. In what kind of internal notation is the system's knowledge to be expressed?

These questions are identical with or at least correspond to some traditional questions of philosophy, especially in metaphysics, epistemology and philosophic logic. Therefore, it is important for the research worker in artificial intelligence to consider what the philosophers have had to say.

Since the philosophers have not really come to an agreement in 2500 years it might seem that artificial intelligence is in a rather hopeless state if it is to depend on getting concrete enough information out of philosophy to write computer programs. Fortunately, merely undertaking to embody the philosophy in a computer program involves making enough philosophical presuppositions to exclude most philosophy as irrelevant. Undertaking to construct a general intelligent computer program seems to entail the following presuppositions:

1. The physical world exists and already contains some intelligent machines called people.
2. Information about this world is obtainable through the senses and is expressible internally.
3. Our common-sense view of the world is approximately correct and so is our scientific view.

4. The right way to think about the general problems of metaphysics and epistemology is not to attempt to clear one's own mind of all knowledge and start with 'Cogito ergo sum' and build up from there. Instead, we propose to use all of our own knowledge to construct a computer program that knows. The correctness of our philosophical system will be tested by numerous comparisons between the beliefs of the program and our own observations and knowledge. (This point of view corresponds to the presently dominant attitude towards the foundations of mathematics. We study the structure of mathematical systems—from the outside as it were—using whatever metamathematical tools seem useful instead of assuming as little as possible and building up axiom by axiom and rule by rule within a system.)
5. We must undertake to construct a rather comprehensive philosophical system, contrary to the present tendency to study problems separately and not try to put the results together.
6. The criterion for definiteness of the system becomes much stronger. Unless, for example, a system of epistemology allows us, at least in principle, to construct a computer program to seek knowledge in accordance with it, it must be rejected as too vague.
7. The problem of 'free will' assumes an acute but concrete form. Namely, in common-sense reasoning, a person often decides what to do by evaluating the results of the different actions he can do. An intelligent program must use this same process, but using an exact formal sense of of *can*, must be able to show that it has these alternatives without denying that it is a deterministic machine.
8. The first task is to define even a naïve, common-sense view of the world precisely enough to program a computer to act accordingly. This is a very difficult task in itself.

We must mention that there is one possible way of getting an artificial intelligence without having to understand it or solve the related philosophical problems. This is to make a computer simulation of natural selection in which intelligence evolves by mutating computer programs in a suitably demanding environment. This method has had no substantial success so far, perhaps due to inadequate models of the world and of the evolutionary process, but it might succeed. It would seem to be a dangerous procedure, for a program that was intelligent in a way its designer did not understand might get out of control. In any case, the approach of trying to make an artificial intelligence through understanding what intelligence is, is more congenial to the present authors and seems likely to succeed sooner.

Reasoning programs and the Missouri program

The philosophical problems that have to be solved will be clearer in connection with a particular kind of proposed intelligent program, called a reasoning program or RP for short. RP interacts with the world through

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

input and output devices some of which may be general sensory and motor organs (for example, television cameras, microphones, artificial arms) and others of which are communication devices (for example, teletypes or keyboard-display consoles). Internally, RP may represent information in a variety of ways. For example, pictures may be represented as dot arrays or a lists of regions and edges with classifications and adjacency relations. Scenes may be represented as lists of bodies with positions, shapes, and rates of motion. Situations may be represented by symbolic expressions with allowed rules of transformation. Utterances may be represented by digitized functions of time, by sequences of phonemes, and parsings of sentences.

However, one representation plays a dominant role and in simpler systems may be the only representation present. This is a representation by sets of sentences in a suitable formal logical language, for example w -order logic with function symbols, description operator, conditional expressions, sets, etc. Whether we must include modal operators with their referential opacity is undecided. This representation dominates in the following sense:

1. All other data structures have linguistic descriptions that give the relations between the structures and what they tell about the world.
2. The subroutines have linguistic descriptions that tell what they do, either internally manipulating data, or externally manipulating the world.
3. The rules that express RP's beliefs about how the world behaves and that give the consequences of strategies are expressed linguistically.
4. RP's goals, as given by the experimenter, its devised subgoals, its opinion on its state of progress are all linguistically expressed.
5. We shall say that RP's information is adequate to solve a problem if it is a logical consequence of all these sentences that a certain strategy of action will solve it.
6. RP is a deduction program that tries to find strategies of action that it can prove will solve a problem; on finding one, it executes it.
7. Strategies may involve subgoals which are to be solved by RP, and part or all of a strategy may be purely intellectual, that is, may involve the search for a strategy, a proof, or some other intellectual object that satisfies some criteria.

Such a program was first discussed in McCarthy (1959) and was called the Advice Taker. In McCarthy (1963) a preliminary approach to the required formalism, now superseded by this paper, was presented. This paper is in part an answer to Y. Bar-Hillel's comment, when the original paper was presented at the 1958 Symposium on the Mechanization of Thought Processes, that the paper involved some philosophical presuppositions.

Constructing RP involves both the epistemological and the heuristic parts of the artificial intelligence problem: that is, the information in memory must be adequate to determine a strategy for achieving the goal (this strategy

may involve the acquisition of further information) and RP must be clever enough to find the strategy and the proof of its correctness. Of course, these problems interact, but since this paper is focused on the epistemological part, we mention the Missouri program (MP) that involves only this part.

The Missouri program (its motto is, 'Show me') does not try to find strategies or proofs that the strategies achieve a goal. Instead, it allows the experimenter to present its proof steps and checks their correctness. Moreover, when it is 'convinced' that it ought to perform an action or execute a strategy it does so. We may regard this paper as being concerned with the construction of a Missouri program that can be persuaded to achieve goals.

Representations of the world

The first step in the design of RP or MP is to decide what structure the world is to be regarded as having, and how information about the world and its laws of change are to be represented in the machine. This decision turns out to depend on whether one is talking about the expression of general laws or specific facts. Thus, our understanding of gas dynamics depends on the representation of a gas as a very large number of particles moving in space; this representation plays an essential rôle in deriving the mechanical, thermal electrical and optical properties of gases. The state of the gas at a given instant is regarded as determined by the position, velocity and excitation states of each particle. However, we never actually determine the position, velocity or excitation of even a single molecule. Our practical knowledge of a particular sample of gas is expressed by parameters like the pressure, temperature and velocity fields or even more grossly by average pressures and temperatures. From our philosophical point of view this is entirely normal, and we are not inclined to deny existence to entities we cannot see, or to be so anthropocentric as to imagine that the world must be so constructed that we have direct or even indirect access to all of it.

From the artificial intelligence point of view we can then define three kinds of adequacy for representations of the world.

A representation is called metaphysically adequate if the world could have that form without contradicting the facts of the aspect of reality that interests us. Examples of metaphysically adequate representations for different aspects of reality are:

1. The representation of the world as a collection of particles interacting through forces between each pair of particles.
2. Representation of the world as a giant quantum-mechanical wave function.
3. Representation as a system of interacting discrete automata. We shall make use of this representation.

Metaphysically adequate representations are mainly useful for constructing general theories. Deriving observable consequences from the theory is a further step.

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

A representation is called epistemologically adequate for a person or machine if it can be used practically to express the facts that one actually has about the aspect of the world. Thus none of the above-mentioned representations are adequate to express facts like 'John is at home' or 'dogs chase cats' or 'John's telephone number is 321-7580'. Ordinary language is obviously adequate to express the facts that people communicate to each other in ordinary language. It is not, for instance, adequate to express what people know about how to recognize a particular face. The second part of this paper is concerned with an epistemologically adequate formal representation of common-sense facts of causality, ability and knowledge.

A representation is called heuristically adequate if the reasoning processes actually gone through in solving a problem are expressible in the language. We shall not treat this somewhat tentatively proposed concept further in this paper except to point out later that one particular representation seemsepistemologically but not heuristically adequate.

In the remaining sections of the first part of the paper we shall use the representations of the world as a system of interacting automata to explicate notions of causality, ability and knowledge (including self-knowledge).

The automaton representation and the notion of 'can'

Let S be a system of interacting discrete finite automata such as that shown in figure 1

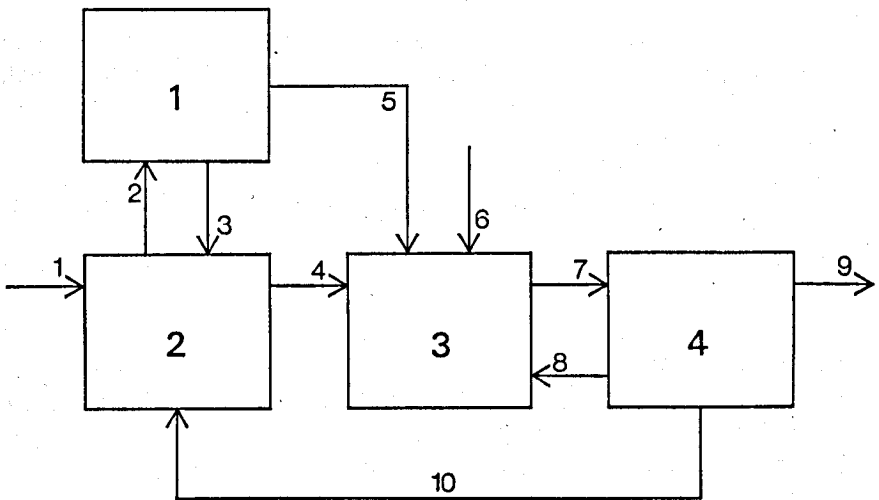


Figure 1

Each box represents a subautomaton and each line represents a signal. Time takes on integer values and the dynamic behaviour of the whole automaton is given by the equations:

$$(1) \begin{aligned} a_1(t+1) &= A_1(a_1(t), s_3(t)) \\ a_2(t+1) &= A_2(a_2(t), s_1(t), s_2(t), s_{10}(t)) \\ a_3(t+1) &= A_3(a_3(t), s_4(t), s_5(t), s_6(t)) \\ a_4(t+1) &= A_4(a_4(t), s_7(t)) \end{aligned}$$

$$(2) \begin{aligned} s_2(t) &= S_2(a_1(t)) \\ s_3(t) &= S_3(a_2(t)) \\ s_4(t) &= S_4(a_2(t)) \\ s_5(t) &= S_5(a_1(t)) \\ s_7(t) &= S_7(a_4(t)) \\ s_8(t) &= S_8(a_4(t)) \\ s_9(t) &= S_9(a_4(t)) \\ s_{10}(t) &= S_{10}(a_4(t)) \end{aligned}$$

The interpretation of these equations is that the state of any automaton at time $t+1$ is determined by its state at time t and by the signals received at time t . The value of a particular signal at time t is determined by the state at time t of the automaton from which it comes. Signals without a source automaton represent inputs from the outside and signals without a destination represent outputs.

Finite automata are the simplest examples of systems that interact over time. They are completely deterministic; if we know the initial states of all the automata and if we know the inputs as a function of time, the behaviour of the system is completely determined by equations (1) and (2) for all future time.

The automaton representation consists in regarding the world as a system of interacting subautomata. For example, we might regard each person in the room as a subautomaton and the environment as consisting of one or more additional subautomata. As we shall see, this representation has many of the qualitative properties of interactions among things and persons. However, if we take the representation too seriously and attempt to represent particular situations by systems of interacting automata we encounter the following difficulties:

1. The number of states required in the subautomata is very large, for example $2^{10^{10}}$, if we try to represent someone's knowledge. Automata this large have to be represented by computer programs, or in some other way that does not involve mentioning states individually.
2. Geometric information is hard to represent. Consider, for example, the location of a multi-jointed object such as a person or a matter of even more difficulty – the shape of a lump of clay.
3. The system of fixed interconnections is inadequate. Since a person may handle any object in the room, an adequate automaton representation would require signal lines connecting him with every object.
4. The most serious objection, however, is that (in our terminology) the automaton representation is epistemologically inadequate. Namely, we

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

do not ever know a person well enough to list his internal states. The kind of information we do have about him needs to be expressed in some other way.

Nevertheless, we may use the automaton representation for concepts of *can*, *causes*, some kinds of counterfactual statements ('If I had struck this match yesterday it would have lit') and, with some elaboration of the representation, for a concept of *believes*.

