Meltzer
& Michie

# Machine Intelligence 5

Edinburgh

# MACHINE INTELLIGENCE 5

# MACHINE INTELLIGENCE 5

edited by
BERNARD MELTZER
Metamathematics Unit
University of Edinburgh
and
DONALD MICHIE
Department of Machine Intelligence and Perception
University of Edinburgh

with a previously unpublished report by
A. M. TURING [1912–1954]

EDINBURGH at the University Press

© 1969 Edinburgh University Press
22 George Square, Edinburgh

Printed in Great Britain at
The Aberdeen University Press

85224 176 3

Library of Congress
Catalog Card Number
67-13648

# CONTENTS


INTRODUCTION vii

PROLOGUE

Intelligent machinery. A. M. TURING [1912–1954] 3

MATHEMATICAL FOUNDATIONS

1 Properties of programs and partial function logic. Z. MANNA and J. MCCARTHY 27
2 Program schemes and recursive function theory. R. MILNER 39
3 Fixpoint induction and proofs of program properties D. PARK 59
4 Formal description of program structure and semantics in first order logic. R. M. BURSTALL 79
5 A program machine symmetric automata theory. P. J. LANDIN 99

MECHANIZED REASONING

6 A note on mechanizing higher order logic. J. A. ROBINSON 121
7 Transformational systems and the algebraic structure of atomic formulas. J. C. REYNOLDS 135
8 A note on inductive generalization. G. D. PLOTKIN 153
9 Power amplification for automatic theorem-provers. B. MELTZER 165
10 Search strategies for theorem-proving. R. KOWALSKI 181
11 An experiment in automatic induction. R. J. POPPLESTONE 203

MACHINE LEARNING AND HEURISTIC SEARCH

12 First results on the effect of error in heuristic search. I. POHL 219
13 A set-oriented property-structure representation for binary relations, SPB. E. J. SANDEWALL 237
14 Rediscovering some problems of artificial intelligence in the context of organic chemistry. B. G. BUCHANAN, G. L. SUTHERLAND and E. A. FEIGENBAUM 253
15 Memo functions, the Graph Traverser and a simple control situation. D. L. MARSH 281
16 Experiments with the adaptive Graph Traverser. D. MICHIE and R. ROSS 301

## MAN–MACHINE INTERACTION

17 An interactive theorem-proving program. J.R.ALLEN and D.LUCKHAM 321
18 A symbol manipulation system.
F.V.MCBRIDE, D.J.T.MORRISON and R.M.PENGELLY 337

## COGNITIVE PROCESSES: METHODS AND MODELS

19 Associative memory models.
D.WILLSHAW and H.C.LONGUET-HIGGINS 351
20 Hierarchical decomposition of complexity. M.H.VAN EMDEN 361

## PATTERN RECOGNITION

21 A grammar for the topological analysis of plane figures. P.BUNEMAN 383
22 Shape analysis by use of Walsh functions. N.H.SEARLE 395
23 Conic sections in automatic chromosome analysis. K.A.PATON 411
24 Centromere finding: some shape descriptors for small chromosome outlines. D. RUTOVITZ 435
25 ESOTERIC II – an approach to practical voice control: progress report 69.
D.R.HILL and E.B.WACKER 463
26 On imitative systems theory and pattern recognition. P.A.V.HALL 495

## PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

27 Planning and robots. J.E.DORAN 519
28 Robotologic. P.J.HAYES 533
29 Design of low-cost equipment for cognitive robot research.
H.G.BARROW and S.H.SALTER 555

## APPENDIX

Bibliography on proving the correctness of computer programs.
R.L.LONDON 569

## INDEX

581

# INTRODUCTION

The aim of the Annual Machine Intelligence Workshop is to bring into a single compass separate schools of work which have something in common: that something is the potentiality for introducing attributes of intelligence into computing systems. We have this year the opportunity of prefacing the published proceedings with a document of unusual historical interest, which was written by A. M. Turing – but never published – at the moment when the first stored-program digital machines were becoming operational. His analyses and forecasts, made in 1947, can here be read and compared with what has actually occurred since then.

The fact that less than a quarter of a century has since elapsed indicates an impressive tempo of development, which is quickening rather than slackening. The effects on our society are likely to be profound, and it can be argued that the practitioners themselves should now attempt to acquire some understanding of the impact of the nascent new technology which they are introducing into our lives. This was the theme of a thoughtful lunch-time address delivered to the Workshop by Dr Jeremy Bray, MP, who was at that time Joint Parliamentary Secretary in the Ministry of Technology with special responsibility for computing. The closing session of the Workshop, which discussed the issues raised, resolved to invite in each subsequent year a paper, usually from outside the ranks of the profession, which should be concerned with social applications and implications of developments in artificial intelligence.

Once again we record our gratitude to the Science Research Council who met the main costs of holding the Workshop. We also owe thanks to the University of Edinburgh and to its Principal, Professor M. M. Swann FRS, for the provision of facilities for the participants and for acting as host to the Workshop lunch; to Mrs J. E. Hayes of the Department of Machine Intelligence and Perception for her help with the Workshop arrangements; and to Dr Helen Muirhead of the Edinburgh University Press for the precision and enthusiasm with which, once again, a high standard and speed of publication has been attained.

B. MELTZER
D. MICHIE
*December* 1969

# PROLOGUE

*Editors' note*

The essay by Alan Turing, which we reproduce here, was written in September 1947, when the world's first stored-program digital computers, to a significant degree his own conceptual creation, were about to become operational. The paper was submitted in 1948 to the National Physical Laboratory, where Turing was then employed, as a report on his year's sabbatical leave which he had spent at Cambridge. During the same period Turing achieved his demonstration of the unsolvability of the word problem for semi-groups with cancellation.

We here record our thanks to Dr R. O. Gandy for making the typescript available, and to him and the North Holland Publishing Company for waiving certain restrictions of copyright. A condensed version is to appear in the *Collected Works of A.M.Turing* which is forthcoming under Dr Gandy's editorship. We also thank Mr Michael Woodger, who incidentally helped Turing finish it by drawing the original diagrams, for an unforgettable account of the furore created by Turing at N.P.L. with his prognostications of intelligent machinery: 'Turing is going to infest the countryside' some declared 'with a robot which will live on twigs and scrap iron!'

The anticipation of the notion of a sub-routine on page 21 and of the device of doing machine problem-solving *via* theorem-proving algorithms (p. 22) are striking examples of the prophetic insight which pervades the essay.

# Intelligent Machinery

A. M. Turing
[1912—1954]

**Abstract**

The possible ways in which machinery might be made to show intelligent behaviour are discussed. The analogy with the human brain is used as a guiding principle. It is pointed out that the potentialities of the human intelligence can only be realized if suitable education is provided. The investigation mainly centres round an analogous teaching process applied to machines. The idea of an unorganized machine is defined, and it is suggested that the infant human cortex is of this nature. Simple examples of such machines are given, and their education by means of rewards and punishments is discussed. In one case the education process is carried through until the organization is similar to that of an ACE.

I propose to investigate the question as to whether it is possible for machinery to show intelligent behaviour. It is usually assumed without argument that it is not possible. Common catch phrases such as 'acting like a machine', 'purely mechanical behaviour' reveal this common attitude. It is not difficult to see why such an attitude should have arisen. Some of the reasons are:

(a) An unwillingness to admit the possibility that mankind can have any rivals in intellectual power. This occurs as much amongst intellectual people as amongst others: they have more to lose. Those who admit the possibility all agree that its realization would be very disagreeable. The same situation arises in connection with the possibility of our being superseded by some other animal species. This is almost as disagreeable and its theoretical possibility is indisputable.

(b) A religious belief that any attempt to construct such machines is a sort of Promethean irreverence.

(c) The very limited character of the machinery which has been used until recent times (e.g. up to 1940). This encouraged the belief that machinery was necessarily limited to extremely straightforward, possibly even to repetitive, jobs. This attitude is very well expressed by Dorothy Sayers (*The Mind of the Maker* p. 46) '. . . which imagines that God, having created his Universe, has now screwed the cap on His pen, put His feet on the

mantelpiece and left the work to get on with itself.' This, however, rather comes into St Augustine's category of figures of speech or enigmatic sayings framed from things which do not exist at all. We simply do not know of any creation which goes on creating itself in variety when the creator has withdrawn from it. The idea is that God simply created a vast machine and has left it working until it runs down from lack of fuel. This is another of those obscure analogies, since we have no experience of machines that produce variety of their own accord; the nature of a machine is to 'do the same thing over and over again so long as it keeps going'.

(d) Recently the theorem of Gödel and related results (Gödel 1931, Church 1936, Turing 1937) have shown that if one tries to use machines for such purposes as determining the truth or falsity of mathematical theorems and one is not willing to tolerate an occasional wrong result, then any given machine will in some cases be unable to give an answer at all. On the other hand the human intelligence seems to be able to find methods of ever-increasing power for dealing with such problems 'transcending' the methods available to machines.

(e) In so far as a machine can show intelligence this is to be regarded as nothing but a reflection of the intelligence of its creator.

## REFUTATION OF SOME OBJECTIONS

In this section I propose to outline reasons why we do not need to be influenced by the above-described objections. The objections (a) and (b), being purely emotional, do not really need to be refuted. If one feels it necessary to refute them there is little to be said that could hope to prevail, though the actual production of the machines would probably have some effect. In so far then as we are influenced by such arguments we are bound to be left feeling rather uneasy about the whole project, at any rate for the present. These arguments cannot be wholly ignored, because the idea of 'intelligence' is itself emotional rather than mathematical.

The objection (c) in its crudest form is refuted at once by the actual existence of machinery (ENIAC etc.) which can go on through immense numbers (e.g. $10^{60,000}$ about for ACE) of operations without repetition, assuming no breakdown. The more subtle forms of this objection will be considered at length on pages 18–22.

The argument from Gödel's and other theorems (objection d) rests essentially on the condition that the machine must not make mistakes. But this is not a requirement for intelligence. It is related that the infant Gauss was asked at school to do the addition $15+18+21+\ldots+54$ (or something of the kind) and that he immediately wrote down 483, presumably having calculated it as $(15+54)(54-12)/2.3$. One can imagine circumstances where a foolish master told the child that he ought instead to have added 18 to 15 obtaining 33, then added 21, etc. From some points of view this would be a 'mistake', in spite of the obvious intelligence involved. One can also

imagine a situation where the children were given a number of additions to do, of which the first 5 were all arithmetic progressions, but the 6th was say $23+34+45+\ldots+100+112+122+\ldots+199$. Gauss might have given the answer to this as if it were an arithmetic progression, not having noticed that the 9th term was 112 instead of 111. This would be a definite mistake, which the less intelligent children would not have been likely to make.

The view (d) that intelligence in machinery is merely a reflection of that of its creator is rather similar to the view that the credit for the discoveries of a pupil should be given to his teacher. In such a case the teacher would be pleased with the success of his methods of education, but would not claim the results themselves unless he had actually communicated them to his pupil. He would certainly have envisaged in very broad outline the sort of thing his pupil might be expected to do, but would not expect to foresee any sort of detail. It is already possible to produce machines where this sort of situation arises in a small degree. One can produce 'paper machines' for playing chess. Playing against such a machine gives a definite feeling that one is pitting one's wits against something alive.

These views will all be developed more completely below.

## VARIETIES OF MACHINERY

It will not be possible to discuss possible means of producing intelligent machinery without introducing a number of technical terms to describe different kinds of existent machinery.

'*Discrete*' *and* '*continuous*' *machinery.* We may call a machine 'discrete' when it is natural to describe its possible states as a discrete set, the motion of the machine occurring by jumping from one state to another. The states of 'continuous' machinery on the other hand form a continuous manifold, and the behaviour of the machine is described by a curve on this manifold. All machinery can be regarded as continuous, but when it is possible to regard it as discrete it is usually best to do so. The states of discrete machinery will be described as 'configurations'.

'*Controlling*' *and* '*active*' *machinery.* Machinery may be described as 'controlling' if it only deals with information. In practice this condition is much the same as saying that the magnitude of the machine's effects may be as small as we please, so long as we do not introduce confusion through Brownian movement, etc. 'Active' machinery is intended to produce some definite physical effect.

| | | |
|---|---|---|
| *Examples* | A Bulldozer | Continuous Active |
| | A Telephone | Continuous Controlling |
| | A Brunsviga* | Discrete Controlling |
| | A Brain (probably) | Continuous Controlling, but is very similar to much discrete machinery |

* The main desk calculating machine of the day.

| | |
|---|---|
| The ENIAC, ACE, etc. | Discrete Controlling |
| A Differential Analyser | Continuous Controlling. |

We shall mainly be concerned with discrete controlling machinery. As we have mentioned, brains very nearly fall into this class, and there seems every reason to believe that they could have been made to fall genuinely into it without any change in their essential properties. However, the property of being 'discrete' is only an advantage for the theoretical investigator, and serves no evolutionary purpose, so we could not expect Nature to assist us by producing truly 'discrete' brains.

Given any discrete machine the first thing we wish to find out about it is the number of states (configurations) it can have. This number may be infinite (but enumerable) in which case we say that the machine has infinite memory (or storage) capacity. If the machine has a finite number $N$ of possible states then we say that it has a memory capacity of (or equivalent to) $\log_2 N$ binary digits. According to this definition we have the following table of capacities, very roughly

| | |
|---|---|
| Brunsviga | 90 |
| ENIAC without cards and with fixed programme | 600 |
| ACE as proposed | 60,000 |
| Manchester machine (as actually working 8 August 1947) | 1,100 |

The memory capacity of a machine more than anything else determines the complexity of its possible behaviour.

The behaviour of a discrete machine is completely described when we are given the state (configuration) of the machine as a function of the immediately preceding state and the relevant external data.

#### Logical computing machines (LCMs)

In Turing (1937) a certain type of discrete machine was described. It had an infinite memory capacity obtained in the form of an infinite tape marked out into squares on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol and its behaviour is in part described by that symbol, but the symbols on the tape elsewhere do not affect the behaviour of the machine. However the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings.

These machines will here be called 'Logical Computing Machines'. They are chiefly of interest when we wish to consider what a machine could in principle be designed to do, when we are willing to allow it both unlimited time and unlimited storage capacity.

*Universal logical computing machines.* It is possible to describe LCMs in a very standard way, and to put the description into a form which can be

'understood' (i.e., applied by) a special machine. In particular it is possible to design a 'universal machine' which is an LCM such that if the standard description of some other LCM is imposed on the otherwise blank tape from outside, and the (universal) machine then set going it will carry out the operations of the particular machine whose description it was given. For details the reader must refer to Turing (1937).

The importance of the universal machine is clear. We do not need to have an infinity of different machines doing different jobs. A single one will suffice. The engineering problem of producing various machines for various jobs is replaced by the office work of 'programming' the universal machine to do these jobs.

It is found in practice that LCMs can do anything that could be described as 'rule of thumb' or 'purely mechanical'. This is sufficiently well established that it is now agreed amongst logicians that 'calculable by means of an LCM' is the correct accurate rendering of such phrases. There are several mathematically equivalent but superficially very different renderings.

#### Practical computing machines (PCMs)

Although the operations which can be performed by LCMs include every rule-of-thumb process, the number of steps involved tends to be enormous. This is mainly due to the arrangement of the memory along the tape. Two facts which need to be used together may be stored very far apart on the tape. There is also rather little encouragement, when dealing with these machines, to condense the stored expressions at all. For instance the number of symbols required in order to express a number in Arabic form (e.g., 149056) cannot be given any definite bound, any more than if the numbers are expressed in the 'simplified Roman' form (IIIII...I, with 149056 occurrences of I). As the simplified Roman system obeys very much simpler laws one uses it instead of the Arabic system.

In practice however one *can* assign finite bounds to the numbers that one will deal with. For instance we can assign a bound to the number of steps that we will admit in a calculation performed with a real machine in the following sort of way. Suppose that the storage system depends on charging condensers of capacity $C=1\,\mu$f, and that we use two states of charging, $E=100$ volts and $-E=-100$ volts. When we wish to use the information carried by the condenser we have to observe its voltage. Owing to thermal agitation the voltage observed will always be slightly wrong, and the probability of an error between $V$ and $V-\mathrm{d}V$ volts is

$$\frac{2kT}{\pi C}\, e^{-\frac{1}{2}V^2C/kT}\, V\mathrm{d}V$$

where $k$ is Boltzmann's constant. Taking the values suggested we find that the probability of reading the sign of the voltage wrong is about $10^{-1\cdot2\times10^{16}}$. If then a job took more than $10^{10^{17}}$ steps we should be virtually certain of

getting the wrong answer, and we may therefore restrict ourselves to jobs with fewer steps. Even a bound of this order might have useful simplifying effects. More practical bounds are obtained by assuming that a light wave must travel at least 1 cm between steps (this would only be false with a very small machine), and that we could not wait more than 100 years for an answer. This would give a limit of $10^{20}$ steps. The storage capacity will probably have a rather similar bound, so that we could use sequences of 20 decimal digits for describing the position in which a given piece of data was to be found, and this would be a really valuable possibility.

Machines of the type generally known as 'Automatic Digital Computing Machines' often make great use of this possibility. They also usually put a great deal of their stored information in a form very different from the tape form. By means of a system rather reminiscent of a telephone exchange it is made possible to obtain a piece of information almost immediately by 'dialling' the position of this information in the store. The delay may be only a few microseconds with some systems. Such machines will be described as 'Practical Computing Machines'.

*Universal practical computing machines.* Nearly all of the PCMS now under construction have the essential properties of the 'Universal Logical Computing Machines' mentioned earlier. In practice, given any job which could have been done on an LCM one can also do it on one of these digital computers. I do not mean that we can do any required job of the type mentioned on it by suitable programming. The programming is pure paper work. It naturally occurs to one to ask whether, e.g., the ACE would be truly universal if its memory capacity were infinitely extended. I have investigated this question, and the answer appears to be as follows, though I have not proved any formal mathematical theorem about it. As has been explained, the ACE at present uses finite sequences of digits to describe positions in its memory: they are actually sequences of 9 binary digits (September 1947). The ACE also works largely for other purposes with sequences of 32 binary digits. If the memory were extended, e.g., to 1000 times its present capacity, it would be natural to arrange the memory in blocks of nearly the maximum capacity which can be handled with the 9 digits, and from time to time to switch from block to block. A relatively small part would never be switched. This would contain some of the more fundamental instruction tables and those concerned with switching. This part might be called the 'central part'. One would then need to have a number which described which block was in action at any moment. However this number might be as large as one pleased. Eventually the point would be reached where it could not be stored in a word (32 digits), or even in the central part. One would then have to set aside a block for storing the number, or even a sequence of blocks, say blocks 1, 2, . . . $n$. We should then have to store $n$, and in theory it would be of indefinite size. This sort of process can be extended in all sorts of ways, but we shall always be left with a positive integer which is of indefinite size

and which needs to be stored somewhere, and there seems to be no way out of the difficulty but to introduce a 'tape'. But once this has been done, and since we are only trying to prove a theoretical result, one might as well, whilst proving the theorem, ignore all the other forms of storage. One will in fact have a ULCM with some complications. This in effect means that one will not be able to prove any result of the required kind which gives any intellectual satisfaction.

#### Paper machines

It is possible to produce the effect of a computing machine by writing down a set of rules of procedure and asking a man to carry them out. Such a combination of a man with written instructions will be called a 'Paper Machine'. A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine. The expression 'paper machine' will often be used below.

#### Partially random and apparently partially random machines

It is possible to modify the above described types of discrete machines by allowing several alternative operations to be applied at some points, the alternatives to be chosen by a random process. Such a machine will be described as 'partially random'. If we wish to say definitely that a machine is not of this kind we will describe it as 'determined'. Sometimes a machine may be strictly speaking determined but appear superficially as if it were partially random. This would occur if for instance the digits of the number $\pi$ were used to determine the choices of a partially random machine, where previously a dice thrower or electronic equivalent had been used. These machines are known as apparently partially random.

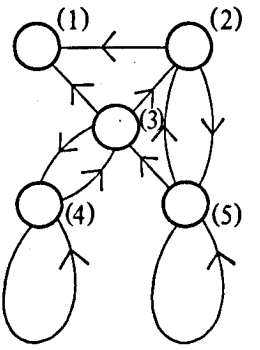
### UNORGANIZED MACHINES

So far we have been considering machines which are designed for a definite purpose (though the universal machines are in a sense an exception). We might instead consider what happens when we make up a machine in a comparatively unsystematic way from some kind of standard components. We could consider some particular machine of this nature and find out what sort of things it is likely to do. Machines which are largely random in their construction in this way will be called 'Unorganized Machines'. This does not pretend to be an accurate term. It is conceivable that the same machine might be regarded by one man as organized and by another as unorganized.

A typical example of an unorganized machine would be as follows. The machine is made up from a rather large number $N$ of similar units. Each unit has two input terminals, and has an output terminal which can be connected to the input terminals of (0 or more) other units. We may imagine that for each integer $r$, $1 \leqslant r \leqslant N$ two numbers $i(r)$ and $j(r)$ are chosen at random

from 1 . . . $N$ and that we connect the inputs of unit $r$ to the outputs of units $(r)$ and $j(r)$. All of the units are connected to a central synchronizing unit from which synchronizing pulses are emitted at more or less equal intervals of time. The times when these pulses arrive will be called 'moments'. Each unit is capable of having two states at each moment. These states may be called 0 and 1. The state is determined by the rule that the states of the units from which the input leads come are to be taken at the previous moment, multiplied together and the result subtracted from 1. An unorganized machine of this character is shown in the diagram below.

| r | i(r) | j(r) |
|---|---|---|
| 1 | 3 | 2 |
| 2 | 3 | 5 |
| 3 | 4 | 5 |
| 4 | 3 | 4 |
| 5 | 2 | 5 |



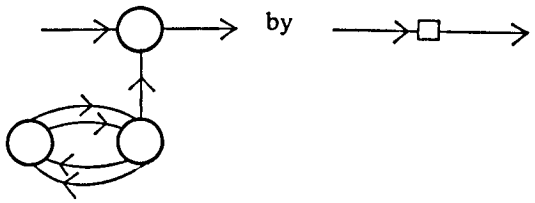

A sequence of six possible consecutive conditions for the whole machine is:

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 1 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 1 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 | 1 | 0 |

The behaviour of a machine with so few units is naturally very trivial. However, machines of this character can behave in a very complicated manner when the number of units is large. We may call these A-type unorganized machines. Thus the machine in the diagram is an A-type unorganized machine of 5 units. The motion of an A-type machine with $N$ units is of course eventually periodic, as in any determined machine with finite memory capacity. The period cannot exceed $2^N$ moments, nor can the length of time before the periodic motion begins. In the example above the period is 2 moments and there are 3 moments before the periodic motion begins. $2^N$ is 32.