Let us consider the notion of *can*. Let S be a system of subautomata without external inputs such as that of figure 2. Let p be one of the subautomata, and suppose that there are m signal lines coming out of p . What p can do is defined in terms of a new system S_p , which is obtained from the system S by disconnecting the m signal lines coming from p and replacing them by m external input lines to the system. In figure 2, subautomaton 1 has one output, and in the system S_1 this is replaced by an external input. The new system S_p always has the same set of states as the system S . Now let π be a condition on the state such as, ' a_2 is even' or ' $a_2 = a_3$ '. (In the applications π may be a condition like 'The box is under the bananas'.)

We shall write

$$can(p, \pi, s)$$

which is read, 'The subautomaton p can bring about the condition π in the situation s ' if there is a sequence of outputs from the automaton S_p that will eventually put S into a state a' that satisfies $\pi(a')$. In other words, in determining what p can achieve, we consider the effects of sequences of its actions, quite apart from the conditions that determine what it actually will do.

In figure 2, let us consider the initial state a to be one in which all subautomata are initially in state 0. Then the reader will easily verify the following propositions:

1. Subautomaton 2 will never be in state 1.
2. Subautomaton 1 can put subautomaton 2 in state 1.
3. Subautomaton 3 cannot put subautomaton 2 in state 1.

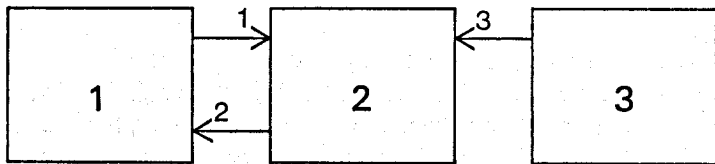


Figure 2. System S

$$a_1(t+1) = a_1(t) + s_2(t)$$

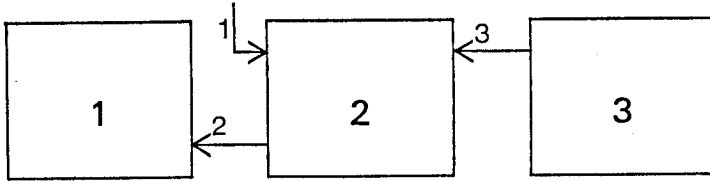
$$a_2(t+1) = a_2(t) + s_1(t) + 2s_3(t)$$

$$a_3(t+1) = \text{if } a_3(t) = 0 \text{ then } 0 \text{ else } a_3(t) + 1$$

$$s_1(t) = \text{if } a_1(t) = 0 \text{ then } 2 \text{ else } 1$$

$$s_2(t) = 1$$

$$s_3(t) = \text{if } a_3(t) = 0 \text{ then } 0 \text{ else } 1$$

System S_1

We claim that this notion of *can* is, to a first approximation, the appropriate one for an automaton to use internally in deciding what to do by reasoning. We also claim that it corresponds in many cases to the common sense notion of *can* used in everyday speech.

In the first place, suppose we have an automaton that decides what to do by reasoning, for example suppose it is a computer using an RP. Then its output is determined by the decisions it makes in the reasoning process. It does not know (has not computed) in advance what it will do, and, therefore, it is appropriate that it considers that it can do anything that can be achieved by some sequence of its outputs. Common-sense reasoning seems to operate in the same way.

The above rather simple notion of *can* requires some elaboration both to represent adequately the commonsense notion and for practical purposes in the reasoning program.

First, suppose that the system of automata admits external inputs. There are two ways of defining *can* in this case. One way is to assert $\text{can}(p, \pi, s)$ if p can achieve π regardless of what signals appear on the external inputs. Thus, instead of requiring the existence of a sequence of outputs of p that achieves the goal we shall require the existence of a strategy where the output at any time is allowed to depend on the sequence of external inputs so far received by the system. Note that in this definition of *can* we are not requiring that p have any way of knowing what the external inputs were. An alternative definition requires the outputs to depend on the inputs of p . This is equivalent to saying that p can achieve a goal provided the goal would be achieved for arbitrary inputs by some automaton put in place of p . With either of these definitions *can* becomes a function of the place of the subautomaton in the system rather than of the subautomaton itself. We do not know which of these treatments is preferable, and so we shall call the first concept *cana* and the second *canb*.

The idea that what a person can do depends on his position rather than on

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

his characteristics is somewhat counter-intuitive. This impression can be mitigated as follows: Imagine the person to be made up of several sub-automata; the output of the outer subautomaton is the motion of the joints. If we break the connection to the world at that point we can answer questions like, 'Can he fit through a given hole?' We shall get some counter-intuitive answers, however, such as that he can run at top speed for an hour or can jump over a building, since there are sequences of motions of his joints that would achieve these results.

The next step, however, is to consider a subautomaton that receives the nerve impulses from the spinal cord and transmits them to the muscles. If we break at the input to this automaton, we shall no longer say that he can jump over a building or run long at top speed since the limitations of the muscles will be taken into account. We shall, however, say that he can ride a unicycle since appropriate nerve signals would achieve this result.

The notion of *can* corresponding to the intuitive notion in the largest number of cases might be obtained by hypothesizing an 'organ of will', which makes decisions to do things and transmits these decisions to the main part of the brain that tries to carry them out and contains all the knowledge of particular facts. If we make the break at this point we shall be able to say that so-and-so cannot dial the President's secret and private telephone number because he does not know it, even though if the question were asked could he dial that particular number, the answer would be yes. However, even this break would not give the statement, 'I cannot go without saying goodbye, because this would hurt the child's feelings'.

On the basis of these examples, one might try to postulate a sequence of narrower and narrower notions of *can* terminating in a notion according to which a person can do only what he actually does. This notion would then be superfluous. Actually, one should not look for a single best notion of *can*; each of the above-mentioned notions is useful and is actually used in some circumstances. Sometimes, more than one notion is used in a single sentence, when two different levels of constraint are mentioned.

Besides its use in explicating the notion of *can*, the automaton representation of the world is very suited for defining notions of causality. For, we may say that subautomaton p caused the condition π in state s , if changing the output of p would prevent π . In fact the whole idea of a system of interacting automata is just a formalization of the commonsense notion of causality.

Moreover, the automaton representation can be used to explicate certain counterfactual conditional sentences. For example, we have the sentence, 'If I had struck this match yesterday at this time it would have lit'. In a suitable automaton representation, we have a certain state of the system yesterday at that time, and we imagine a break made where the nerves lead from my head or perhaps at the output of my 'decision box', and the appropriate signals to strike the match having been made. Then it is a definite and decidable question about the system S_p , whether the match lights or not,

depending on whether it is wet, etc. This interpretation of this kind of counterfactual sentence seems to be what is needed for RP to learn from its mistakes, by accepting or generating sentences of the form, 'had I done thus-and-so I would have been successful, so I should alter my procedures in some way that would have produced the correct action in that case'.

In the foregoing we have taken the representation of the situation as a system of interacting subautomata for granted. However, a given overall situation might be represented as a system of interacting subautomata in a number of ways, and different representations might yield different results about what a given subautomaton can achieve, what would have happened if some subautomaton had acted differently, or what caused what. Indeed, in a different representation, the same or corresponding subautomata might not be identifiable. Therefore, these notions depend on the representation chosen.

For example, suppose a pair of Martians observe the situation in a room. One Martian analyses it as a collection of interacting people as we do, but the second Martian groups all the heads together into one subautomaton and all the bodies into another. (A creature from momentum space would regard the Fourier components of the distribution of matter as the separate interacting subautomata.) How is the first Martian to convince the second that his representation is to be preferred? Roughly speaking, he would argue that the interaction between the heads and bodies of the same person is closer than the interaction between the different heads, and so more of an analysis has been achieved from 'the primordial muddle' with the conventional representation. He will be especially convincing when he points out that when the meeting is over the heads will stop interacting with each other, but will continue to interact with their respective bodies.

We can express this kind of argument formally in terms of automata as follows: Suppose we have an autonomous automaton A , that is an automaton without inputs, and let it have k states. Further, let m and n be two integers such that $m, n \geq k$. Now label k points of an m -by- n array with the states of A . This can be done in $\binom{mn}{k}!$ ways. For each of these ways we have a representation of the automaton A as a system of an m -state automaton B interacting with an n -state automaton C . Namely, corresponding to each row of the array we have a state of B and to each column a state of C . The signals are in 1-1 correspondence with the states themselves; thus each subautomaton has just as many values of its output as it has states. Now it may happen that two of these signals are equivalent in their effect on the other subautomaton, and we use this equivalence relation to form equivalence classes of signals. We may then regard the equivalence classes as the signals themselves. Suppose then that there are now r signals from B to C and s signals from C to B . We ask how small r and s can be taken in general compared to m and n . The answer may be obtained by counting the number of inequivalent automata with k states and comparing it with the number of systems of two automata

with m and n states respectively and r and s signals going in the respective directions. The result is not worth working out in detail, but tells us that only a few of the k state automata admit such a decomposition with r and s small compared to m and n . Therefore, if an automaton happens to admit such a decomposition it is very unusual for it to admit a second such decomposition that is not equivalent to the first with respect to some renaming of states. Applying this argument to the real world, we may say that it is overwhelmingly probable that our customary decomposition of the world automaton into separate people and things has a unique, objective and usually preferred status. Therefore, the notions of *can*, of causality, and of counterfactual associated with this decomposition also have a preferred status.

In our opinion, this explains some of the difficulty philosophers have had in analysing counterfactuals and causality. For example, the sentence, 'If I had struck this match yesterday, it would have lit' is meaningful only in terms of a rather complicated model of the world, which, however, has an objective preferred status. However, the preferred status of this model depends on its correspondence with a large number of facts. For this reason, it is probably not fruitful to treat an individual counterfactual conditional sentence in isolation.

It is also possible to treat notions of belief and knowledge in terms of the automaton representation. We have not worked this out very far, and the ideas presented here should be regarded as tentative. We would like to be able to give conditions under which we may say that a subautomaton p believes a certain proposition. We shall not try to do this directly but only relative to a predicate $B_p(s, w)$. Here s is the state of the automaton p and w is a proposition; $B_p(s, w)$ is true if p is to be regarded as believing w when in state s and is false otherwise. With respect to such a predicate B we may ask the following questions:

1. Are p 's beliefs consistent? Are they correct?
2. Does p reason? That is, do new beliefs arise that are logical consequences of previous beliefs?
3. Does p observe? That is, do true propositions about automata connected to p cause p to believe them?
4. Does p behave rationally? That is, when p believes a sentence asserting that it should do something, does p do it?
5. Does p communicate in language L ? That is, regarding the content of a certain input or output signal line as a text in language L , does this line transmit beliefs to or from p ?
6. Is p self-conscious? That is, does it have a fair variety of correct beliefs about its own beliefs and the processes that change them?

It is only with respect to the predicate B_p that all these questions can be asked. However, if questions 1 thru 4 are answered affirmatively for some predicate B_p , this is certainly remarkable, and we would feel fully entitled to consider B_p a reasonable notion of belief.

In one important respect the situation with regard to belief or knowledge is the same as it was for counterfactual conditional statements: no way is provided to assign a meaning to a single statement of belief or knowledge, since for any single statement a suitable B_p can easily be constructed. Individual statements about belief or knowledge are made on the basis of a larger system which must be validated as a whole.

2. FORMALISM

In part 1 we showed how the concepts of ability and belief could be given formal definition in the metaphysically adequate automaton model and indicated the correspondence between these formal concepts and the corresponding commonsense concepts. We emphasized, however, that practical systems require epistemologically adequate systems in which those facts which are actually ascertainable can be expressed.

In this part we begin the construction of an epistemologically adequate system. Instead of giving formal definitions, however, we shall introduce the formal notions by informal natural-language descriptions and give examples of their use to describe situations and the possibilities for action they present. The formalism presented is intended to supersede that of McCarthy (1963).

Situations

A situation s is the complete state of the universe at an instant of time. We denote by Sit the set of all situations. Since the universe is too large for complete description, we shall never completely describe a situation; we shall only give facts about situations. These facts will be used to deduce further facts about that situation, about future situations and about situations that persons can bring about from that situation.

This requires that we consider not only situations that actually occur, but also hypothetical situations such as the situation that would arise if Mr Smith sold his car to a certain person who has offered \$250 for it. Since he is not going to sell the car for that price, the hypothetical situation is not completely defined; for example, it is not determined what Smith's mental state would be and therefore it is also undetermined how quickly he would return to his office, etc. Nevertheless, the representation of reality is adequate to determine some facts about this situation, enough at least to make him decide not to sell the car.