The A-type unorganized machines are of interest as being about the simplest model of a nervous system with a random arrangement of neurons. It would therefore be of very great interest to find out something about their behaviour. A second type of unorganized machine will now be described, not because it is

of any great intrinsic importance, but because it will be useful later for illustrative purposes. Let us denote the circuit




as an abbreviation. Then for each A-type unorganized machine we can construct another machine by replacing each connection ——→—— in it by ——→□——→. The resulting machines will be called B-type unorganized machines. It may be said that the B-type machines are all A-type. To this I would reply that the above definitions if correctly (but drily!) set out would take the form of describing the probability of an A- (or B-) type machine belonging to a given set; it is not merely a definition of which are the A-type machines and which are the B-type machines. If one chooses an A-type machine, with a given number of units, at random, it will be extremely unlikely that one will get a B-type machine.

It is easily seen that the connection ——→□——→ can have three conditions. It may (i) pass all signals through with interchange of 0 and 1, or (ii) it may convert all signals into 1, or again (iii) it may act as in (i) and (ii) in alternate moments. (Alternative (iii) has two sub-cases.) Which of these cases applies depends on the initial conditions. There is a delay of two moments in going through ——→□——→.

## INTERFERENCE WITH MACHINERY. MODIFIABLE AND SELF-MODIFYING MACHINERY

The types of machine that we have considered so far are mainly ones that are allowed to continue in their own way for indefinite periods without interference from outside. The universal machines were an exception to this, in that from time to time one might change the description of the machine which is being imitated. We shall now consider machines in which such interference is the rule rather than the exception.

We may distinguish two kinds of interference. There is the extreme form in which parts of the machine are removed and replaced by others. This may be described as 'screwdriver interference'. At the other end of the scale is 'paper interference', which consists in the mere communication of information to the machine, which alters its behaviour. In view of the properties of the universal machine we do not need to consider the difference between these

two kinds of machine as being so very radical after all. Paper interference when applied to the universal machine can be as useful as screwdriver interference.

We shall mainly be interested in paper interference. Since screwdriver interference can produce a completely new machine without difficulty there is rather little to be said about it. In future 'interference' will normally mean 'paper interference'.

When it is possible to alter the behaviour of a machine very radically we may speak of the machine as being 'modifiable'. This is a relative term. One machine may be spoken of as being more modifiable than another.

One may also sometimes speak of a machine modifying itself, or of a machine changing its own instructions. This is really a nonsensical form of phraseology, but is convenient. Of course, according to our conventions the 'machine' is completely described by the relation between its possible configurations at consecutive moments. It is an abstraction which, by the form of its definition, cannot change in time. If we consider the machine as starting in a particular configuration, however, we may be tempted to ignore those configurations which cannot be reached without interference from it. If we do this we should get a 'successor relation' for the configurations with different properties from the original one and so a different 'machine'.

If we now consider interference, we should say that each time interference occurs the machine is probably changed. It is in this sense that interference 'modifies' a machine. The sense in which a machine can modify itself is even more remote. We may, if we wish, divide the operations of the machine into two classes, normal and self-modifying operations. So long as only normal operations are performed we regard the machine as unaltered. Clearly the idea of 'self-modification' will not be of much interest except where the division of operations into the two classes is made very carefully. The sort of case I have in mind is a computing machine like the ACE where large parts of the storage are normally occupied in holding instruction tables. (Instruction tables are the equivalent in UPCMs of descriptions of machines in ULCMs). Whenever the content of this storage was altered by the internal operations of the machine, one would naturally speak of the machine 'modifying itself'.

## MAN AS A MACHINE

A great positive reason for believing in the possibility of making thinking machinery is the fact that it is possible to make machinery to imitate any small part of a man. That the microphone does this for the ear, and the television camera for the eye are commonplaces. One can also produce remote-controlled robots whose limbs balance the body with the aid of servo-mechanisms. Here we are chiefly interested in the nervous system. We could produce fairly accurate electrical models to copy the behaviour of nerves, but there seems very little point in doing so. It would be rather like

putting a lot of work into cars which walked on legs instead of continuing to use wheels. The electrical circuits which are used in electronic computing machinery seem to have the essential properties of nerves. They are able to transmit information from place to place, and also to store it. Certainly the nerve has many advantages. It is extremely compact, does not wear out (probably for hundreds of years if kept in a suitable medium!) and has a very low energy consumption. Against these advantages the electronic circuits have only one counter-attraction, that of speed. This advantage is, however, on such a scale that it may possibly outweigh the advantages of the nerve.

One way of setting about our task of building a 'thinking machine' would be to take a man as a whole and to try to replace all the parts of him by machinery. He would include television cameras, microphones, loudspeakers, wheels and 'handling servo-mechanisms' as well as some sort of 'electronic brain'. This would be a tremendous undertaking of course. The object, if produced by present techniques, would be of immense size, even if the 'brain' part were stationary and controlled the body from a distance. In order that the machine should have a chance of finding things out for itself it should be allowed to roam the countryside, and the danger to the ordinary citizen would be serious. Moreover even when the facilities mentioned above were provided, the creature would still have no contact with food, sex, sport and many other things of interest to the human being. Thus although this method is probably the 'sure' way of producing a thinking machine it seems to be altogether too slow and impracticable.

Instead we propose to try and see what can be done with a 'brain' which is more or less without a body providing, at most, organs of sight, speech, and hearing. We are then faced with the problem of finding suitable branches of thought for the machine to exercise its powers in. The following fields appear to me to have advantages:

(i) Various games, e.g., chess, noughts and crosses, bridge, poker

(ii) The learning of languages

(iii) Translation of languages

(iv) Cryptography

(v) Mathematics.

Of these (i), (iv), and to a lesser extent (iii) and (v) are good in that they require little contact with the outside world. For instance in order that the machine should be able to play chess its only organs need be 'eyes' capable of distinguishing the various positions on a specially made board, and means for announcing its own moves. Mathematics should preferably be restricted to branches where diagrams are not much used. Of the above possible fields the learning of languages would be the most impressive, since it is the most human of these activities. This field seems however to depend rather too much on sense organs and locomotion to be feasible.

The field of cryptography will perhaps be the most rewarding. There is a remarkably close parallel between the problems of the physicist and those of the cryptographer. The system on which a message is enciphered corresponds to the laws of the universe, the intercepted messages to the evidence available, the keys for a day or a message to important constants which have to be determined. The correspondence is very close, but the subject matter of cryptography is very easily dealt with by discrete machinery, physics not so easily.

## EDUCATION OF MACHINERY

Although we have abandoned the plan to make a 'whole man', we should be wise to sometimes compare the circumstances of our machine with those of a man. It would be quite unfair to expect a machine straight from the factory to compete on equal terms with a university graduate. The graduate has had contact with human beings for twenty years or more. This contact has been modifying his behaviour pattern throughout that period. His teachers have been intentionally trying to modify it. At the end of the period a large number of standard routines will have been superimposed on the original pattern of his brain. These routines will be known to the community as a whole. He is then in a position to try out new combinations of these routines, to make slight variations on them, and to apply them in new ways.

We may say then that in so far as a man is a machine he is one that is subject to very much interference. In fact interference will be the rule rather than the exception. He is in frequent communication with other men, and is continually receiving visual and other stimuli which themselves constitute a form of interference. It will only be when the man is 'concentrating' with a view to eliminating these stimuli or 'distractions' that he approximates a machine without interference.

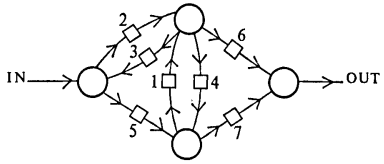
We are chiefly interested in machines with comparatively little interference, for reasons given in the last section, but it is important to remember that although a man when concentrating may behave like a machine without interference, his behaviour when concentrating is largely determined by the way he has been conditioned by previous interference.

If we are trying to produce an intelligent machine, and are following the human model as closely as we can, we should begin with a machine with very little capacity to carry out elaborate operations or to react in a disciplined manner to orders (taking the form of interference). Then by applying appropriate interference, mimicking education, we should hope to modify the machine until it could be relied on to produce definite reactions to certain commands. This would be the beginning of the process. I will not attempt to follow it further now.

## ORGANIZING UNORGANIZED MACHINERY

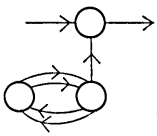
Many unorganized machines have configurations such that if once that configuration is reached, and if the interference thereafter is appropriately

restricted, the machine behaves as one organized for some definite purpose. For instance, the B-type machine shown below was chosen at random.

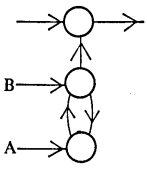



If the connections numbered 1, 3, 6, 4, are in condition (ii) initially and connections 2, 5, 7 are in condition (i), then the machine may be considered to be one for the purpose of passing on signals with a delay of 4 moments. This is a particular case of a very general property of B-type machines (and many other types), viz., that with suitable initial conditions they will do any required job, given sufficient time and provided the number of units is sufficient. In particular with a B-type unorganized machine with sufficient units one can find initial conditions which will make it into a universal machine with a given storage capacity. (A formal proof to this effect might be of some interest, or even a demonstration of it starting with a particular unorganized B-type machine, but I am not giving it as it lies rather too far outside the main argument.)

With these B-type machines the possibility of interference which could set in appropriate initial conditions has not been arranged for. It is however not difficult to think of appropriate methods by which this could be done. For instance instead of the connection



one might use



Here *A*, *B* are interfering inputs, normally giving the signal '1'. By supplying appropriate other signals at *A*, *B* we can get the connection into condition (i) or (ii), as desired. However this requires two special interfering inputs for each connection.

We shall be interested mainly in cases where there are only quite few independent inputs altogether, so that all the interference which sets up the 'initial conditions' of the machine has to be provided through one or two inputs. The process of setting up these initial conditions so that the machine will carry out some particular useful task may be called 'organizing the machine'. 'Organizing' is thus a form of 'modification'.

## THE CORTEX AS AN UNORGANIZED MACHINE

Many parts of a man's brain are definite nerve circuits required for quite definite purposes. Examples of these are the 'centres' which control respiration, sneezing, following moving objects with the eyes, etc.: all the reflexes proper (not 'conditioned') are due to the activities of these definite structures in the brain. Likewise the apparatus for the more elementary analysis of shapes and sounds probably comes into this category. But the more intellectual activities of the brain are too varied to be managed on this basis. The difference between the languages spoken on the two sides of the Channel is not due to difference in development of the French-speaking and English-speaking parts of the brain. It is due to the linguistic parts having been subjected to different training. We believe then that there are large parts of the brain, chiefly in the cortex, whose function is largely indeterminate. In the infant these parts do not have much effect: the effect they have is uncoordinated. In the adult they have great and purposive effect: the form of this effect depends on the training in childhood. A large remnant of the random behaviour of infancy remains in the adult.

All of this suggests that the cortex of the infant is an unorganized machine, which can be organized by suitable interfering training. The organizing might result in the modification of the machine into a universal machine or something like it. This would mean that the adult will obey orders given in appropriate language, even if they were very complicated; he would have no common sense, and would obey the most ridiculous orders unflinchingly. When all his orders had been fulfilled he would sink into a comatose state or perhaps obey some standing order, such as eating. Creatures not unlike this can really be found, but most people behave quite differently under many circumstance. However the resemblance to a universal machine is still very great, and suggests to us that the step from the unorganized infant to a universal machine is one which should be understood. When this has been mastered we shall be in a far better position to consider how the organizing process might have been modified to produce a more normal type of mind.

This picture of the cortex as an unorganized machine is very satisfactory

from the point of view of evolution and genetics. It clearly would not require any very complex system of genes to produce something like the A- or B-type unorganized machine. In fact this should be much easier than the production of such things as the respiratory centre. This might suggest that intelligent races could be produced comparatively easily. I think this is wrong because the possession of a human cortex (say) would be virtually useless if no attempt was made to organize it. Thus if a wolf by a mutation acquired a human cortex there is little reason to believe that he would have any selective advantage. If however the mutation occurred in a milieu where speech had developed (parrot-like wolves), and if the mutation by chance had well permeated a small community, then some selective advantage might be felt. It would then be possible to pass information on from generation to generation. However this is all rather speculative.

## EXPERIMENTS IN ORGANIZING: PLEASURE-PAIN SYSTEMS

It is interesting to experiment with unorganized machines admitting definite types of interference and try to organize them, e.g., to modify them into universal machines.

The organization of a machine into a universal machine would be most impressive if the arrangements of interference involve very few inputs. The training of the human child depends largely on a system of rewards and punishments, and this suggests that it ought to be possible to carry through the organizing with only two interfering inputs, one for 'pleasure' or 'reward' (R) and the other for 'pain' or punishment' (P). One can devise a large number of such 'pleasure–pain' systems. I will use this term to mean an unorganized machine of the following general character: The configurations of the machine are described by two expressions, which we may call the character-expression and the situation-expression. The character and situation at any moment, together with the input signals, determine the character and situation at the next moment. The character may be subject to some random variation. Pleasure interference has a tendency to fix the character, i.e., towards preventing it changing, whereas pain stimuli tend to disrupt the character, causing features which had become fixed to change, or to become again subject to random variation.

This definition is probably too vague and general to be very helpful. The idea is that when the 'character' changes we like to think of it as a change in the machine, but the 'situation' is merely the configuration of the machine described by the character. It is intended that pain stimuli occur when the machine's behaviour is wrong, pleasure stimuli when it is particularly right. With appropriate stimuli on these lines, judiciously operated by the 'teacher', one may hope that the 'character' will converge towards the one desired, i.e., that wrong behaviour will tend to become rare.

I have investigated a particular type of pleasure–pain system, which I will now describe.

## THE P-TYPE UNORGANIZED MACHINE

The P-type machine may be regarded as an LCM without a tape, and whose description is largely incomplete. When a configuration is reached, for which the action is undetermined, a random choice for the missing data is made and the appropriate entry is made in the description, tentatively, and is applied. When a pain stimulus occurs all tentative entries are cancelled, and when a pleasure stimulus occurs they are all made permanent.

Specifically. The situation is a number $s=1, 2, \ldots, N$ and corresponds to the configuration of the incomplete machine. The character is a table of $N$ entries showing the behaviour of the machine in each situation. Each entry has to say something both about the next situation and about what action the machine has to take. The action part may be either

(i) To do some externally visible act $A_1$ or $A_2 \ldots A_K$

(ii) To set one of the memory units $M_1 \ldots M_R$ either into the '1' condition or into the '0' condition.

The next situation is always the remainder either of $2s$ or of $2s+1$ on division by $N$. These may be called alternatives 0 and 1. Which alternative applies may be determined by either

(a) one of the memory units

(b) a sense stimulus

(c) the pleasure–pain arrangements.

In each situation it is determined which of these applies when the machine is made, i.e., interference cannot alter which of the three cases applies. Also in cases (a) and (b) interference can have no effect. In case (c) the entry in the character table may be either U ('uncertain'), or T0 (tentative 0), T1, D0 (definite 0) or D1. When the entry in the character for the current situation is U then the alternative is chosen at random, and the entry in the character is changed to T0 or T1 according as 0 or 1 was chosen. If the character entry was T0 or D0 then the alternative is 0 and if it is T1 or D1 then the alternative is 1. The changes in character include the above mentioned change from U to T0 or T1, and a change of every T to D when a pleasure stimulus occurs, changes of T0 and T1 to U when a pain stimulus occurs.

We may imagine the memory units essentially as 'trigger circuits' or switches. The sense stimuli are means by which the teacher communicates 'unemotionally' to the machine, i.e., otherwise than by pleasure and pain stimuli. There are a finite number $S$ of sense stimulus lines, and each always carries either the signal 0 or 1.

A small P-type machine is described in the table below

| | | | |
|---|---|---|---|
| 1 | P | A | |
| 2 | P | B | M1=1 |
| 3 | P | B | |
| 4 | S1 | A | M1=0 |
| 5 | M1 | C | |

In this machine there is only one memory unit M1 and one sense line S1. Its behaviour can be described by giving the successive situations together with the actions of the teacher: the latter consist of the values of S1 and the rewards and punishments. At any moment the 'character' consists of the above table with each 'P' replaced by either U, T0, D0 or D1. In working out the behaviour of the machine it is convenient first of all to make up a sequence of random digits for use when the U cases occur. Underneath these we may write the sequence of situations, and have other rows for the corresponding entries from the character, and for the actions of the teacher. The character and the values stored in the memory units may be kept on another sheet. The T entries may be made in pencil and the D entries in ink. A bit of the behaviour of the machine is given below:

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Random sequence | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| Situations | 3 | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 2 | 4 | 4 | 4 | 3 | 2 | . | . | | |
| Alternative given by | | U | T | T | T | T | T | U | U | S | S | S | U | T | | | | |
| | | | 0 | 0 | 0 | 0 | 0 | | | 1 | 1 | 1 | | 0 | | | | |
| Visible action | B | A | B | A | B | A | B | A | B | A | A | A | B | B | | | | |
| Rew. & Pun. | | | | | | | P | | | | | | | | | | | |
| Changes in S1 | 1 | | | | | | | | | | | 0 | | | | | | |

It will be noticed that the machine very soon got into a repetitive cycle. This became externally visible through the repetitive BABAB.... By means of a pain stimulus this cycle was broken.

It is probably possible to organize these P-type machines into universal machines, but it is not easy because of the form of memory available. It would be necessary to organize the randomly distributed 'memory units' to provide a systematic form of memory, and this would not be easy. If, however, we supply the P-type machine with a systematic external memory this organizing becomes quite feasible. Such a memory could be provided in the form of a tape, and the externally visible operations could include movement to right and left along the tape, and altering the symbol on the tape to 0 or to 1. The sense lines could include one from the symbol on the tape. Alternatively, if the memory were to be finite, e.g., not more than $2^{32}$ binary digits, we could use a dialling system. (Dialling systems can also be used with an infinite memory, but this is not of much practical interest.) I have succeeded in organizing such a (paper) machine into a universal machine.

The details of the machine involved were as follows. There was a circular memory consisting of 64 squares of which at any moment one was in the machine ('scanned') and motion to right or left were among the 'visible actions'. Changing the symbol on the square was another 'visible action', and the symbol was connected to one of the sense lines S1. The even-numbered squares also had another function, they controlled the dialling of information to or from the main memory. This main memory consisted of $2^{32}$ binary

digits. At any moment one of these digits was connected to the sense line S2. The digit of the main memory concerned was that indicated by the 32 even positioned digits of the circular memory. Another two of the 'visible actions' were printing 0 or 1 in this square of the main memory. There were also three ordinary memory units and three sense units S3, S4, S5. Also six other externally visible actions A,B,C,D,E,F.

This P-type machine with external memory has, it must be admitted, considerably more 'organization' than say the A-type unorganized machine. Nevertheless the fact that it can be organized into a universal machine still remains interesting.

The actual technique by which the 'organizing' of the P-type machine was carried through is perhaps a little disappointing. It is not sufficiently analogous to the kind of process by which a child would really be taught. The process actually adopted was first to let the machine run for a long time with continuous application of pain, and with various changes of the sense data S3, S4, S5. Observation of the sequence of externally visible actions for some thousands of moments made it possible to set up a scheme for identifying the situations, i.e., by which one could at any moment find out what the situation was, except that the situations as a whole had been renamed. A similar investigation, with less use of punishment, enables one to find the situations which are affected by the sense lines; the data about the situations involving the memory units can also be found but with more difficulty. At this stage the character has been reconstructed. There are no occurrences of T0, T1, D0, D1. The next stage is to think up some way of replacing the 0s of the character by D0, D1 in such a way as to give the desired modification. This will normally be possible with the suggested number of situations (1000), memory units, etc. The final stage is the conversion of the character into the chosen one. This may be done simply by allowing the machine to wander at random through a sequence of situations, and applying pain stimuli when the wrong choice is made, pleasure stimuli when the right one is made. It is best also to apply pain stimuli when irrelevant choices are made. This is to prevent getting isolated in a ring of irrelevant situations. The machine is now 'ready for use'.

The form of universal machine actually produced in this process was as follows. Each instruction consisted of 128 digits, which we may regard as forming four sets of 32, each of which describes one place in the main memory. These places may be called P,Q,R,S. The meaning of the instruction is that if $p$ is the digit at P and $q$ that at Q then $1-pq$ is to be transferred to position R and that the next instruction will be found in the 128 digits beginning at S. This gives a UPCM, though with rather less facilities than are available say on the ACE.

I feel that more should be done on these lines. I would like to investigate other types of unorganized machines, and also to try out organizing methods that would be more nearly analogous to our 'methods of education'. I made

a start on the latter but found the work altogether too laborious at present. When some electronic machines are in actual operation I hope that they will make this more feasible. It should be easy to make a model of any particular machine that one wishes to work on within such a UPCM instead of having to work with a paper machine as at present. If also one decided on quite definite 'teaching policies' these could also be programmed into the machine. One would then allow the whole system to run for an appreciable period, and then break in as a kind of 'inspector of schools' and see what progress had been made. One might also be able to make some progress with unorganized machines more like the A- and B-types. The work involved with these is altogether too great for pure paper-machine work.

One particular kind of phenomenon I had been hoping to find in connection with the P-type machines. This was the incorporation of old routines into new. One might have 'taught' (i.e., modified or organized) a machine to add (say). Later one might teach it to multiply by small numbers by repeated addition and so arrange matters that the same set of situations which formed the addition routine, as originally taught, was also used in the additions involved in the multiplication. Although I was able to obtain a fairly detailed picture of how this might happen I was not able to do experiments on a sufficient scale for such phenomena to be seen as part of a large context.