We shall further assume that the laws of motion determine, given a situation, all future situations.*

In order to give partial information about situations we introduce the notion of fluent.

* This assumption is difficult to reconcile with quantum mechanics, and relativity tells us that any assignment of simultaneity to events in different places is arbitrary. However, we are proceeding on the basis that modern physics is irrelevant to common sense in deciding what to do, and in particular is irrelevant to solving the 'free will problem'.

Fluents

A *fluent* is a function whose domain is the space *Sit* of situations. If the range of the function is (*true*, *false*), then it is called a *propositional fluent*. If its range is *Sit*, then it is called a *situational fluent*.

Fluents are often the values of functions. Thus *raining*(*x*) is a fluent such that *raining*(*x*)(*s*) is true if and only if it is raining at the place *x* in the situation *s*. We can also write this assertion as *raining*(*x,s*) making use of the well-known equivalence between a function of two variables and a function of the first variable whose value is a function of the second variable.

Suppose we wish to assert about a situation *s* that person *p* is in place *x* and that it is raining in place *x*. We may write this in several ways each of which has its uses:

1. $at(p,x)(s) \wedge raining(x)(s)$. This corresponds to the definition given.
2. $at(p,x,s) \wedge raining(x,s)$. This is more conventional mathematically and a bit shorter.
3. $[at(p,x) \wedge raining(x)](s)$. Here we are introducing a convention that operators applied to fluents give fluents whose values are computed by applying the logical operators to the values of the operand fluents, that is, if *f* and *g* are fluents then

$$(f \text{ op } g)(s) = f(s) \text{ op } g(s)$$

4. $[\lambda s'. at(p,x,s') \wedge raining(x,s')](s)$. Here we have formed the composite fluent by λ -abstraction.

Here are some examples of fluents and expressions involving them:

1. *time*(*s*). This is the time associated with the situation *s*. It is essential to consider time as dependent on the situation as we shall sometimes wish to consider several different situations having the same time value, for example, the results of alternative courses of actions.
2. *in*(*x,y,s*). This asserts that *x* is in the location *y* in situation *s*. The fluent *in* may be taken as satisfying a kind of transitive law, namely:

$$\forall x . \forall y . \forall z . \forall s . in(x,y,s) \wedge in(y,z,s) \supset in(x,z,s)$$

We can also write this law

$$\forall x . \forall y . \forall z . \forall . in(x,y) \wedge in(y,z) \supset in(x,z)$$

where we have adopted the convention that a quantifier without a variable is applied to an implicit situation variable which is the (suppressed) argument of a propositional fluent that follows. Suppressing situation arguments in this way corresponds to the natural language convention of writing sentences like, 'John was at home' or 'John is at home' leaving understood the situations to which these assertions apply.

3. *has*(*Monkey,Bananas,s*). Here we introduce the convention that capitalized words denote proper names, for example, 'Monkey' is the

name of a particular individual. That the individual is a monkey is not asserted, so that the expression *monkey(Monkey)* may have to appear among the premisses of an argument. Needless to say, the reader has a right to feel that he has been given a hint that the individual *Monkey* will turn out to be a monkey. The above expression is to be taken as asserting that in the situation *s* the individual *Monkey* has the object *Bananas*. We shall, in the examples below, sometimes omit premisses such as *monkey(Monkey)*, but in a complete system they would have to appear.

Causality

We shall make assertions of causality by means of a fluent $F(\pi)$ where π is itself a propositional fluent. $F(\pi, s)$ asserts that the situation *s* will be followed (after an unspecified time) by a situation that satisfies the fluent π .

We may use F to assert that if a person is out in the rain he will get wet, by writing:

$$\forall x . \forall p . \forall s . \text{raining}(x, s) \wedge \text{at}(p, x, s) \wedge \text{outside}(p, s) \supset F(\lambda s' . \text{wet}(p, s'), s)$$

Suppressing explicit mention of situations gives:

$$\forall x . \forall p . \forall . \text{raining}(x) \wedge \text{at}(p, x) \wedge \bar{\text{outside}}(p) \supset F(\text{wet}(p)).$$

In this case suppressing situations simplifies the statement.

F can also be used to express physical laws. Consider the law of falling bodies which is often written

$$h = h_0 + v_0 \cdot (t - t_0) - \frac{1}{2}g \cdot (t - t_0)^2$$

together with some prose identifying the variables. Since we need a formal system for machine reasoning we cannot have any prose. Therefore, we write:

$$\forall b . \forall t . \forall s . \text{falling}(b, s) \wedge t \geq 0 \wedge \text{height}(b, s) + \text{velocity}(b, s) \cdot t - \frac{1}{2}gt^2 > 0$$

$$\supset F(\lambda s' . \text{time}(s') = \text{time}(s) + t \wedge \text{falling}(b, s') \wedge \text{height}(b, s') = \text{height}(b, s) + \text{velocity}(b, s) \cdot t - \frac{1}{2}gt^2, s)$$

Suppressing explicit mention of situations in this case requires the introduction of real auxiliary quantities v , h and τ so that the sentence takes the following form

$$\forall b . \forall t . \forall \tau . \forall v . \forall h .$$

$$\text{falling}(b) \wedge t \geq 0 \wedge h = \text{height}(b) \wedge v = \text{velocity}(b) \wedge h + vt - \frac{1}{2}gt^2 > 0 \wedge \text{time} = \tau \supset F(\text{time} = t + \tau \wedge \text{falling}(b) \wedge \text{height} = h + vt - \frac{1}{2}gt^2)$$

There has to be a convention (or declarations) so that it is determined that $\text{height}(b)$, $\text{velocity}(b)$ and time are fluents, whereas t , v , τ and h denote ordinary real numbers.

$F(\pi, s)$ as introduced here corresponds to A.N.Prior's (1957, 1968) expression $F\pi$.

The use of situation variables is analogous to the use of time-instants in the calculi of world-states which Prior (1968) calls *U-T* calculi. Prior provides many interesting correspondences between his *U-T* calculi and various axiomatizations of the modal tense-logics (that is, using this *F*-operator: see part 4). However, the situation calculus is richer than any of the tense-logics Prior considers.

Besides *F* he introduces three other operators which we also find useful; we thus have:

1. $F(\pi, s)$. For some situation s' in the future of s , $\pi(s')$ holds.
2. $G(\pi, s)$. For all situations s' in the future of s , $\pi(s')$ holds.
3. $P(\pi, s)$. For some situations s' in the past of s , $\pi(s')$ holds.
4. $H(\pi, s)$. For all situations s' in the past of s , $\pi(s')$ holds.

It seems also useful to define a situational fluent $next(\pi)$ as the next situation s' in the future of s for which $\pi(s')$ holds. If there is no such situation, that is, if $\neg F(\pi, s)$, then $next(\pi, s)$ is considered undefined. For example, we may translate the sentence 'By the time John gets home, Henry will be home too' as $at(Henry, home(Henry), next(at(John, home(John)), s))$. Also the phrase 'when John gets home' translates into $time(next(at(John, home(John)), s))$.

Though $next(\pi, s)$ will never actually be computed since situations are too rich to be specified completely, the values of fluents applied to $next(\pi, s)$ will be computed.

Actions

A fundamental rôle in our study of actions is played by the situational fluent $result(p, \sigma, s)$

Here, p is a person, σ is an action or more generally a strategy, and s is a situation. The value of $result(p, \sigma, s)$ is the situation that results when p carries out σ , starting in the situation s . If the action or strategy does not terminate, $result(p, \sigma, s)$ is considered undefined.

With the aid of $result$ we can express certain laws of ability. For example:

$$has(p, k, s) \wedge fits(k, sf) \wedge at(p, sf, s) \supset open(sf, result(p, opens(sf, k), s))$$

This formula is to be regarded as an axiom schema asserting that if in a situation s a person p has a key k that fits the safe sf , then in the situation resulting from his performing the action $opens(sf, k)$, that is, opening the safe sf with the key k , the safe is open. The assertion $fits(k, sf)$ carries the information that k is a key and sf a safe. Later we shall be concerned with combination safes that require p to *know* the combination.

Strategies

Actions can be combined into strategies. The simplest combination is a finite sequence of actions. We shall combine actions as though they were

ALGOL statements, that is, procedure calls. Thus, the sequence of actions, ('move the box under the bananas', 'climb onto the box', and 'reach for the bananas') may be written:

```
begin move(Box, Under-Bananas); climb(Box); reach-for(Bananas) end;
```

A strategy in general will be an ALGOL-like compound statement containing actions written in the form of procedure calling assignment statements, and conditional **go to**'s. We shall not include any declarations in the program since they can be included in the much larger collection of declarative sentences that determine the effect of the strategy.

Consider for example the strategy that consists of walking 17 blocks south, turning right and then walking till you come to Chestnut Street. This strategy may be written as follows:

```
begin
  face(South);
  n:=0;
b: if n=17 then go to a;
     walk-a-block, n:=n+1;
     go to b;
a: turn-right;
c: walk-a-block;
     if name-on-street-sign ≠ 'Chestnut Street' then go to c
end;
```

In the above program the external actions are represented by procedure calls. Variables to which values are assigned have a purely internal significance (we may even call it mental significance) and so do the statement labels and the **go to** statements.

For the purpose of applying the mathematical theory of computation we shall write the program differently: namely, each occurrence of an action α is to be replaced by an assignment statement $s := result(p, \alpha, s)$. Thus the above program becomes

```
begin
  s:=result(p, face(South), s);
  n:=0;
b: if n=17 then go to a;
     s:=result(p, walk-a-block, s);
     n:=n+1;
     go to b;
a: s:=result(p, turn-right, s);
c: s:=result(p, walk-a-block, s);
     if name-on-street-sign(s) ≠ 'Chestnut Street' then go to c.
end;
```

Suppose we wish to show that by carrying out this strategy John can go home provided he is initially at his office. Then according to the methods of Zohar

Manna (1968a, 1968b), we may derive from this program together with the initial condition $at(John, office(John), s_0)$ and the final condition $at(John, home(John), s)$, a sentence W of first-order logic. Proving W will show that the procedure terminates in a finite number of steps and that when it terminates s will satisfy $at(John, home(John), s)$.

According to Manna's theory we must prove the following collection of sentences inconsistent for arbitrary interpretations of the predicates q_1 and q_2 and the particular interpretations of the other functions and predicates in the program:

$$\begin{aligned}
 & at(John, office(John), s_0), \\
 & q_1(0, result(John, face(South), s_0)), \\
 & \forall n . \forall s . q_1(n, s) \supset \text{if } n = 17 \\
 & \qquad \text{then } q_2(result(John, walk-a-block, result(John, turn-right, \\
 & \qquad \qquad \qquad s))) \\
 & \qquad \text{else } q_1(n + 1, result(John, walk-a-block, s)), \\
 & \forall s . q_2(s) \supset \text{if name-on-street-sign}(s) \neq \text{'Chestnut Street'} \\
 & \qquad \text{then } q_2(result(John, walk-a-block, s)) \\
 & \qquad \text{else } \neg at(John, home(John), s)
 \end{aligned}$$

Therefore the formula that has to be proved may be written

$$\begin{aligned}
 & \exists s_0 \{ at(John, office(John), s_0) \wedge q_1(0, result(John, face(South), s_0)) \} \\
 & \qquad \supset \\
 & \exists n . \exists s . \{ q_1(n, s) \wedge \text{if } n = 17 \\
 & \qquad \text{then } \wedge q_2(result(John, walk-a-block, result(John, turn- \\
 & \qquad \qquad \qquad right, s))) \\
 & \qquad \text{else } \neg q_1(n + 1, result(John, walk-a-block, s)) \} \\
 & \qquad \vee \\
 & \exists s . \{ q_2(s) \wedge \text{if name-on-street-sign}(s) \neq \text{'Chestnut Street'} \\
 & \qquad \text{then } \neg q_2(result(John, walk-a-block, s)) \\
 & \qquad \text{else } at(John, home(John), s) \}
 \end{aligned}$$

In order to prove this sentence we would have to use the following kinds of facts expressed as sentences or sentence schemas of first-order logic:

1. Facts of geography. The initial street stretches at least 17 blocks to the south, and intersects a street which in turn intersects Chestnut Street a number of blocks to the right; the location of John's home and office.
2. The fact that the fluent name-on-street-sign will have the value 'Chestnut Street' at that point.
3. Facts giving the effects of action α expressed as predicates about $result(p, \alpha, s)$ deducible from sentences about s .
4. An axiom schema of induction that allows us to deduce that the loop of walking 17 blocks will terminate.