I also hoped to find something rather similar to the 'irregular verbs' which add variety to language. We seem to be quite content that things should not obey too mathematically regular rules. By long experience we can pick up and apply the most complicated rules without being able to enunciate them at all. I rather suspect that a P-type machine without the systematic memory would behave in a rather similar manner because of the randomly distributed memory units. Clearly this could only be verified by very painstaking work; by the very nature of the problem 'mass production' methods like built-in teaching procedures could not help.

## DISCIPLINE AND INITIATIVE

If the untrained infant's mind is to become an intelligent one, it must acquire both discipline and initiative. So far we have been considering only discipline. To convert a brain or machine into a universal machine is the extremest form of discipline. Without something of this kind one cannot set up proper communication. But discipline is certainly not enough in itself to produce intelligence. That which is required in addition we call initiative. This statement will have to serve as a definition. Our task is to discover the nature of this residue as it occurs in man, and to try and copy it in machines.

Two possible methods of setting about this present themselves. On the one hand we have fully disciplined machines immediately available, or in a matter of months or years, in the form of various UPCMs. We might try to graft some initiative onto these. This would probably take the form of programming the machine to do every kind of job that could be done, as a

matter of principle, whether it were economical to do it by machine or not. Bit by bit one would be able to allow the machine to make more and more 'choices' or 'decisions'. One would eventually find it possible to program it so as to make its behaviour be the logical result of a comparatively small number of general principles. When these became sufficiently general, interference would no longer be necessary, and the machine would have 'grown up'. This may be called the 'direct method'.

The other method is to start with an unorganized machine and to try to bring both discipline and initiative into it at once, i.e., instead of trying to organize the machine to become a universal machine, to organize it for initiative as well. Both methods should, I think, be attempted.

#### Intellectual, genetical and cultural searches

A very typical sort of problem requiring some sort of initiative consists of those of the form 'Find a number $n$ such that . . .'. This form covers a very great variety of problems. For instance problems of the form 'See if you can find a way of calculating the function which will enable us to obtain the values for arguments . . . to accuracy . . . within a time . . . using the UPCM . . .' are reducible to this form, for the problem is clearly equivalent to that of finding a program to put on the machine in question, and it is easy to put the programs into correspondence with the positive integers in such a way that given either the number or the program the other can easily be found. We should not go far wrong for the time being if we assumed that all problems were reducible to this form. It will be time to think again when something turns up which is obviously not of this form.

The crudest way of dealing with such a problem is to take the integers in order and to test each one to see whether it has the required property, and to go on until one is found which has it. Such a method will only be successful in the simplest cases. For instance in the case of problems of the kind mentioned above, where one is really searching for a program, the number required will normally be somewhere between $2^{1000}$ and $2^{1,000,000}$. For practical work therefore some more expeditious method is necessary. In a number of cases the following method would be successful. Starting with a UPCM we first put a program into it which corresponds to building in a logical system (like Russell's *Principia Mathematica*). This would not determine the behaviour of the machine completely: at various stages more than one choice as to the next step would be possible. We might arrange, however, to take all possible arrangement of choices in order, and go on until the machine proved a theorem, which, by its form, could be verified to give a solution of the problem. This may be seen to be a conversion of the original problem into another of the same form. Instead of searching through values of the original variable $n$ one searches through values of something else. In practice when solving problems of the above kind one will probably apply some very complex 'transformation' of the original problem, involving searching through

various variables, some more analogous to the original one, some more like a 'search through all proofs'. Further research into intelligence of machinery will probably be very greatly concerned with 'searches' of this kind. We may perhaps call such searches 'intellectual searches'. They might very briefly be defined as 'searches carried out by brains for combinations with particular properties'.

It may be of interest to mention two other kinds of search in this connection. There is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being survival value. The remarkable success of this search confirms to some extent the idea that intellectual activity consists mainly of various kinds of search.

The remaining form of search is what I should like to call the 'cultural search'. As I have mentioned, the isolated man does not develop any intellectual power. It is necessary for him to be immersed in an environment of other men, whose techniques he absorbs during the first twenty years of his life. He may then perhaps do a little research of his own and make a very few discoveries which are passed on to other men. From this point of view the search for new techniques must be regarded as carried out by the human community as a whole, rather than by individuals.

## INTELLIGENCE AS AN EMOTIONAL CONCEPT

The extent to which we regard something as behaving in an intelligent manner is determined as much by our own state of mind and training as by the properties of the object under consideration. If we are able to explain and predict its behaviour or if there seems to be little underlying plan, we have little temptation to imagine intelligence. With the same object therefore it is possible that one man would consider it as intelligent and another would not; the second man would have found out the rules of its behaviour.

It is possible to do a little experiment on these lines, even at the present stage of knowledge. It is not difficult to devise a paper machine which will play a not very bad game of chess. Now get three men as subjects for the experiment A,B,C. A and C are to be rather poor chess players, B is the operator who works the paper machine. (In order that he should be able to work it fairly fast it is advisable that he be both mathematician and chess player.) Two rooms are used with some arrangement for communicating moves, and a game is played between C and either A or the paper machine. C may find it quite difficult to tell which he is playing. (This is a rather idealized form of an experiment I have actually done.)

## REFERENCES

Church, Alonzo (1936) An unsolvable problem of elementary number theory. *Amer. J. of Math.* **58**, 345–63.

Gödel, K. (1931) Über formal unentscheidbare Satze der Principia Mathematica und verwandter Systeme. *Monatshefte fur Math. und Phys.* **38**, 173–89.

Turing, A.M. (1937) On computable numbers with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* **42**, 230–65.

# MATHEMATICAL FOUNDATIONS

# 1

# Properties of Programs and Partial Function Logic

Zohar Manna and John McCarthy
Computer Science Department
Stanford University

**Abstract**

We consider recursive definitions which consist of ALGOL-like conditional expressions. By specifying a computation rule for evaluating such a recursive definition, a partial function is determined. However, for different computation rules, the same recursive definition may determine different partial functions. We distinguish between two types of computation rules: sequential and parallel.

The purpose of this paper is to formalize properties (such as termination, correctness, and equivalence) of these partial functions by means of the satisfiability or validity of certain formulas in partial function logic.

## INTRODUCTION

Partial functions on a domain have values only for some of their arguments; for other arguments they may have no value. If the partial function $f$ has a value at argument $a$ we say that $f(a)$ is defined and write $*f(a)$. Functions of several arguments are treated similarly, including predicates – i.e., functions taking the values $T$ (true) and $F$ (false). We include as limiting cases of partial functions, the partial functions defined for all arguments (called *total* functions) as well as the partial function undefined for all arguments.

Partial functions arise naturally in connection with computation because a computing process may give a result for some arguments and for others run on indefinitely. Any attempt to get rid of the theory of partial functions and keep all the total computable functions fails because it is undecidable whether a function is total or even whether it is defined for particular arguments.

This circumstance suggests studying a first order logic of partial functions analogous to the usual first order logic of total functions (*see*, for example, Hayes 1969, Kleene 1952, McCarthy 1963a, Wang 1961). Partial functions defined by computing processes can be substituted for the function letters in

formulas of this calculus without having either to prove them total first or to extend them to total functions by some convention. This has turned out to be useful in proving the termination (also called convergence), correctness and equivalence of different computation processes, as will be described in this paper. Papers related to this application include those of McCarthy (1962, 1963b) and Manna and Pnueli (1968, 1969).

## RECURSIVE DEFINITIONS

We shall consider recursive definitions, over a given non-empty domain $D$, of the form: $f(\bar{x}) \Leftarrow \tau(\bar{x}, f)$, where $\tau$ is any ALGOL-like conditional expression which consists only of 'if–then–else' operators, known total functions, total predicates and constants over $D$, the individual variables $\bar{x} = (x_1, \ldots, x_n)$ where $n \geqslant 0$, and the function symbol $f$ itself. (We do not allow logical connectives such as $\sim, \wedge, \vee, \supset$ and $\equiv$, but this is not a real restriction, as will be explained later.)

Some well-known examples of such recursive definitions are:

(a) *Factorial function*

$$f(x) \Leftarrow \textbf{if } x=0 \textbf{ then } 1 \textbf{ else } x \cdot f(x-1),$$

where $D = \{\text{the non-negative integers}\}$.

(b) *Ackermann's function*

$$f(x_1, x_2) \Leftarrow \textbf{if } x_1=0 \textbf{ then } x_2+1 \textbf{ else if } x_2=0 \textbf{ then } f(x_1-1, 1) \textbf{ else } f(x_1-1, f(x_1, x_2-1)),$$

where $D = \{\text{the non-negative integers}\}$.

(c) *Append (Concatenation) function*

$$f(x_1, x_2) \Leftarrow \textbf{if null } (x_1) \textbf{ then } x_2 \textbf{ else cons } (\text{car } x_1, f(\text{cdr } x_1, x_2)),$$

where $D = \{\text{all lists}\}$.

In general, we can introduce a system of recursive definitions by which several partial functions are defined simultaneously. In fact, every flowchart can be expressed as such a system of recursive definitions (*see* McCarthy 1962). However, for simplicity we shall restrict ourselves in this presentation to a single recursive definition. The extension of the results to the general case is straightforward.

Let us consider the recursive definition:

$$f(x) \Leftarrow \textbf{ if } x>100 \textbf{ then } x-10 \textbf{ else } f(f(x+11)),$$

where $D = \{\text{the integers}\}$.

For a given integer $x$, we can construct a sequence of terms, called the computation sequence of $f(x)$. For example, the computation sequence of $f(99)$ is

$$f(99) \rightarrow f(f(110)) \rightarrow f(100) \rightarrow f(f(111)) \rightarrow f(101) \rightarrow 91.$$

In this case we say that $f(99)$ is defined and $f(99) = 91$.

Notice that, for example, for the recursive definition:

$$f'(x) \Leftarrow \textbf{if } x>100 \textbf{ then } x-11 \textbf{ else } f'(f'(x+11)),$$

the computation sequence of $f'(99)$ is

$$f'(99) \rightarrow f'(f'(110)) \rightarrow f'(99) \rightarrow f'(f'(110)) \rightarrow f'(99) \rightarrow \ldots,$$

i.e., the computation sequence is infinite. In this case we say that $f'(99)$ is undefined.

Thus in general the recursive definition defines $f$ as a *partial* function mapping $D^n$ into $D$. However, our definition is still vague. Consider, for example, the following simple recursive definition:

$$f(x) \Leftarrow \textbf{if } f(x)=0 \textbf{ then } 0 \textbf{ else } 0,$$

where $D=\{0\}$. While someone might say that $f(0)$ is defined and $f(0)=0$, someone else might say that $f(0)$ is undefined. The ambiguity clearly arises from the fact that we have not yet specified exactly what rules define the computation sequence.

In general, we move from step to step by first applying a substitution rule and then making all possible simplifications. However, we distinguish between two types of operations: sequential operations and parallel operations.

**Sequential operations**

1. Substitution rule.

Replace the *leftmost* occurrence of $f$ of the form $f(\xi)$, $\xi \in D^n$, by its definition, i.e., by $\tau(\xi,f)$.

2. Simplification rules.

(a) For each known function and predicate, *with arguments free of f*, write its value (i.e., an element of $D$ for a function and $T$ or $F$ for a predicate).