5. A fact that says that Chestnut Street is a finite number of blocks to the right after going 17 blocks south. This fact has nothing to do with the possibility of walking. It may also have to be expressed as a sentence schema or even as a sentence of second-order logic.

When we consider making a computer carry out the strategy, we must distinguish the variable s from the other variables in the second form of the program. The other variables are stored in the memory of the computer and the assignments may be executed in the normal way. The variable s represents the state of the world and the computer makes an assignment to it by performing an action. Likewise the fluent name-on-street-sign requires an action, of observation.

Knowledge and ability

In order to discuss the rôle of knowledge in one's ability to achieve goals let us return to the example of the safe. There we had

$$1. \text{has}(p,k,s) \wedge \text{fits}(k,sf) \wedge \text{at}(p,sf,s) \supset \text{open}(sf, \text{result}(p, \text{opens}(sf,k), s)),$$

which expressed sufficient conditions for the ability of a person to open a safe with a key. Now suppose we have a combination safe with a combination c . Then we may write:

$$2. \text{fits2}(c,sf) \wedge \text{at}(p,sf,s) \supset \text{open}(sf, \text{result}(p, \text{opens2}(sf,c), s)),$$

where we have used the predicate *fits2* and the action *opens2* to express the distinction between a key fitting a safe and a combination fitting it, and also the distinction between the acts of opening a safe with a key and a combination. In particular, *opens2*(sf,c) is the act of manipulating the safe in accordance with the combination c . We have left out a sentence of the form *has2*(p,c,s) for two reasons. In the first place it is unnecessary: if you manipulate a safe in accordance with its combination it will open; there is no need to have anything. In the second place it is not clear what *has2*(p,c,s) means. Suppose, for example, that the combination of a particular safe sf is the number 34125, then *fits*(34125, sf) makes sense and so does the act *opens2*($sf, 34125$). (We assume that *open*($sf, \text{result}(p, \text{opens2}(sf, 34111), s)$) would not be true.) But what could *has*($p, 34125, s$) mean? Thus, a direct parallel between the rules for opening a safe with a key and opening it with a combination seems impossible.

Nevertheless, we need some way of expressing the fact that one has to know the combination of a safe in order to open it. First we introduce the function *combination*(sf) and rewrite 2 as

$$3. \text{at}(p,sf,s) \wedge \text{csafe}(sf) \supset \text{open}(sf, \text{result}(p, \text{opens2}(sf, \text{combination}(sf)), s))$$

where *csafe*(sf) asserts that sf is a combination safe and *combination*(sf) denotes the combination of sf . (We could not write *key*(sf) in the other case unless we wished to restrict ourselves to the case of safes with only one key.)

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

Next we introduce the notion of a feasible strategy for a person. The idea is that a strategy that would achieve a certain goal might not be feasible for a person because he lacks certain knowledge or abilities.

Our first approach is to regard the action $opens2(sf, combination(sf))$ as infeasible because p might not know the combination. Therefore, we introduce a new function $idea-of-combination(p, sf, s)$ which stands for person p 's idea of the combination of sf in situation s . The action $opens2(sf, idea-of-combination(p, sf, s))$ is regarded as feasible for p , since p is assumed to know his idea of the combination if this is defined. However, we leave sentence 3 as it is so we cannot yet prove $open(sf, result(p, opens2(sf, idea-of-combination(p, sf, s)), s))$. The assertion that p knows the combination of sf can now be expressed as

$$5. \text{idea-of-combination}(p, sf, s) = \text{combination}(sf)$$

and with this, the possibility of opening the safe can be proved.

Another example of this approach is given by the following formalization of getting into conversation with someone by looking up his number in the telephone book and then dialling it.

The strategy for p in the first form is

```
begin
  lookup(q, Phone-book);
  dial(idea-of-phone-number(q, p))
end;
```

or in the second form

```
begin
  s := result(p, lookup(q, Phone-book), s0);
  s := result(p, dial(idea-of-phone-number(q, p, s)), s)
end;
```

The premisses to write down appear to be

1. $has(p, Phone-book, s_0)$
2. $listed(q, Phone-book, s_0)$
3. $\forall s . \forall p . \forall q . has(p, Phone-book, s) \wedge listed(q, Phone-book, s) \supset$
 $phone-number(q) = idea-of-phone-number(p, q, result(p, lookup(q, Phone-$
 $book), s))$
4. $\forall s . \forall p . \forall q . \forall x . at(q, home(q), s) \wedge has(p, x, s) \wedge telephone(x) \supset$
 $in-conversation(p, q, result(p, dial(phone-number(q)), s))$
5. $at(q, home(q), s_0)$
6. $telephone(Telephone)$
7. $has(p, Telephone, s_0)$

Unfortunately, these premisses are not sufficient to allow one to conclude that $in-conversation(p, q, result(p, begin \text{lookup}(q, Phone-book); \text{dial}(idea-of-phone-number(q, p)) \text{end};, s_0))$.

The trouble is that one cannot show that the fluents $at(q, home(q))$ and $has(p, Telephone)$ still apply to the situation $result(p, lookup(q, Phone-book), s_0)$. To make it come out right we shall revise the third hypothesis to read:

$$\forall s . \forall p . \forall q . \forall x . \forall y . at(q, y, s) \wedge has(p, x, s) \wedge has(p, Phone-book, s) \wedge listed(q, Phone-book) \Rightarrow [\lambda r. at(q, y, r) \wedge has(p, x, r) \wedge phone-number(q) = idea-of-phone-number(p, q, r)] (result(p, lookup(q, Phone-book), s)).$$

This works, but the additional hypotheses about what remains unchanged when p looks up a telephone number are quite *ad hoc*. We shall treat this problem in a later section.

The present approach has a major technical advantage for which, however, we pay a high price. The advantage is that we preserve the ability to replace any expression by an equal one in any expression of our language. Thus if $phone-number(John) = 3217580$, any true statement of our language that contains 3217580 or $phone-number(John)$ will remain true if we replace one by the other. This desirable property is termed referential transparency.

The price we pay for referential transparency is that we have to introduce $idea-of-phone-number(p, q, s)$ as a separate *ad hoc* entity and cannot use the more natural $idea-of(p, phone-number(q), s)$ where $idea-of(p, \phi, s)$ is some kind of operator applicable to the concept ϕ . Namely, the sentence $idea-of(p, phone-number(q), s) = phone-number(q)$ would be supposed to express that p knows q 's phone-number, but $idea-of(p, 3217580, s) = 3217580$ expresses only that p understands that number. Yet with transparency and the fact that $phone-number(q) = 3217580$ we could derive the former statement from the latter.

A further consequence of our approach is that feasibility of a strategy is a referentially opaque concept since a strategy containing $idea-of-phone-number(p, q, s)$ is regarded as feasible while one containing $phone-number(q)$ is not, even though these quantities may be equal in a particular case. Even so, our language is still referentially transparent since feasibility is a concept of the metalanguage.

A classical poser for the reader who wants to solve these difficulties to ponder is, 'George IV wondered whether the author of the Waverley novels was Walter Scott' and 'Walter Scott is the author of the Waverley novels', from which we do not wish to deduce, 'George IV wondered whether Walter Scott was Walter Scott'. This example and others are discussed in the first chapter of Church's *Introduction to Mathematical Logic* (1956).

In the long run it seems that we shall have to use a formalism with referential opacity and formulate precisely the necessary restrictions on replacement of equals by equals; the program must be able to reason about the feasibility of its strategies, and users of natural language handle referential opacity without disaster. In part 4 we give a brief account of the partly successful approach to problems of referential opacity in modal logic.

3. REMARKS AND OPEN PROBLEMS

The formalism presented in part 2 is, we think, an advance on previous attempts, but it is far from epistemological adequacy. In the following sections we discuss a number of problems that it raises. For some of them we have proposals that might lead to solutions.

The approximate character of *result* (p, σ, s).

Using the situational fluent *result*(p, σ, s) in formulating the conditions under which strategies have given effects has two advantages over the *can*(p, π, s) of part 1. It permits more compact and transparent sentences, and it lends itself to the application of the mathematical theory of computation to prove that certain strategies achieve certain goals.

However, we must recognize that it is only an approximation to say that an action, other than that which will actually occur, leads to a definite situation. Thus if someone is asked, 'How would you feel tonight if you challenged him to a duel tomorrow morning and he accepted?' he might well reply, 'I can't imagine the mental state in which I would do it; if the words inexplicably popped out of my mouth as though my voice were under someone else's control that would be one thing; if you gave me a long-lasting belligerence drug that would be another'.

From this we see that *result*(p, σ, s) should not be regarded as being defined in the world itself, but only in certain representations of the world; albeit in representations that may have a preferred character as discussed in part 1.

We regard this as a blemish on the smoothness of interpretation of the formalism, which may also lead to difficulties in the formal development. Perhaps another device can be found which has the advantages of *result* without the disadvantages.

Possible meanings of 'can' for a computer program

A computer program can readily be given much more powerful means of introspection than a person has, for we may make it inspect the whole of its memory including program and data to answer certain introspective questions, and it can even simulate (slowly) what it would do with given initial data. It is interesting to list various notions of *can*(*Program*, π) for a program.

1. There is a sub-program σ and room for it in memory which would achieve π if it were in memory, and control were transferred to σ . No assertion is made that *Program* knows σ or even knows that σ exists.
2. σ exists as above and that σ will achieve π follows from information in memory according to a proof that *Program* is capable of checking.
3. *Program's* standard problem-solving procedure will find σ if achieving π is ever accepted as a subgoal.

The frame problem

In the last section of part 2, in proving that one person could get into conversation with another, we were obliged to add the hypothesis that if a person has a telephone he still has it after looking up a number in the telephone book. If we had a number of actions to be performed in sequence, we would have quite a number of conditions to write down that certain actions do not change the values of certain fluents. In fact with n actions and m fluents we might have to write down mn such conditions.

We see two ways out of this difficulty. The first is to introduce the notion of frame, like the state vector in McCarthy (1962). A number of fluents are declared as attached to the frame and the effect of an action is described by telling which fluents are changed, all others being presumed unchanged.

This can be formalized by making use of yet more ALGOL notation, perhaps in a somewhat generalized form. Consider a strategy in which p performs the action of going from x to y . In the first form of writing strategies we have $go(x,y)$ as a program step. In the second form we have $s := result(p, go(x,y), s)$. Now we may write

$$location(p) := tryfor(y,x)$$

and the fact that other variables are unchanged by this action follows from the general properties of assignment statements. Among the conditions for successful execution of the program will be sentences that enable us to show that when this statement is executed, $tryfor(y,x) = y$. If we were willing to consider that p could go anywhere we could write the assignment statement simply as

$$location(p) := y.$$

The point of using *tryfor* here is that a program using this simpler assignment is, on the face of it, not possible to execute, since p may be unable to go to y . We may cover this case in the more complex assignment by agreeing that when p is barred from y , $tryfor(y,x) = x$.

In general, restrictions on what could appear on the right side of an assignment to a component of the situation would be included in the conditions for the feasibility of the strategy. Since components of the situation that change independently in some circumstances are dependent in others, it may be worthwhile to make use of the block structure of ALGOL. We shall not explore this approach further in this paper.

Another approach to the frame problem may follow from the methods of the next section; and in part 4 we mention a third approach which may be useful, although we have not investigated it at all fully.

Formal literatures

In this section we introduce the notion of formal literature which is to be contrasted with the well-known notion of formal language. We shall mention

some possible applications of this concept in constructing an epistemologically adequate system.

A formal literature is like a formal language with a history: we imagine that up to a certain time a certain sequence of sentences have been said. The literature then determines what sentences may be said next. The formal definition is as follows.

Let A be a set of potential sentences, for example, the set of all finite strings in some alphabet. Let $Seq(A)$ be the set of finite sequences of elements of A and let $L:Seq(A) \rightarrow \{\text{true}, \text{false}\}$ be such that if $\sigma \in Seq(A)$ and $L(\sigma)$, that is, $L(\sigma) = \text{true}$, and σ_1 is an initial segment of σ then $L(\sigma_1)$. The pair (A, L) is termed a *literature*. The interpretation is that a_n may be said after a_1, \dots, a_{n-1} , provided $L((a_1, \dots, a_n))$. We shall also write $\sigma \in L$ and refer to σ as a string of the literature L .

From a literature L and a string $\sigma \in L$ we introduce the derived literature L_σ . Namely, $\tau \in L_\sigma$ if and only if $\sigma * \tau \in L$, where $\sigma * \tau$ denotes the concatenation of σ and τ .

We shall say that the language L is universal for the class Φ of literatures if for every literature $M \in \Phi$ there is a string $\sigma(M) \in L$ such that $M = L_{\sigma(M)}$; that is, $\tau \in M$ if and only if $\sigma(M) * \tau \in L$.

We shall call a literature computable if its strings form a recursively enumerable set. It is easy to see that there is a computable literature U_C that is universal with respect to the set C of computable literatures. Namely, let e be a computable literature and let c be the representation of the Gödel number of the recursively enumerable set of e as a string of elements of A . Then, we say $c * \tau \in U_C$ if and only if $\tau \in e$.

It may be more convenient to describe natural languages as formal literatures than as formal languages: if we allow the definition of new terms and require that new terms be used in accordance with their definitions, then we have restrictions on sentences that depend on what sentences have previously been uttered. In a programming language, the restriction that an identifier not be used until it has been declared, and then only consistently with the declaration, is of this form.

Any natural language may be regarded as universal with respect to the set of natural languages in the approximate sense that we might define French in terms of English and then say 'From now on we shall speak only French'.

All the above is purely syntactic. The applications we envisage to artificial intelligence come from a certain kind of interpreted literature. We are not able to describe precisely the class of literatures that may prove useful, only to sketch a class of examples.

Suppose we have an interpreted language such as first-order logic perhaps including some modal operators. We introduce three additional operators: *consistent*(ϕ), *normally*(ϕ), and *probably*(ϕ). We start with a list of sentences as hypotheses. A new sentence may be added to a string σ of sentences according to the following rules:

1. Any consequence of sentences of σ may be added.
2. If a sentence ϕ is consistent with σ , then *consistent*(ϕ) may be added. Of course, this is a non-computable rule. It may be weakened to say that *consistent*(ϕ) may be added provided ϕ can be shown to be consistent with σ by some particular proof procedure.
3. *normally*(ϕ), *consistent*(ϕ) \vdash *probably*(ϕ).
4. $\phi \vdash$ *probably*(ϕ) is a possible deduction.
5. If $\phi_1, \phi_2, \dots, \phi_n \vdash \phi$ is a possible deduction then *probably*(ϕ_1), ..., *probably*(ϕ_n) \vdash *probably*(ϕ) is also a possible deduction.

The intended application to our formalism is as follows:

In part 2 we considered the example of one person telephoning another, and in this example we assumed that if p looks up q 's phone-number in the book, he will know it, and if he dials the number he will come into conversation with q . It is not hard to think of possible exceptions to these statements such as:

1. The page with q 's number may be torn out.
2. p may be blind.
3. Someone may have deliberately inked out q 's number.
4. The telephone company may have made the entry incorrectly.
5. q may have got the telephone only recently.
6. The phone system may be out of order.
7. q may be incapacitated suddenly.

For each of these possibilities it is possible to add a term excluding the difficulty in question to the condition on the result of performing the action. But we can think of as many additional difficulties as we wish, so it is impractical to exclude each difficulty separately.

We hope to get out of this difficulty by writing such sentences as

$$\forall p . \forall q . \forall s . at(q, home(q), s) \supset normally(in-conversation(p, q, result(p, dials(phone-number(q)), s)))$$

We would then be able to deduce

$$probably(in-conversation(p, q, result(p, dials(phone-number(q)), s_0)))$$

provided there were no statements like

$$kaput(Phone-system, s_0)$$

and

$$\forall s . kaput(Phone-system, s) \supset \neg in-conversation(p, q, result(p, dials(phone-number(q)), s))$$

present in the system.

Many of the problems that give rise to the introduction of frames might be handled in a similar way.

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

The operators *normally*, *consistent* and *probably* are all modal and referentially opaque. We envisage systems in which *probably*(π) and *probably* ($\neg\pi$) and therefore *probably*(false) will arise. Such an event should give rise to a search for a contradiction.

We hereby warn the reader, if it is not already clear to him, that these ideas are very tentative and may prove useless, especially in their present form. However, the problem they are intended to deal with, namely the impossibility of naming every conceivable thing that may go wrong, is an important one for artificial intelligence, and some formalism has to be developed to deal with it.

Probabilities

On numerous occasions it has been suggested that the formalism take uncertainty into account by attaching probabilities to its sentences. We agree that the formalism will eventually have to allow statements about the probabilities of events, but attaching probabilities to all statements has the following objections:

1. It is not clear how to attach probabilities to statements containing quantifiers in a way that corresponds to the amount of conviction people have.
2. The information necessary to assign numerical probabilities is not ordinarily available. Therefore, a formalism that required numerical probabilities would be epistemologically inadequate.

Parallel processing

Besides describing strategies by ALGOL-like programs we may also want to describe the laws of change of the situation by such programs. In doing so we must take into account the fact that many processes are going on simultaneously and that the single-activity-at-a-time ALGOL-like programs will have to be replaced by programs in which processes take place in parallel, in order to get an epistemologically adequate description. This suggests examining the so-called simulation languages; but a quick survey indicates that they are rather restricted in the kinds of processes they allow to take place in parallel and in the types of interaction allowed. Moreover, at present there is no developed formalism that allows proofs of the correctness of parallel programs.

4. DISCUSSION OF LITERATURE

The plan for achieving a generally intelligent program outlined in this paper will clearly be difficult to carry out. Therefore, it is natural to ask if some simpler scheme will work, and we shall devote this section to criticising some simpler schemes that have been proposed.

1. L. Fogel (1966) proposes to evolve intelligent automata by altering their state transition diagrams so that they perform better on tasks of greater

and greater complexity. The experiments described by Fogel involve machines with less than 10 states being evolved to predict the next symbol of a quite simple sequence. We do not think this approach has much chance of achieving interesting results because it seems limited to automata with small numbers of states, say less than 100, whereas computer programs regarded as automata have 2^{10^5} to 2^{10^7} states. This is a reflection of the fact that, while the representation of behaviours by finite automata is metaphysically adequate – in principle every behaviour of which a human or machine is capable can be so represented – this representation is not epistemologically adequate; that is, conditions we might wish to impose on a behaviour, or what is learned from an experience, are not readily expressible as changes in the state diagram of an automaton.

2. A number of investigators (Galanter 1956, Pivar and Finkelstein 1964) have taken the view that intelligence may be regarded as the ability to predict the future of a sequence from observation of its past. Presumably, the idea is that the experience of a person can be regarded as a sequence of discrete events and that intelligent people can predict the future. Artificial intelligence is then studied by writing programs to predict sequences formed according to some simple class of laws (sometimes probabilistic laws). Again the model is metaphysically adequate but epistemologically inadequate.

In other words, what we know about the world is divided into knowledge about many aspects of it, taken separately and with rather weak interaction. A machine that worked with the undifferentiated encoding of experience into a sequence would first have to solve the encoding, a task more difficult than any the sequence extrapolators are prepared to undertake. Moreover, our knowledge is not usable to predict exact sequences of experience. Imagine a person who is correctly predicting the course of a football game he is watching; he is not predicting each visual sensation (the play of light and shadow, the exact movements of the players and the crowd). Instead his prediction is on the level of: team A is getting tired; they should start to fumble or have their passes intercepted.

3. Friedberg (1958, 1959) has experimented with representing behaviour by a computer program and evolving a program by random mutations to perform a task. The epistemological inadequacy of the representation is expressed by the fact that desired changes in behaviour are often not representable by small changes in the machine language form of the program. In particular, the effect on a reasoning program of learning a new fact is not so representable.

4. Newell and Simon worked for a number of years with a program called the General Problem Solver (Newell *et al.* 1959, Newell and Simon 1961). This program represents problems as the task of transforming one symbolic expression into another using a fixed set of transformation rules. They succeeded in putting a fair variety of problems into this form, but for a number of problems the representation was awkward enough so that GPS could only

do small examples. The task of improving GPS was studied as a GPS task, but we believe it was finally abandoned. The name, General Problem Solver, suggests that its authors at one time believed that most problems could be put in its terms, but their more recent publications have indicated other points of view.

It is interesting to compare the point of view of the present paper with that expressed in Newell and Ernst (1965) from which we quote the second paragraph:

We may consider a problem solver to be a process that takes a problem as input and provides (when successful) the solution as output. The problem consists of the problem statement, or what is immediately given; and auxiliary information, which is potentially relevant to the problem but available only as the result of processing. The problem solver has available certain methods for attempting to solve the problem. These are to be applied to an internal representation of the problem. For the problem solver to be able to work on a problem it must first transform the problem statement from its external form into the internal representation. Thus (roughly), the class of problems the problem solver can convert into its internal representation determines how broad or general it is; and its success in obtaining solutions to problems in internal form determines its power. Whether or not universal, such a decomposition fits well the structure of present problem solving programs.

In a very approximate way their division of the problem solver into the input program that converts problems into internal representation and the problem solver proper corresponds to our division into the epistemological and heuristic parts of the artificial intelligence problem. The difference is that we are more concerned with the suitability of the internal representation itself.

Newell (1965) poses the problem of how to get what we call heuristically adequate representations of problems, and Simon (1966) discusses the concept of 'can' in a way that should be compared with the present approach.

Modal logic

It is difficult to give a concise definition of modal logic. It was originally invented by Lewis (1918) in an attempt to avoid the 'paradoxes' of implication (a false proposition implies any proposition). The idea was to distinguish two sorts of truth: *necessary* truth and mere *contingent* truth. A contingently true proposition is one which, though true, could be false. This is formalized by introducing the modal operator \Box (read 'necessarily') which forms propositions from propositions. Then p 's being a necessary truth is expressed by $\Box p$'s being true. More recently, modal logic has become a much-used tool for analysing the logic of such various propositional operators as belief, knowledge and tense.

There are very many possible axiomatizations of the logic of \Box , none of

which seem more intuitively plausible than many others. A full account of the main classical systems is given by Feys (1965), who also includes an excellent bibliography. We shall give here an axiomatization of a fairly simple modal logic, the system M of Feys-Von Wright. One adds to any full axiomatization of propositional calculus the following:

Ax. 1: $\Box p \supset p$

Ax. 2: $\Box(p \supset p) \supset (\Box p \supset \Box q)$

Rule 1: from p and $p \supset q$, infer q

Rule 2: from p , infer $\Box p$.

(This axiomatization is due to Gödel).

There is also a dual modal operator \Diamond , defined as $\neg \Box \neg$. Its intuitive meaning is 'possibly': $\Diamond p$ is true when p is at least possible, although p may be in fact false (or true). The reader will be able to see the intuitive correspondence between $\neg \Diamond p - p$ is impossible, and $\Box \sim p$ - that is, p is necessarily false.

M is a fairly weak modal logic. One can strengthen it by adding axioms, for example, adding Ax. 3: $\Box p \supset \Box \Box p$ yields the system called $S4$; adding Ax. 4: $\Diamond p \supset \Box \Diamond p$ yields $S5$; and other additions are possible. However, one can also weaken all these systems in various ways, for instance by changing Ax. 1 to Ax. 1': $\Box p \supset \Diamond p$. One easily sees that Ax. 1 implies Ax. 1', but the converse is not true. The systems obtained in this way are known as the *deontic* versions of the systems. These modifications will be useful later when we come to consider tense-logics as modal logics.

One should note that the truth or falsity of $\Box p$ is not decided by p 's being true. Thus \Box is not a truth-functional operator (unlike the usual logical connectives, for instance) and so there is no direct way of using truth-tables to analyse propositions containing modal operators. In fact the decision problem for modal propositional calculi has been quite nontrivial. It is just this property which makes modal calculi so useful, as belief, tense, etc., when interpreted as propositional operators, are all nontruthfunctional.

The proliferation of modal propositional calculi, with no clear means of comparison, we shall call the *first problem* of modal logic. Other difficulties arise when we consider modal predicate calculi, that is, when we attempt to introduce quantifiers. This was first done by Barcan-Marcus (1946).