(b) Replace (without computing $B$ at all)

'**if** $T$ **then** $A$ **else** $B$' by '$A$'

'**if** $F$ **then** $B$ **else** $A$' by '$A$'.

**Parallel operations**

1. Substitution rule.

Replace *every* occurrence of $f$ of the form $f(\xi)$, $\xi \in D^n$, by its definition, i.e., by $\tau(\xi,f)$.

2. Simplification rules.

(a) As above, i.e., for each known function and predicate, *with arguments free of f*, write its value.

(*b*) Replace (without finishing the computation of $B$)

'**if** $T$ **then** $A$ **else** $B$' by '$A$'

'**if** $F$ **then** $B$ **else** $A$' by '$A$'

'**if** $B$ **then** $A$ **else** $A$' by '$A$'.

We are ready now to introduce more precisely our notion of computation using recursive definitions.

Let $f(\bar{x}) \Leftarrow \tau(\bar{x}, f)$ be a given recursive definition over $D$.

For every $\xi \in D^n$ one can generate a sequence of terms $\tau_0, \tau_1, \tau_2, \ldots$, called the $\left\{\begin{matrix} s \\ p \end{matrix}\right\}$-*computation sequence of* $f(\xi)$ (we are defining two notions at once: $s$-computation sequence and $p$-computation sequence), according to the following rules:

1. $\tau_0$ is $\tau(\xi, f)$, after all possible $\left\{\begin{matrix} \text{sequential} \\ \text{parallel} \end{matrix}\right\}$ simplifications;
2. $\tau_{i+1}$, $i \geqslant 0$, is obtained from $\tau_i$ by applying first the $\left\{\begin{matrix} \text{sequential} \\ \text{parallel} \end{matrix}\right\}$ substitution rule on $\tau_i$, and then applying all possible $\left\{\begin{matrix} \text{sequential} \\ \text{parallel} \end{matrix}\right\}$ simplifications;
3. The sequence is finite and $\tau_k$ is the final term in the sequence if and only if $\tau_k$ does not contain any occurrence of $f$, or, in other words, $\tau_k = \eta$ where $\eta \in D$. In this case, we say that $\left\{\begin{matrix} f_s(\xi) \\ f_p(\xi) \end{matrix}\right\}$ *is defined and its value is* $\eta$. Otherwise, i.e., if the sequence is infinite, we say that $\left\{\begin{matrix} f_s(\xi) \\ f_p(\xi) \end{matrix}\right\}$ *is undefined.*

Thus, the unique (!) partial function (mapping $D^n$ into $D$) obtained by applying only $\left\{\begin{matrix} \text{sequential} \\ \text{parallel} \end{matrix}\right\}$ operations is denoted by $\left\{\begin{matrix} f_s \\ f_p \end{matrix}\right\}$. Note that for a recursive definition in which all the occurrences of the 'if–then–else' operators are of the form '**if** $A$ **then** $B$ **else** $C$', where $A$ is free of $f$s, $f_s$ and $f_p$ are identical.

Let $f$ and $f'$ be any two partial functions (mapping $D^n$ into $D$). $f'$ is called an *extension* of $f$ if for every $\xi \in D^n$: if $f(\xi)$ is defined then $f'(\xi)$ is also defined and $f'(\xi) = f(\xi)$.

Note that, for the same recursive definition, $f_p$ is always an extension of $f_s$, but not necessarily conversely. For example, consider again the recursive definition:

$$f(x) \Leftarrow \textbf{if } f(x) = 0 \textbf{ then } 0 \textbf{ else } 0,$$

where $D = \{0\}$. $f_s(0)$ is undefined, but $f_p(0)$ is clearly defined and $f_p(0) = 0$. However, for the recursive definition:

$$f(x) \Leftarrow \textbf{if } f(x) = 0 \textbf{ then } f(x) \textbf{ else } 0,$$

where $D = \{0\}$, both $f_s(0)$ and $f_p(0)$ are undefined. This example clearly suggests that there are still other computation rules intuitively appealing, defining partial functions which are extensions of $f_p$.

The purpose of this paper is to formalize properties of $f_s$ and $f_p$ using a variant of partial function logic.

## PARTIAL FUNCTION LOGIC

Every domain $D$ can be extended to $D^+ = D \cup \{U\}$, where $U$ denotes the distinct 'undefined' element. In particular the set of two truth-values $\tau = \{T, F\}$ is extended to be the set of three truth-values $\tau^+ = \{T, F, U\}$.

The usual connectives (such as $\sim$, $\supset$, $\wedge$, $\vee$ and $\equiv$) and quantifiers ($\exists$ and $\forall$) will always be applied in our calculus to the restricted domains (i.e., without the 'undefined' element) and thus have the same meaning as usual. However, we have two additional connectives and two additional operators applied to the extended domains:

1. $*a: D^+ \to \tau$

is defined as $\begin{cases} T \text{ if } a \in D \\ F \text{ otherwise (i.e., if } a \text{ is } U). \end{cases}$

2. $a \overset{*}{=} b: D^+ \times D^+ \to \tau$ (*extended equivalence*)

is defined as $\begin{cases} T \text{ if } a, b \in D \text{ and } a = b \\ \quad \text{or if } a \text{ and } b \text{ are } U \\ F \text{ otherwise.} \end{cases}$

Thus the extended equivalence relation ($\overset{*}{=}$) over $D^+$ is just the natural extension of the regular equivalence relation ($=$) over $D$. (Note that we are using the regular equality sign ($=$) both in the metalanguage and to indicate equality of elements in the restricted domain. However, we are sure that this will cause no confusion.) Using the * operator, $a \overset{*}{=} b$ can be defined as

$$(*a \vee *b) \supset (*a \wedge *b \wedge a = b),$$

or equivalently

$$(*a \equiv *b) \wedge (*a \supset a = b).$$

3. (**if** $A$ **then** $a$ **else** $b)_s: \tau^+ \times D^+ \times D^+ \to D^+$

is defined as

(**if** $T$ **then** $a$ **else** $b)_s \to a$,
(**if** $F$ **then** $a$ **else** $b)_s \to b$, and
(**if** $U$ **then** $a$ **else** $b)_s \to U$.

The values of the (**if** $A$ **then** $a$ **else** $b)_s$ operator are obtained by imagining the computation of $A$, $a$ and $b$ to proceed *in sequence*, i.e., first $A$ is computed and then either $a$ or $b$ is computed. If the computation of $A$ never terminates, then the value of the compound is $U$ (we never get to compute $a$ or $b$). If the computation of $A$ terminates with value $T$ we proceed to compute $a$, and if the computation of $A$ terminates with value $F$ we proceed to compute $b$.

4. (**if** $A$ **then** $a$ **else** $b)_p: \tau^+ \times D^+ \times D^+ \to D^+$,

is defined as

(**if** $A$ **then** $a$ **else** $b)_s$, but with one exception:
(**if** $A$ **then** $a$ **else** $a)_p \to a$ for every $a \in D^+$.

Therefore, the definition is

(**if** $T$ **then** $a$ **else** $b)_p \to a$
(**if** $F$ **then** $a$ **else** $b)_p \to b$

(**if** $U$ **then** $a$ **else** $b)_p \to \begin{cases} a \text{ if } a \overset{*}{=} b \\ U \text{ otherwise.} \end{cases}$

Thus using the previous operators it can be defined as:

(**if** $a \overset{*}{=} b$ **then** $a$ **else** (**if** $A$ **then** $a$ **else** $b)_s)_s$.

The values of the **(if** $A$ **then** $a$ **else** $b)_p$ operator are obtained by imagining the computation of $A$, $a$ and $b$ to proceed *in parallel*, i.e., simultaneously. Thus if the computation of $A$ terminates with value $T$, we stop the computation of $b$ and proceed only with $a$, and if the computation of $A$ terminates with value $F$, we stop the computation of $a$ and proceed only with $b$. If the computation of $a$ and $b$ terminates before the computation of $A$ and $a=b$, we stop the computation of $A$ and the value of the computation is $a$ (or $b$).

The interesting property of the last two operators is that if we define:

$\sim A$ as **(if** $A$ **then** $F$ **else** $T$**)**,
$A \supset B$ as **(if** $A$ **then** $B$ **else** $T$**)**,
$A \vee B$ as **(if** $A$ **then** $T$ **else** $B$**)**,
$A \wedge B$ as **(if** $A$ **then** $B$ **else** $F$**)**, and
$A \equiv B$ as **(if** $A$ **then** $B$ **else (if** $B$ **then** $F$ **else** $T$**))**,

then by using the **(if–then–else)**$_s$ operators we obtain the sequential connectives whose truth tables are (see, for example, McCarthy 1963a):

| $\supset$ | T | F | U |
|---|---|---|---|
| T | T | F | U |
| F | T | T | T |
| U | (U) | U | U |

| $\vee$ | T | F | U |
|---|---|---|---|
| T | T | T | T |
| F | T | F | U |
| U | (U) | U | U |

| $\wedge$ | T | F | U |
|---|---|---|---|
| T | T | F | U |
| F | F | F | F |
| U | U | (U) | U |

| $\sim$ | |
|---|---|
| T | F |
| F | T |
| U | U |

| $\equiv$ | T | F | U |
|---|---|---|---|
| T | T | F | U |
| F | F | T | U |
| U | U | U | U |

while by using the **(if–then–else)**$_p$ operators we obtain the parallel connectives (originally introduced by Łukasiewicz (1941), *see* also Kleene (1952), Wang (1961) and Hayes (1969)), whose truth tables are given on p. 33.

Note that the truth tables of $\sim$ and $\equiv$ are the same for both types of operations, the other corresponding truth tables differ only in the circled element.

In the following sections, an occurrence of an 'if–then–else' operator not labeled by $p$ or $s$ should always be considered as **(if–then–else)**$_s$.

Our *well-formed formulas* (wffs) are constructed in the usual way. Roughly speaking, by specifying a non-empty domain $D$ and a non-negative integer $n$, and using

(a) known partial functions, partial predicates and constants over $D$,

| $\supset$ | T | F | U |
|---|---|---|---|
| T | T | F | U |
| F | T | T | T |
| U | (T) | U | U |

| $\vee$ | T | F | U |
|---|---|---|---|
| T | T | T | T |
| F | T | F | U |
| U | (T) | U | U |

| $\wedge$ | T | F | U |
|---|---|---|---|
| T | T | F | U |
| F | F | F | F |
| U | U | (F) | U |

| $\sim$ | |
|---|---|
| T | F |
| F | T |
| U | U |

| $\equiv$ | T | F | U |
|---|---|---|---|
| T | T | F | U |
| F | F | T | U |
| U | U | U | U |

(b) the individual variables $\bar{x}=(x_1, \ldots, x_n)$, and the function symbol $f$ (of $n$ arguments),
(c) regular connectives ($\sim, \wedge, \vee, \supset$ and $\equiv$) and quantifiers ($\forall x_i$ and $\exists x_i$), and
(d) the additional connectives ($*$ and $\overset{*}{=}$) and the additional operators ($(\textbf{if-then-else})_s$ and $(\textbf{if-then-else})_p$),
in such a way that after every assignment of a partial function (mapping $D^n$ into $D$) to $f$, the expression obtained has the value $T$ (true) or $F$ (false).