Unfortunately, all the early attempts at modal predicate calculi had unintuitive theorems (see for instance Kripke 1963a), and, moreover, all of them met with difficulties connected with the failure of Leibniz' law of identity, which we shall try to outline.

Leibniz' law is

$L: \forall x . \forall y. x=y \supset (\Phi(x) \equiv \Phi(y))$

where Φ is any open sentence. Now this law fails in modal contexts. For instance, consider this instance of L :

$L_1: \forall x . \forall y. x=y \supset (\Box(x=x) \equiv \Box(x=y))$

By rule 2 of M (which is present in almost all modal logics), since $x=x$ is a theorem, so is $\Box(x=x)$. Thus L_1 yields

$$L_2: \forall x . \forall y. x=y \supset \Box(x=y)$$

But, the argument goes, this is counterintuitive. For instance the morning star is in fact the same individual as the evening star (the planet Venus). However, they are not *necessarily* equal: one can easily imagine that they might be distinct. This famous example is known as the 'morning star paradox'.

This and related difficulties compel one to abandon Leibniz' law in modal predicate calculi, or else to modify the laws of quantification (so that it is impossible to obtain the undesirable instances of universal sentences such as L_2). This solves the purely formal problem, but leads to severe difficulties in interpreting these calculi, as Quine has urged in several papers (cf. Quine 1964).

The difficulty is this. A sentence $\Phi(a)$ is usually thought of as ascribing some property to a certain individual a . Now consider the morning star; clearly, the morning star is necessarily equal to the morning star. However, the evening star is not necessarily equal to the morning star. Thus, this one individual – the planet Venus – both has and does not have the property of being necessarily equal to the morning star. Even if we abandon proper names the difficulty does not disappear: for how are we to interpret a statement like $\exists x . \exists y(x=y \wedge \Phi(x) \wedge \neg\Phi(y))$?

Barcan-Marcus has urged an unconventional reading of the quantifiers to avoid this problem. The discussion between her and Quine in Barcan-Marcus (1963) is very illuminating. However, this raises some difficulties – see Belnap and Dunn (1968) – and the recent semantic theory of modal logic provides a more satisfactory method of interpreting modal sentences.

This theory was developed by several authors (Hintikka 1963, 1967a; Kanger 1957; Kripke 1963a, 1963b, 1965), but chiefly by Kripke. We shall try to give an outline of this theory, but if the reader finds it inadequate he should consult Kripke (1963a).

The idea is that modal calculi describe several *possible worlds* at once, instead of just one. Statements are not assigned a single truth-value, but rather a spectrum of truth-values, one in each possible world. Now, a statement is necessary when it is true in *all* possible worlds – more or less. Actually, in order to get different modal logics (and even then not all of them) one has to be a bit more subtle, and have a binary relation on the set of possible worlds – the alternativeness relation. Then a statement is necessary in a world when it is true in all alternatives to that world. Now it turns out that many common axioms of modal propositional logics correspond directly to conditions on this relation of alternativeness. Thus for instance in the system M above, $Ax . 1$ corresponds to the reflexiveness of the alternativeness relation; $Ax . 3(\Box p \supset \Box\Box p)$ corresponds to its transitivity. If we make the

alternativeness relation into an equivalence relation, then this is just like not having one at all; and it corresponds to the axiom: $\Diamond p \supset \Box \Diamond p$.

This semantic theory already provides an answer to the first problem of modal logic: a rational method is available for classifying the multitude of propositional modal logics. More importantly, it also provides an intelligible interpretation for modal predicate calculi. One has to imagine each possible world as having a set of individuals and an assignment of individuals to names of the language. Then each statement takes on its truthvalue in a world s according to the particular set of individuals and assignment associated with s . Thus, a possible world is an interpretation of the calculus, in the usual sense.

Now, the failure of Leibniz' law is no longer puzzling, for in one world the morning star – for instance – may be equal to (the same individual as) the evening star, but in another the two may be distinct.

There are still difficulties, both formal – the quantification rules have to be modified to avoid unintuitive theorems (see Kripke, 1963a, for the details) – and interpretative: it is not obvious what it means to have the *same* individual existing in *different* worlds.

It is possible to gain the expressive power of modal logic without using modal operators by constructing an ordinary truth-functional logic which describes the multiple-world semantics of modal logic directly. To do this we give every predicate an extra argument (the world-variable; or in our terminology the situation-variable) and instead of writing ' $\Box \Phi$ ', we write

$$\forall t. A(s,t) \supset \Phi(t),$$

where A is the alternativeness relation between situations. Of course we must provide appropriate axioms for A .

The resulting theory will be expressed in the notation of the situation calculus; the proposition Φ has become a propositional fluent $\lambda s. \Phi(s)$, and the 'possible worlds' of the modal semantics are precisely the situations. Notice, however, that the theory we get is weaker than what would have been obtained by adding modal operators directly to the situation calculus, for we can give no translation of assertions such as $\Box \pi(s)$, where s is a situation, which this enriched situation calculus would contain.

It is possible, in this way, to reconstruct within the situation calculus subtheories corresponding to the tense-logics of Prior and to the knowledge-logics of Hintikka, as we shall explain below. However, there is a qualification here: so far we have only explained how to translate the propositional modal logics into the situation calculus. In order to translate quantified modal logic, with its difficulties of referential opacity, we must complicate the situation calculus to a degree which makes it rather clumsy. There is a special predicate on individuals and situation – *exists* (i,s) – which is regarded as true when i names an individual existing in the situation s . This is necessary because situations may contain different individuals. Then quantified

assertions of the modal logic are translated according to the following scheme:

$$\forall x . \Phi(x) \rightarrow \forall x . \text{exists}(x,s) \supset \Phi(x,s)$$

where s is the introduced situation variable.

We shall not go into the details of this extra translation in the examples below, but shall be content to define the translations of the propositional tense and knowledge logics into the situation calculus.

Logic of knowledge

The logic of knowledge was first investigated as a modal logic by Hintikka in his book *Knowledge and belief* (1962). We shall only describe the knowledge calculus. He introduces the modal operator Ka (read 'a knows that'), and its dual Pa , defined as $\neg Ka\neg$. The semantics is obtained by the analogous reading of Ka as: 'it is true in all possible worlds compatible with a 's knowledge that'. The propositional logic of Ka (similar to \Box) turns out to be $S4$, that is, $M + Ax . 3$; but there are some complexities over quantification. (The last chapter of the book contains another excellent account of the overall problem of quantification in modal contexts.) This analysis of knowledge has been criticized in various ways (Chisholm 1963, Follesdal 1967) and Hintikka has replied in several important papers (1967b, 1967c, 1969). The last paper contains a review of the different senses of 'know' and the extent to which they have been adequately formalized. It appears that two senses have resisted capture. First, the idea of 'knowing how', which appears related to our 'can'; and secondly, the concept of knowing a person (place, etc.), when this means 'being acquainted with' as opposed to simply knowing *who* a person is.

In order to translate the (propositional) knowledge calculus into 'situation' language, we introduce a three-place predicate into the situation calculus termed 'shrug'. $\text{Shrug}(p, s_1, s_2)$, where p is a person and s_1 and s_2 are situations, is true when, if p is in fact in situation s_2 , then for all he knows he might be in situation s_1 . That is to say, s_1 is an *epistemic alternative* to s_2 , as far as the individual p is concerned – this is Hintikka's term for his alternative worlds (he calls them model-sets).

Then we translate $K_p q$, where q is a proposition of Hintikka's calculus, as $\forall t . \text{shrug}(p, t, s) \supset q(t)$, where $\lambda s . q(s)$ is the fluent which translates q . Of course we have to supply axioms for *shrug*, and in fact so far as the pure knowledge-calculus is concerned, the only two necessary are

$$K1: \forall s . \forall p . \text{shrug}(p, s, s)$$

$$\text{and } K2: \forall p . \forall s . \forall t . \forall r . (\text{shrug}(p, t, s) \wedge \text{shrug}(p, r, t)) \supset \text{shrug}(p, r, s)$$

that is, reflexivity and transitivity.

Others of course may be needed when we add tenses and other machinery to the situation calculus, in order to relate knowledge to them.

Tense logics

This is one of the largest and most active areas of philosophic logic. Prior's book *Past, present and future* (1968) is an extremely thorough and lucid account of what has been done in the field. We have already mentioned the four propositional operators F, G, P, H which Prior discusses. He regards these as modal operators; then the alternativeness relation of the semantic theory is simply the time-ordering relation. Various axiomatizations are given, corresponding to deterministic and nondeterministic tenses, ending and nonending times, etc; and the problems of quantification turn up again here with renewed intensity. To attempt a summary of Prior's book is a hopeless task, and we simply urge the reader to consult it. More recently several papers have appeared (see, for instance, Bull 1968) which illustrate the technical sophistication tense-logic has reached, in that full completeness proofs for various axiom systems are now available.

As indicated above, the situation calculus contains a tense-logic (or rather several tense-logics), in that we can define Prior's four operators in our system and by suitable axioms reconstruct various axiomatizations of these four operators (in particular, all the axioms in Bull (1968) can be translated into the situation calculus).

Only one extra nonlogical predicate is necessary to do this: it is a binary predicate of situations called *cohistorical*, and is intuitively meant to assert of its arguments that one is in the other's future. This is necessary because we want to consider some pairs of situations as being not temporally related at all. We now define F (for instance) thus:

$$F(\pi, s) \equiv \exists t . \text{cohistorical}(t, s) \wedge \text{time}(t) > \text{time}(s) \wedge \pi(t).$$

The other operators are defined analogously.

Of course we have to supply axioms for 'cohistorical' and time: this is not difficult. For instance, consider one of Bull's axioms, say $Gp \supset GGp$, which is better (for us) expressed in the form $FFp \supset Fp$. Using the definition, this translates into:

$$\begin{aligned} & (\exists t . \text{cohistorical}(t, s) \wedge \text{time}(t) > \text{time}(s) \wedge \exists r . \text{cohistorical}(r, t) \\ & \wedge \text{time}(r) > \text{time}(t) \wedge \pi(r)) \supset (\exists r . \text{cohistorical}(r, s) \\ & \wedge \text{time}(r) > \text{time}(s) \wedge \pi(r)) \end{aligned}$$

which simplifies (using the transitivity of '>') to

$$\forall t . \forall r . (\text{cohistorical}(r, t) \wedge \text{cohistorical}(t, s)) \supset \text{cohistorical}(r, s)$$

that is, the transitivity of 'cohistorical'. This axiom is precisely analogous to the $S4$ axiom $\Box p \supset \Box \Box p$, which corresponded to transitivity of the alternativeness relation in the modal semantics. Bull's other axioms translate into conditions on 'cohistorical' and time in a similar way; we shall not bother here with the rather tedious details.

Rather more interesting would be axioms relating 'shrug' to 'cohistorical'

and time; unfortunately we have been unable to think of any intuitively plausible ones. Thus, if two situations are epistemic alternatives (that is, $shrug(p, s_1, s_2)$) then they may or may not have the same time value (since we want to allow that p may not know what the time is), and they may or may not be cohistorical.

Logics and theories of actions

The most fully developed theory in this area is von Wright's action logic described in his book *Norm and Action* (1963). Von Wright builds his logic on a rather unusual tense-logic of his own. The basis is a binary modal connective T , so that pTq , where p and q are propositions, means ' p , then q '. Thus the action, for instance, of opening the window is: $(the\ window\ is\ closed)T(the\ window\ is\ open)$. The formal development of the calculus was taken a long way in the book cited above, but some problems of interpretation remained as Castañeda points out in his review (1965). In a more recent paper von Wright (1967) has altered and extended his formalism so as to answer these and other criticisms, and also has provided a sort of semantic theory based on the notion of a life-tree.

We know of no other attempts at constructing a single theory of actions which have reached such a degree of development, but there are several discussions of difficulties and surveys which seem important. Rescher (1967) discusses several topics very neatly, and Davidson (1967) also makes some cogent points. Davidson's main thesis is that, in order to translate statements involving actions into the predicate calculus, it appears necessary to allow actions as values of bound variables, that is (by Quine's test) as real individuals. The situation calculus of course follows this advice in that we allow quantification over strategies, which have actions as a special case. Also important are Simon's papers (1965, 1967) on command-logics. Simon's main purpose is to show that a special logic of commands is unnecessary, ordinary logic serving as the only deductive machinery; but this need not detain us here. He makes several points, most notably perhaps that agents are most of the time not performing actions, and that in fact they only stir to action when forced to by some outside interference. He has the particularly interesting example of a serial processor operating in a parallel-demand environment, and the resulting need for interrupts. Action logics such as von Wright's and ours do not distinguish between action and inaction, and we are not aware of any action-logic which has reached a stage of sophistication adequate to meet Simon's implied criticism.

There is a large body of purely philosophical writings on action, time, determinism, etc., most of which is irrelevant for present purposes. However, we mention two which have recently appeared and which seem interesting: a paper by Chisholm (1967) and another paper by Evans (1967), summarizing the recent discussion on the distinctions between states, performances and activities.

Other topics

There are two other areas where some analysis of actions has been necessary: command-logics and logics and theories of obligation. For the former the best reference is Rescher's book (1966) which has an excellent bibliography. Note also Simon's counterarguments to some of Rescher's theses (Simon 1965, 1967). Simon proposes that no special logic of commands is necessary, commands being analysed in the form 'bring it about that p !' for some proposition p , or, more generally, in the form 'bring it about that $P(x)$ by changing x !', where x is a *command* variable, that is, under the agent's control. The translations between commands and statements take place only in the context of a 'complete model', which specifies environmental constraints and defines the command variables. Rescher argues that these schemas for commands are inadequate to handle the *conditional command* 'when p , do q ', which becomes 'bring it about that $(p \supset q)$!': this, unlike the former, is satisfied by making p false.

There are many papers on the logic of obligation and permission. Von Wright's work is oriented in this direction; Castañeda has many papers on the subject and Anderson also has written extensively (his early influential report (1956) is especially worth reading). The review pages of the *Journal of Symbolic Logic* provide many other references. Until fairly recently these theories did not seem of very much relevance to logics of action, but in their new maturity they are beginning to be so.

Counterfactuals

There is, of course, a large literature on this ancient philosophical problem, almost none of which seems directly relevant to us. However, there is one recent theory, developed by Rescher (1964), which may be of use. Rescher's book is so clearly written that we shall not attempt a description of his theory here. The reader should be aware of Sosa's critical review (1967) which suggests some minor alterations.

The importance of this theory for us is that it suggests an alternative approach to the difficulty which we have referred to as the frame problem. In outline, this is as follows. One assumes, as a rule of procedure (or perhaps as a rule of inference), that when actions are performed, *all* propositional fluents which applied to the previous situation also apply to the new situation. This will often yield an inconsistent set of statements about the new situation; Rescher's theory provides a mechanism for restoring consistency in a rational way, and giving as a by-product those fluents which change in value as a result of performing the action. However, we have not investigated this in detail.

The communication process

We have not considered the problems of formally describing the process of communication in this paper, but it seems clear that they will have to be

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

tackled eventually. Philosophical logicians have been spontaneously active here. The major work is Harrah's book (1963); Cresswell has written several papers on 'the logic of interrogatives', see for instance Cresswell (1965). Among other authors we may mention Åqvist (1965) and Belnap (1963); again the review pages of the *Journal of Symbolic Logic* will provide other references.

Acknowledgements

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183), and in part by the Science Research Council (B/SR/2299)

REFERENCES

- Anderson, A. R. (1956) The formal analysis of normative systems. Reprinted in *The Logic of decision and action* (ed. Rescher, N.). Pittsburgh: University of Pittsburgh Press.
- Åqvist, L. (1965) *A new approach to the logical theory of interrogatives, part I*. Uppsala: Uppsala Philosophical Association.
- Barcan-Marcus, R. C. (1946) A functional calculus of the first order based on strict implication. *Journal of Symbolic Logic*, 11, 1-16.
- Barcan-Marcus, R. C. (1963) Modalities and intensional languages. *Boston studies in the Philosophy of Science*. (ed. Wartofsky, W.). Dordrecht, Holland.
- Belnap, N. D. (1963) *An analysis of questions*. Santa Monica.
- Belnap, N. D. & Dunn, J. M. (1968) The substitution interpretation of the quantifiers. *Noûs*, 2, 177-85.
- Bull, R. A. (1968) An algebraic study of tense logics with linear time. *Journal of Symbolic Logic*, 33, 27-39
- Castañeda, H. N. (1965) The logic of change, action and norms. *Journal of Philosophy*, 62, 333-4.
- Chisholm, R. M. (1963) The logic of knowing. *Journal of Philosophy*, 60, 773-95.
- Chisholm, R. M. (1967) He could have done otherwise. *Journal of Philosophy*, 64, 409-17.
- Church, A. (1956) *Introduction to Mathematical Logic*. Princeton: Princeton University Press.
- Cresswell, M. J. (1965). The logic of interrogatives. *Formal systems and recursive functions*. (ed. Crossley, J. N. & Dummett, M. A. E.). Amsterdam: North-Holland.
- Davidson, D. (1967) The logical form of action sentences. *The logic of decision and action*. (ed. Rescher, N.). Pittsburgh: University of Pittsburgh Press.
- Evans, C. O. (1967) States, activities and performances. *Australasian Journal of Philosophy*, 45, 293-308.
- Feys, R. (1965) *Modal Logics*. (ed. Dopp, J.). Louvain: Coll. de Logique Math. série B.
- Fogel, L. J., Owens, A. J. & Walsh, M. J. (1966) *Artificial Intelligence through simulated evolution*. New York: John Wiley.
- Føllesdal, D. (1967) Knowledge, identity and existence. *Theoria*, 33, 1-27.
- Friedberg, R. M. (1958) A learning machine, part I. *IBM J. Res. Dev.*, 2, 2-13.
- Friedberg, R. M., Dunham, B., & North, J. H. (1959) A learning machine, part II. *IBM J. Res. Dev.*, 3, 282-7.
- Galanter, E. & Gerstenhaber, M. (1956). On thought: the extrinsic theory. *Psychological Review*, 63, 218-27
- Green, C. (1969) Theorem-proving by resolution as a basis for question-answering systems. *Machine Intelligence 4*, pp.183-205 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

- Harrah, D. (1963) *Communication: a logical model*. Cambridge, Massachusetts: MIT press.
- Hintikka, J. (1962) *Knowledge and belief: an introduction to the logic of the two notions*. New York: Cornell University Press.
- Hintikka, J. (1963) The modes of modality. *Acta Philosophica Fennica*, 16, 65–82.
- Hintikka, J. (1967a) A program and a set of concepts for philosophical logic. *The Monist*, 51, 69–72.
- Hintikka, J. (1967b) Existence and identity in epistemic contexts. *Theoria*, 32, 138–47.
- Hintikka, J. (1967c) Individuals, possible worlds and epistemic logic. *Noûs*, 1, 33–62.
- Hintikka, J. (1969) Alternative constructions in terms of the basic epistemological attitudes *Contemporary philosophy in Scandinavia* (ed. Olsen, R. E.) (to appear).
- Kanger, S. (1957) A note on quantification and modalities. *Theoria*, 23, 133–4.
- Kripke, S. (1963a) Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16, 83–94.
- Kripke, S. (1963b) Semantical analysis of modal logic I. *Zeitschrift für math. Logik und Grundlagen der Mathematik*, 9, 67–96.
- Kripke, S. (1965) Semantical analysis of modal logic II. *The theory of models* (eds Addison, Henkin & Tarski). Amsterdam: North-Holland.
- Lewis, C. I. (1918) *A survey of symbolic logic*. Berkeley: University of California Press.
- Manna, Z. (1968a) *Termination of algorithms*. Ph.D Thesis, Carnegie-Mellon University.
- Manna, Z. (1968b) *Formalization of properties of programs*. Stanford Artificial Intelligence Report: Project Memo AI-64.
- McCarthy, J. (1959) Programs with common sense. *Mechanization of thought processes*, Vol. I. London: HMSO
- McCarthy, J. (1962) Towards a mathematical science of computation. *Proc. IFIP Congress 62*. Amsterdam: North-Holland Press.
- McCarthy, J. (1963) *Situations, actions and causal laws*. Stanford Artificial Intelligence Project: Memo 2.
- Minsky, M. (1961) Steps towards artificial intelligence. *Proceedings of the I.R.E.*, 49, 8–30.
- Newell, A., Shaw, V. C. & Simon, H. A. (1959) Report on a general problem-solving program. *Proceedings ICIP*. Paris: UNESCO House.
- Newell, A. & Simon H. A. (1961) GPS – a program that simulates human problem-solving. *Proceedings of a conference in learning automata*. Munich: Oldenbourg.
- Newell, A. (1965) Limitations of the current stock of ideas about problem-solving. *Proceedings of a conference on Electronic Information Handling*, pp. 195–208 (eds Kent, A. & Taulbee, O.). New York: Spartan.
- Newell, A. & Ernst, C. (1965) The search for generality. *Proc. IFIP Congress 65*.
- Pivar, M. & Finkelstein, M. (1964). *The Programming Language LISP: its operation and applications* (eds Berkely, E. C. & Bobrow, D. G.). Cambridge, Massachusetts: MIT Press.
- Prior, A. N. (1957) *Time and modality*. Oxford: Clarendon Press.
- Prior, A. N. (1968) *Past, present and future*. Oxford: Clarendon Press.
- Quine, W. V. O. (1964) Reference and modality. *From a logical point of view*. Cambridge, Massachusetts: Harvard University Press.
- Rescher, N. (1964) *Hypothetical reasoning*. Amsterdam: North-Holland.
- Rescher, N. (1966) *The logic of commands*. London: Routledge.
- Rescher, N. (1967) Aspects of action. *The logic of decision and action* (ed. Rescher, N.). Pittsburgh: University of Pittsburgh Press.
- Shannon, C. (1950) Programming a computer for playing chess. *Philosophical Magazine*, 41.
- Simon, H. A. (1965) The logic of rational decision. *British Journal for the Philosophy of Science*, 16, 169–86.
- Simon, H. A. (1966) *On Reasoning about actions*. Carnegie Institute of Technology: Complex Information Processing Paper 87.

PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

- Simon, H.A. (1967) The logic of heuristic decision making. *The logic of decision and action* (ed. Rescher, N.). Pittsburgh: University of Pittsburgh Press.
- Sosa, E. (1967) Hypothetical reasoning. *Journal of Philosophy*, 64, 293-305.
- Turing, A.M. (1950) Computing machinery and intelligence. *Mind*, 59, 433-60.
- von Wright, C.H. (1963) *Norm and action: a logical enquiry*. London: Routledge.
- von Wright, C.H. (1967) The Logic of Action—a sketch. *The logic of decision and action* (ed. Rescher, N.). Pittsburgh: University of Pittsburgh Press.