Such a wff $W$ is said to be:

1. *satisfiable*, if we can assign to $f$ a partial function (mapping $D^n$ into $D$) such that the value of $W$ is true;
2. *valid*, if for every assignment of a partial function (mapping $D^n$ into $D$) to $f$, the value of $W$ is true.

## THE FORMALIZATION

Consider the recursive definition $f(\bar{x}) \Leftarrow \tau(\bar{x}, f)$ over $D$. Let us denote by $\tau_s$ the result of changing all the 'if-then-else' operators in $\tau$ to $(\textbf{if-then-else})_s$ and by $\tau_p$ the result of changing all the 'if-then-else' operators in $\tau$ to $(\textbf{if-then-else})_p$. Below we shall consider the formulas (in our partial function logic):

$$\forall \bar{x}[f(\bar{x}) \overset{*}{=} \tau_s(\bar{x}, f)] \text{ and}$$
$$\forall \bar{x}[f(\bar{x}) \overset{*}{=} \tau_p(\bar{x}, f)],$$

where '$\forall \bar{x}[\ldots]$' stands for 'for all $\bar{x}$ s.t. $\bar{x} \in D^n, \ldots$'.

Recall that the unique partial function (mapping $D^n$ into $D$) obtained by applying only $\left\{\begin{matrix}\text{sequential}\\ \text{parallel}\end{matrix}\right\}$ operations was denoted by $\left\{\begin{matrix}f_s\\ f_p\end{matrix}\right\}$.

*Lemma* 1

(a) $\forall \bar{x}[f_s(\bar{x}) \overset{*}{=} \tau_s(\bar{x}, f_s)]$ is true, and
(b) $\forall \bar{x}[f_p(\bar{x}) \overset{*}{=} \tau_p(\bar{x}, f_p)]$ is true.

*Lemma* 2

For every partial function $f'$ mapping $D^n$ into $D$,
(a) $\forall \bar{x}[f'(\bar{x}) \stackrel{*}{=} \tau_s(\bar{x}, f')]$ is true $\Rightarrow f'$ is an extension of $f_s$, and
(b) $\forall \bar{x}[f'(\bar{x}) \stackrel{*}{=} \tau_p(\bar{x}, f')]$ is true $\Rightarrow f'$ is an extension of $f_p$.

While the proof of Lemma 1 is straightforward, the proof of Lemma 2 relies on induction on the length of the computation sequences.

From Lemma 2 follows immediately

*Corollary* 1 (*Recursion Induction*):
For every two partial functions $g(\bar{x})$ and $h(\bar{x})$, mapping $D^n$ into $D$,
(a) if there exists a recursive definition $f(\bar{x}) \Leftarrow \tau(\bar{x}, f)$ over $D$ such that
$\forall \bar{x}[g(\bar{x}) \stackrel{*}{=} \tau_s(\bar{x}, g)]$ is true, and
$\forall \bar{x}[h(\bar{x}) \stackrel{*}{=} \tau_s(\bar{x}, h)]$ is true,
then for every $\xi \in D^n : f_s(\xi)$ is defined $\Rightarrow g(\xi)$ and $h(\xi)$ are defined and $g(\xi) = h(\xi)$.
(b) if there exists a recursive definition $f(\bar{x}) \Leftarrow \tau(\bar{x}, f)$ over $D$, such that
$\forall \bar{x}[g(\bar{x}) \stackrel{*}{=} \tau_p(\bar{x}, g)]$ is true, and
$\forall \bar{x}[h(\bar{x}) \stackrel{*}{=} \tau_p(\bar{x}, h)]$ is true,
then for every $\xi \in D^n : f_p(\xi)$ is defined $\Rightarrow g(\xi)$ and $h(\xi)$ are defined and $g(\xi) = h(\xi)$.

*Definitions.* Given:
(a) a partial function $f(\bar{x})$ mapping $D^n$ into $D$,
(b) a total predicate $\phi(\bar{x})$ over $D^n$, called the *input predicate*, and
(c) a total predicate $\psi(\bar{x}, y)$ over $D^n \times D$, called the *output predicate*.

We say that

1. $f$ is *partially correct w.r.t.* $\phi$ *and* $\psi$ if
$\forall \xi$ s.t. $\phi(\xi) = T$, if $f(\xi)$ is defined then $\psi(\xi, f(\xi)) = T$.
2. $f$ is (*totally*) *correct w.r.t.* $\phi$ *and* $\psi$ if
$\forall \xi$ s.t. $\phi(\xi) = T$, $f(\xi)$ is defined and $\psi(\xi, f(\xi)) = T$.
3. $f$ is *defined w.r.t.* $\phi$ if
$\forall \xi$ s.t. $\phi(\xi) = T$, $f(\xi)$ is defined.

We proceed to formalize these notions for the partial functions $f_s$ and $f_p$ for a given recursive definition $f(\bar{x}) \Leftarrow \tau(\bar{x}, f)$ over $D$.

*Theorem* 1
(a) $f_s$ is partially correct w.r.t. $\phi$ and $\psi$
if and only if
$\forall \bar{x}[f(\bar{x}) \stackrel{*}{=} \tau_s(\bar{x}, f)] \wedge \forall \bar{x}[[\phi(\bar{x}) \wedge {}^*f(\bar{x})] \supset \psi(\bar{x}, f(\bar{x}))]$ is satisfiable.
(b) $f_p$ is partially correct w.r.t. $\phi$ and $\psi$
if and only if
$\forall \bar{x}[f(\bar{x}) \stackrel{*}{=} \tau_p(\bar{x}, f)] \wedge \forall \bar{x}[[\phi(\bar{x}) \wedge {}^*f(\bar{x})] \supset \psi(\bar{x}, f(\bar{x}))]$ is satisfiable.

*Theorem* 2
(a) $f_s$ is correct w.r.t. $\phi$ and $\psi$
if and only if
$\forall \bar{x}[f(\bar{x}) \stackrel{*}{=} \tau_s(\bar{x}, f)] \supset \forall \bar{x}[\phi(\bar{x}) \supset [{}^*f(\bar{x}) \wedge \psi(\bar{x}, f(\bar{x}))]]$ is valid.

(b) $f_p$ is correct w.r.t. $\phi$ and $\psi$
if and only if
$\forall \bar{x}[f(\bar{x}) \stackrel{*}{=} \tau_p(\bar{x}, f)] \supset \forall \bar{x}[\phi(\bar{x}) \supset [{}^*f(\bar{x}) \wedge \psi(\bar{x}, f(\bar{x}))]]$ is valid.

For $\psi \equiv T$, Theorem 2 implies

*Corollary* 2

(a) $f_s$ is defined w.r.t. $\phi$
if and only if
$\forall \bar{x}[f(\bar{x}) \stackrel{*}{=} \tau_s(\bar{x}, f)] \supset \forall \bar{x}[\phi(\bar{x}) \supset {}^*f(\bar{x})]$ is valid.

(b) $f_p$ is defined w.r.t. $\phi$
if and only if
$\forall \bar{x}[f(\bar{x}) \stackrel{*}{=} \tau_p(\bar{x}, f)] \supset \forall \bar{x}[\phi(\bar{x}) \supset {}^*f(\bar{x})]$ is valid.

*Example*

For illustration we shall use the recursive definition

$$f(x) \Leftarrow \textbf{if } x > 100 \textbf{ then } x - 10 \textbf{ else } f(f(x+11)),$$

where $D = \{\text{the integers}\}$. Since $x > 100$ is a total predicate, $f_s(x)$ and $f_p(x)$ are identical for this recursive definition. We denote this unique function by $f_{91}(x)$.

Let $\phi_{91}(x)$ be $T$ and $\psi_{91}(x, y)$ be defined as:

$$\textbf{if } x > 100 \textbf{ then } y = x - 10 \textbf{ else } y = 91.$$

1. Using Theorem 1 we shall prove that $f_{91}$ is partially correct w.r.t. $\phi_{91}$ and $\psi_{91}$, i.e., for every integer $\xi$: if $f_{91}(\xi)$ is defined then $\psi_{91}(\xi, f_{91}(\xi)) = T$. Thus, by Theorem 1, we have to specify a partial function $f$ (mapping integers into integers) such that the expression

$$\forall x[f(x) \stackrel{*}{=} \textbf{if } x > 100 \textbf{ then } x - 10 \textbf{ else } f(f(x+11))]$$
$$\wedge \forall x[{}^*f(x) \supset \textbf{if } x > 100 \textbf{ then } f(x) = x - 10 \textbf{ else } f(x) = 91]$$

will be true.

The reader can verify easily that the expression is true for $f(x)$ chosen as: **if** $x > 100$ **then** $x - 10$ **else** 91.

2. Using Theorem 2 we shall prove that $f_{91}$ is correct w.r.t. $\phi_{91}$ and $\psi_{91}$, i.e., for every integer $\xi$: $f_{91}(\xi)$ is defined and $\psi_{91}(\xi, f_{91}(\xi)) = T$. Thus, we have to prove that for every assignment of a partial function (mapping integers into integers) for $f$, the following formula is true:

$$\forall x[f(x) \stackrel{*}{=} \textbf{if } x > 100 \textbf{ then } x - 10 \textbf{ else } f(f(x+11))]$$
$$\supset \forall x[{}^*f(x) \wedge \textbf{if } x > 100 \textbf{ then } f(x) = x - 10 \textbf{ else } f(x) = 91].$$

Let $f(x)$ be any partial function mapping integers into integers. Assuming that the antecedent, i.e., $\forall x[f(x) \stackrel{*}{=} \textbf{if } x > 100 \textbf{ then } x - 10 \textbf{ else } f(f(x+11))]$, is true, we shall prove by (generalized) induction on $x$ that the consequent, i.e., $\forall x[{}^*f(x) \wedge \psi_{91}(x, f(x))]$, is also true. (*See* Burstall (1969). This presentation of our example was suggested by Rod Burstall.)

For $x > 100$, ${}^*f(x) \wedge \psi_{91}(x, f(x))$ is clearly true.

Now suppose that ${}^*f(x) \wedge \psi_{91}(x, f(x))$ is true for every $x$, $x > x_0$, we shall show that ${}^*f(x_0) \wedge \psi_{91}(x_0, f(x_0))$ is also true. We distinguish between two cases:

(a) $100 \geqslant x_0 > 89$

Since the antecedent implies that

$$f(x) \overset{*}{=} \textbf{if } x>100 \textbf{ then } x-10 \textbf{ else } f(f(x+11))$$
$$\overset{*}{=} \textbf{if } x>100 \textbf{ then } x-10 \textbf{ else } f(\textbf{if } x+11>100 \textbf{ then } x+11-10 \textbf{ else } f(f(x+11+11)))$$
$$\overset{*}{=} \textbf{if } x>100 \textbf{ then } x-10 \textbf{ else (if } x>89 \textbf{ then } f(x+1) \textbf{ else } f(f(f(x+22)))),$$

it follows that $f(x_0) \overset{*}{=} f(x_0+1)$. By the induction hypothesis $*f(x_0+1) \wedge \psi_{91}(x_0+1, f(x_0+1))$ is true, thus $*f(x_0) \wedge \psi_{91}(x_0, f(x_0))$ is also true.