INDEX



INDEX

- Absys-1 423
- adaline 395-96, 398
- advice-taker 184, 452, 468
- algebra 17-42
 - Ω -algebra 19-37
- ALGOL 3, 50, 361, 481, 487, 490
- alpha-beta heuristic 258, 261
- Anderson 499, 500
- Andreae 433, 451-4
- answer statement 192-9
- application 151-68
 - Åqvist 500
- Arbib 23, 39, 43
- Arrow 289, 310
- assertions 423-9
- Atkinson 263
- ATLAS-AUTOCODE 99
- automaton 23, 285-7, 291, 298-310, 433-54, 463, 470-5

- Backer, 175, 181
- Balzer 290, 310
- Barcan-Marcus 493-4, 500
- Bar-Hillel 468
- Belnap 494, 500
- Beth 62, 70
- Bharucha-Reid 321, 335
- bint* 367
- Black 203
- Blum 403, 420
- Bohnert 175, 181
- Booth, A. D. 337, 348
- Booth, K. H. V. 337, 348
- Boring 379, 381
- Borovikov 285, 310
- Bousfield 320, 335
- Bratley 273, 280, 284
- Bryzgalov 285, 310
- Buchanan 461, 462
- Bull 497, 500
- Bullock 318, 335
- Burks 290, 310
- Burstall 27, 29, 43, 176, 179, 181, 335, 446, 454, 457, 462

- Cahn 420
- caltrop 392-6, 398
- Carson 86, 95, 144-5, 181, 205
- Cartwright 321, 336
- Cashin 452, 454
- Castañeda 498-500
- cdc 6600 computer 81
- Chambers 287, 310, 433, 454
- character recognition 385, 396, 413
- chess 255, 258, 262-3, 267, 464
- Chinlund 64, 70
- Chisholm 496, 498, 500
- choice-tree 457-9, 461
- Chomsky 184, 205, 361-2, 364-5, 380-3
- chromosome 403-4, 413-14, 417-19
- Church 142, 144, 485, 500
- clash 87, 89, 92-8, 123-5
- Clowes 362, 371, 381
- Cohen 326, 336
- Cohn 19, 28, 33, 43
- Coles, L. S. 183, 203, 204
- Coles, W. 403, 420
- Colin 263
- Collier 349, 357
- Cooper, D. C. 29, 43, 62, 70, 186, 205
- Cooper, W. S. 203, 205
- Copi 174, 175, 181
- Cox 335
- Craik 335
- Cresswell 500

- Dakin 273, 284
- Darlington 100, 139, 144, 174, 175, 181
- Davidson 498, 500
- Davis 62, 64, 68-70, 140, 144, 186, 205
- Deese 324, 329, 336
- DENDRAL 209, 211, 221, 227-8, 234-5, 249-51, 254, 461, 462
 - algorithm 252
 - notation 246
- dependency theory 272-81
- Deutsch 403, 420
- Dewar 280, 284
- Djerassi 211

INDEX

- Doran 433, 450, 454, 456-7, 462
 Duffield 211
 Dunham 62, 64, 70, 500
 Dunn 494, 500
- Elcock 423, 429
 Elliot 4120 computer 423
 Elliot 4130 computer 448
 equality 100, 103ff, 125, 135-6, 139-42, 153-4, 158, 173
 Ernst 492, 501
 Evans, C. O. 498, 500
 Evans, T. G. 377, 381
 Everett 395
- factor 90, 92-5, 141, 150, 177, 199, 201
 Fatt 318, 336
 Feigerbaum 254, 462
 Feys 493, 500
 Finkelstein 491, 501
 Floyd 3, 4, 7, 15, 29, 43
 fluent 477-87, 495
 Fogel 490-1, 500
 Føllesdal 496, 500
 FORTRAN 257
 Foster 423, 429
 Freddy 455, 456, 459, 460
 Frick 371, 381
 Fridsal 62, 64, 70
 Friedberg 491, 500
 Frischkopf 318, 338
- Gaifman 272, 284
 Galanter 491, 500
 Gel'fand 286, 310
 Gentzen sequents 62
 Gerstenhaber 500
 Gilmore 61, 70
 Ginzburg 285, 310
 Gödel 493
 Goore game 285-7, 310
 grammar 272, 321, 361-7, 381, 455
 graph 45-50, 52, 54, 211, 251, 321, 324-6
 centre of 47-50
 chemical 235
 molecular 209
 spanning tree of, *see* tree
 Graph Traverser program 450, 456-7
 Greanias 376, 381
 Green, B. F. jnr. 184, 205, 381
 Green, C. C. 175, 179, 181, 185, 186, 202, 205, 452, 454, 463, 500
 Greenblatt 267
 group theory 80, 140-1, 145
 Grundfest 318, 336
 Guzman 377, 381
- Harary 321, 325, 336
 hare and hounds 337-46
 Harrah 500, 501
 Harris 321, 322, 336
 Hart 202, 205
 Hasenjaeger 181
 Hayes 100, 176-8, 181, 199, 205
 Hays 272
 Henkin 161, 170
 Henkin's theorem 160, 161
 Herbrand's theorem 74, 87, 89, 90, 103, 107, 122
 Herbrand Universe 60, 63, 74, 88, 105, 107, 122, 135, 192
 Hinman 64, 70
 Hintikka 494-6, 501
 holograph 349, 352-3, 356-7
 holophone 349-57
 Hormann 452, 454
 Hurwicz 289, 310
 hyper-resolution 87, 97-9, 177, 181
- Ianov 4, 15
 IBM 7090 computer 181, 420
 ICL-1900 computer 256, 267
 information retrieval 173-5, 179, 184, 187, 281, 315, 316, 334
 information theory 388
 interaction
 in automata collectives 285-7, 290, 305, 308, 310
 man-machine 151, 162, 169, 199, 256
 Izzo 403, 420
- Jenkins 323, 336
- Kamensky, 388, 395
 Kanger 62, 63, 70, 494, 501
 Kaplan 29, 43
 Karlgren 68
 Katz 318, 336
 KDF 9 computer 50, 99, 274
 Kirsch 362, 381, 403, 420
 Kiss 315, 324-6, 329, 330, 336, 453
 Klemmer 371, 381
 König's infinity lemma 61, 90, 161
 Kowalski 176-8, 181
 Kripke 493-5, 501
- lambda-calculus 151, 165, 167, 169
 lambda-expressions 425-7, 478
 Lance 381
 Landin 24, 43, 169, 170
 Langridge 370, 381
 Laughery 184, 205, 381
 Lederberg 211, 220, 250, 254
 Ledley 376, 381
- Hall 145

- Levenstein 290-1, 296, 310
 Levien 173, 178, 181
 Levinson 348
 Lewis 492, 501
 Lindsay 184, 205
 LISP 21, 23, 186, 200, 211, 222, 226, 228,
 232, 235, 238, 501
 Liu 388, 395
 logic
 first-order 3-10, 13-15, 59, 73, 103ff.,
 135-6, 141-2, 169, 183-6, 196, 199, 202-3,
 482
 higher-order 18, 151ff., 168-9, 468
 modal 485, 492ff.
 second-order 3, 7-9, 14, 483
 Longuet-Higgins 349, 357
 Loveland 64, 73, 77, 80, 82, 83, 86
 Luce 317, 336
 Luckham 4, 15, 82-4, 86, 97, 99

 Manna 3, 4, 7, 8, 15, 482, 501
 Marakhovskii 290, 310, 311
 Maron 173, 178, 181
 mass spectrometry 209-13, 218, 220, 223-7,
 230, 232, 234, 239, 245-9, 251, 464
 mathematical induction 165
 matrix 65-8, 73, 103, 192, 290, 325, 330,
 332, 389-96, 404
 matrix reduction, principle of 66-8, 70
 McCarthy 29, 43, 100, 184, 199, 203-5, 452,
 454, 463, 468, 477, 487, 501
 McCormick 403, 420
 McGill 320-1, 336
 McIlroy 64
 Meagher 381
 Meinwald 211
 Meleshina 287, 311
 Meltzer 87, 99, 175, 178, 181
 Meyer 132
 Michie 257, 263, 287, 335, 348, 433, 438, 442,
 454
 Miller 371, 381
 Minsky 362, 381, 452, 454, 464
 model elimination procedure 73, 77-86
 Moore 290-1, 304, 310
 Murray 429
 Myhill 291

 Narasimhan 362-3, 381, 403, 420
 Naur 3, 15
 von Neumann 290, 310
 Newell 452, 454, 464, 491-2, 501
 Nilsson 389, 396
 Nimzowitch 268
 Norman 321, 336
 North 500
 Notley 462
 numerical taxonomy 456

 Obruca 51-4
 organic chemistry 209ff.
 Ovsievich 310
 Owens 500

 P₁-deduction 95-9, 175-81
 Painter 29, 43
 Palermo 323, 336
 Paley 393, 396
 paramodulation 135, 139-47
 Park 4, 15
 Parris 55
 Paterson 4, 5, 10, 11, 15
 PDP-6 computer 211
 PDP-10 computer 267
 Percival 51, 55
 Perekrest 287, 311
 Peschanskii 290, 310, 311
 Peterson 395
 Petri 181
 Pfalz 403, 405, 420
 Philbrick 403, 420
 phrase structure 272, 275, 277, 361, 367,
 455, 456
 Pittel' 286, 310
 Pivar 491, 501
 PLAN-3 256
 planning tree 438-41, 449, 451-3
 Pollis 324-6, 336
 POP-2 27, 43, 274, 354, 446, 454, 457-9, 462
 Popplestone 27, 43, 335, 438, 442, 446, 454,
 457, 462
 Prawitz, D. 61, 62, 66, 68-71
 Prawitz, H. 61, 62, 71
 predicate calculus, *see* logic, first-order
 Preston 403, 420
 Prior 479, 480, 495, 497, 501
 program scheme 3-15
 proof procedure 59, 103, 151, 159-61, 168,
 195
 Putnam 62, 64, 68, 70, 140, 144
 Pyatetskii-Shapiro 286, 310

 question-answering system 183-5
 Quine 142, 144, 494, 501

 Raphael 175, 179, 181, 185, 186, 202-5
 Ray 420
 Read 50, 51, 55
 renaming 87, 95, 99, 175-7
 Rescher 498, 499, 501
 resolution 67, 73, 77-86, 89, 92, 94, 95, 122,
 137, 141-4, 145, 147, 173, 176, 177-86,
 189, 193, 195, 198-9, 201, 205, 454
 Roberts 392, 396, 461-2
 Robertson 211
 Robinson G. A. 86, 95, 141-5, 150, 181,
 205

INDEX

- Robinson J. A. 64, 71-4, 86-7, 90-5, 100,
 103-7, 114, 122-7, 132-3, 141, 143-4,
 169, 170, 173, 175, 177, 181, 185-6, 192,
 205
 Rosenberg 326, 336
 Rosenblatt 386, 396
 Rosenblith 318, 336
 Rosenfeld 403, 405, 420
 Ross 450
 Russell 323, 336
 Rutledge 4, 15
 Rutovitz 420

 Sabidussi 49, 55
 Safier 185, 203, 205
 Sandewall 452, 454
 Saraga 388, 396
 Schlosberg 328, 336
 Scoins 46, 55
 Scott 267
 SDs 940 computer 186
 Sedgewick 320, 335
 semantic partition 159-63, 165, 168
 semantic trees 87-91, 181
 set of support strategy 83, 95-6, 99, 100, 131,
 139, 143, 174-5, 178, 180-1, 200, 202,
 205
 extended 186
 Shalla 145, 205
 Shannon 321, 464, 501
 Shaw 362-3, 382, 452, 454, 501
 Sibert 100-1
 Simon 203, 205, 452, 454, 491-2, 498-9,
 501-2
 skeleton 403, 407-20
 Skolem 59, 61, 71
 Skolem function 60, 97, 167, 189, 191-4
 functional form 60, 73, 77, 173
 Skolem-Löwenheim-Gödel theorem 192
 Slagle 87, 95-8, 103, 123, 133, 143-4, 176,
 181, 185, 203, 205
 Smith 100
 Sneath 456, 462
 SNOBOL 23
 Sokal 456, 462
 Sosa 499, 502
 Stanton 381
 Storey 348
 subsumption 91, 92, 125-30, 199-201
 Sutherland, G. 254, 462
 Sutherland, I. E. 362, 382
 Sward 62, 64, 70
 synchronisation problem 290, 296, 298-302,
 305, 310
 Taylor 386, 396
 Thorne 280, 284
 time-sharing 78, 186, 256
 travelling salesman problem 287, 289-90
 Travis 452, 454
 tree 46, 53-4, 88, 161, 361
 centre of 253
 free 47
 height representation of 46, 51
 mushrooming 53-4
 planning, *see* planning tree
 rooted 46-7
 semantic, *see* semantic tree
 sentence 278
 spanning, of a graph 46-7, 49, 51-4
 Treisman 317, 327, 336
 Tsetlin 285-6, 289, 310-11
 Turing 464-5, 502
 type 151-3, 161-2, 169, 424
 type theory 170

 Uffelman 389, 396
 Ulam 290, 311
 unit preference strategy 86, 174, 181, 200,
 201, 205
 Urban 420
 Urquhart 273
 Uzawa 289, 310

 Varshavsky 285, 287, 289-91, 311
 Verveen 318, 336
 Vigor 281, 284
 Voghera 61, 62, 71
 Volkonskii 286, 311
 Vorontsova 285, 289, 311

 Waksman 290, 311
 Walsh 500
 Wang 61, 62, 71, 103, 133
 Weaver 396
 Weiher 211
 White 211
 Widrow 395-6
 Wolf 184, 205, 381
 Woodward 361, 382
 Woodworth 328, 336
 Woollons 396
 word-association networks 323-30
 word store 315-21, 329
 Wos 79, 82, 83, 86, 95, 122, 131, 133, 139,
 141-5, 150, 174, 181, 186, 202, 205
 von Wright 493, 498-9, 502

 Yates 204
 Yule 346-8