(b) $89 \geqslant x_0$

$f(x_0) \overset{*}{=} f(f(x_0+11))$ from the antecedent
$\overset{*}{=} f(91)$ by the ind. hyp. since $100 \geqslant x_0+11 > x_0$
$\overset{*}{=} 91$ by the ind. hyp. since $91 > x_0$.

Thus $*f(x_0) \wedge \psi_{91}(x_0, f(x_0))$ is true.

## PROOFS

We proceed to prove Theorems 1(a) and 2(a) using Lemmas 1(a) and 2(a). Similarly, one can prove Theorems 1(b) and 2(b), using Lemmas 1(b) and 2(b). We shall make use of the fact that for every partial function $f'$ (mapping $D^n$ into $D$) which is an extension of $f_s$:

1. if $f'$ is partially correct w.r.t. $\phi$ and $\psi$, then $f_s$ is partially correct w.r.t. $\phi$ and $\psi$, and

2. if $f_s$ is correct w.r.t. $\phi$ and $\psi$, then $f'$ is correct w.r.t. $\phi$ and $\psi$.

### Proof of Theorem 1 (a)

*partial correctness⇒satisfiability*

Replace $f$ in the formula by $f_s$. By Lemma 1(a)

$$\forall \bar{x}[f_s(\bar{x}) \overset{*}{=} \tau_s(\bar{x}, f_s)]$$

is true. Moreover, since $f_s$ is partially correct w.r.t. $\phi$ and $\psi$, it follows that

$$\forall \bar{x}[[\phi(\bar{x}) \wedge *f_s(\bar{x})] \supset \psi(\bar{x}, f_s(\bar{x}))]$$

is also true. Thus, the formula

$$\forall \bar{x}[f(\bar{x}) \overset{*}{=} \tau_s(\bar{x}, f)] \wedge \forall \bar{x}\ [\phi(\bar{x}) \wedge *f(\bar{x})] \supset \psi(\bar{x}, f(\bar{x}))]$$

is satisfiable.

*satisfiable⇒partial correctness*

Let $f'$ be a partial function (mapping $D^n$ into $D$) for which the formula

$$\forall \bar{x}[f'(\bar{x}) \overset{*}{=} \tau_s(\bar{x}, f')] \wedge \forall \bar{x}[[\phi(\bar{x}) \wedge *f'(\bar{x})] \supset \psi(\bar{x}, f'(\bar{x}))]$$

is true. Since

$$\forall \bar{x}[f'(\bar{x}) \overset{*}{=} \tau_s(\bar{x}, f')]$$

is true, it follows by Lemma 2(a) that $f'$ is an extension of $f_s$, and since

$$\forall \bar{x}[[\phi(\bar{x}) \wedge *f'(\bar{x})] \supset \psi(\bar{x}, f'(\bar{x}))]$$

is also true, it follows that $f'$ is partially correct w.r.t. $\phi$ and $\psi$. Thus, $f_s$ is partially correct w.r.t. $\phi$ and $\psi$.

**Proof of Theorem 2(a)**

*validity⇒correctness*

Since the formula

$$\forall\bar{x}[f(\bar{x})\overset{*}{=}\tau_s(\bar{x},f)]\supset\forall\bar{x}[\phi(\bar{x})\supset[{}^*f(\bar{x})\wedge\psi(\bar{x},f(\bar{x}))]$$

is true for every partial function $f$ (mapping $D^n$ into $D$), it is true in particular for $f_s$. Therefore, since by Lemma 1(a)

$$\forall\bar{x}[f_s(\bar{x})\overset{*}{=}\tau_s(x,f_s)]$$

is true, it follows that

$$\forall\bar{x}[\phi(\bar{x})\supset[{}^*f_s(\bar{x})\wedge\psi(\bar{x},f_s(\bar{x}))]]$$

is also true. Thus, $f_s$ is correct w.r.t. $\phi$ and $\psi$.

*correctness⇒validity*

Let $f'$ be any partial function (mapping $D^n$ into $D$) for which

$$\forall\bar{x}[f'(\bar{x})\overset{*}{=}\tau_s(\bar{x},f')]$$

is true. By Lemma 2(a), $f'$ is an extension of $f_s$. Since $f_s$ is correct w.r.t. $\phi$ and $\psi$, it follows that $f'$ is also correct w.r.t. $\phi$ and $\psi$, or in other words, that

$$\forall\bar{x}[\phi(\bar{x})\supset[{}^*f'(\bar{x})\wedge\psi(\bar{x},f'(\bar{x}))]]$$

is true. Thus, the formula

$$\forall\bar{x}[f(\bar{x})\overset{*}{=}\tau_s(\bar{x},f)]\supset\forall\bar{x}[\phi(\bar{x})\supset[{}^*f(\bar{x})\wedge\psi(\bar{x},f(\bar{x}))]]$$

is valid.

## REFERENCES


Burstall, R.M. (1969) Proving Properties of Programs by Structural Induction. *Comp. J.*, **12**, 41–8.

Hayes, P.J. (1969) A Machine-Oriented Formulation of the Extended Functional Calculus. Stanford Artificial Intelligence Project, Memo 62. Also appeared as Metamathematics Unit report, University of Edinburgh, Scotland.

Kleene, S.C. (1952) *Introduction to Metamathematics*. New York: Van Nostrand.

Łukasiewicz, J. (1941) Die Logik und das Grundlagenproblem. *Les Entretiens de Zurich*, pp. 82–108.

Manna, Z. & Pnueli, A. (1968) The Validity Problem of the 91-function. Stanford Artificial Intelligence Project, Memo 68.

Manna, Z. & Pnueli, A. (1969) Formalization of Properties of Recursively Defined Functions. *Proceedings of the ACM Symposium on Theory of Computing*, Marina del Rey.

McCarthy, J. (1962) Towards a Mathematical Science of Computation. *Proc. IFIP Congress*, 62. Amsterdam: North Holland.

McCarthy, J. (1963a) Predicate Calculus with 'Undefined' as a Truth-Value. Stanford Artificial Intelligence Project, Memo 1.

McCarthy, J. (1963b) A Basis for a Mathematical Theory of Computation. *Computer Programming and Formal Systems* (eds. Braffort, P. & Hirschberg, D.). Amsterdam: North Holland.

Wang, H. (1961) The Calculus of Partial Predicates and its Extension to Set Theory. *Zeitschift für Math Logik und Grundlagen der Math.*, **7**, 283–8.

# 2

# Program Schemes and Recursive Function Theory

R. Milner
Department of Computer Science
University College of Swansea

## 1. INTRODUCTION

A program scheme (defined formally in section 2) may be regarded as a program with the interpretation (i.e., the meaning of the basic functions and tests) left unspecified. Two schemes are called equivalent if they do the same thing under all interpretations, and thus it is a sufficient condition for the equivalence of two programs that they represent equivalent schemes when their normal interpretation is removed.

Others (Luckham, Park and Paterson 1967, Paterson 1967) have examined the decision problem for program scheme equivalence under all total interpretations (i.e., when the functions and predicates of the interpretation are everywhere defined). In general, there is not even a partial decision procedure for equivalence, and it follows that there can be no general effective procedure for simplifying a program scheme to some canonical form under equivalence-preserving transformations. However, such a procedure may exist for non-trivial subclasses of program schemes; if such a subclass were large enough, this could be of real practical significance. The only such subclasses at present known – *see* Paterson (1967) – are rather restricted.

We define here – *see also* Milner (1969) – a stronger equivalence relation on program schemes; namely, equivalence under *all* interpretations (allowing partial functions and predicates). This is a natural relation, at least in the sense that it turns out to have some simple properties – and it also reflects the real situation where a program primitive is sometimes in fact a subroutine which is not everywhere defined – but equivalence is still not partially decidable. However, it has the compactness property that equivalence under all finite partial interpretations (those in which the function and predicate domains are finite) implies equivalence under all interpretations.

The main purpose of this paper is to attempt to characterize program schemes in terms of recursively enumerable (r.e.) sets (of integers), in such

a way that equivalence of schemes maps on to equality of sets. We attempt this for both the equivalence relations mentioned above. The attempt is successful in that we can define a mapping from schemes to sets with the required property, but only partly successful in obtaining an *independent* description of the class of sets so obtained. Some recursion-theoretic and lattice-theoretic properties of the class are found, but until the class is completely described one cannot expect to prove results about program schemes – such as discovering large subclasses where equivalence is partially decidable – by proving results about sets. It is argued that recursion theory will not yield a complete description by itself.

There is another motive for getting such an alternative characterization of program schemes. In the course of sections 3 and 4 we show that they may be thought of as a subclass of recursive functionals or operators (defined below), closely associated with the class of r.e. sets. The same holds for recursion equation schemes, which bear the same relation to recursion equations – *see*, for example, McCarthy (1963) – as program schemes to programs. Regarded as recursive functionals whose arguments (or oracles) are the functions and predicates of the interpretation, both types of scheme have the special property that *all* their computing and decision-making is done by reference to the oracles; the only thing invariant under change of interpretation (oracles) is the structure of the scheme, i.e., the way it organizes the computation on the basis of these references. (Contrast this with a Turing machine, register machine program, or set of recursion equations with added oracles; here certain base functions and predicates also remain invariant under change of interpretation [oracles].) Thus the class of recursive functionals represented by program schemes (respectively, recursion equation schemes) will characterize the computing structure on which they are based – i.e., the (programless) register machine (respectively, the evaluation mechanism for recursion equations). As a preliminary result in this direction, Paterson (private communication) has shown that recursion equation schemes are strictly more powerful – in our terminology, they represent a larger class of functionals – than program schemes. (Again, contrast the known equivalence of register machines and recursion equations, with or without oracles, *given* certain base functions.) Thus the complete characterization which we aim at will elucidate the difference in power between various computing structures, or (which is the same thing) evaluation mechanisms. In particular, it will capture the limitation due to having finitely many registers in a program scheme.

## 2. DEFINITIONS AND PRELIMINARIES

We use *register letters* $L_1, L_2, \ldots$, a single monadic *function letter* $F$ and a single monadic *test letter* $T$.

An *assignment* is $L_i := F(L_j)$ or $L_i := L_j (i, j \geqq 1)$.

A *test* is $T(L_i)$ $(i \geqq 1)$.

A *program scheme* is a finite directed graph each node of which is labelled by a test and has 2 successor nodes, or by an assignment and has one successor node, or by STOP and has no successor nodes. In case of a test node, the outgoing arcs are labelled 0, 1 (for *false*, *true*).

A (*partial*) *interpretation* $I$ is a quadruple $(D_I, L_I, F_I, T_I)$ where $D_I$ is a universe of individuals, $L_I$ (the input) $\in D_I$, and $F_I: D_I \to D_I$ and $T_I: D_I \to \{0, 1\}$ are partial functions.

A (*partial*) *semi-interpretation* $I$ is a triple $(D_I, F_I, T_I)$ – simply an interpretation but without specifying input. We use $I$ to mean sometimes an interpretation, sometimes a semi-interpretation (the context makes clear which).

$I$ is *total* if $F_I$ and $T_I$ are total.

$I$ is *finite* if $F_I$, $T_I$ have finite domains (thus, a total finite interpretation has a finite universe).

A *free interpretation* $I^f$ is total and has $D_{If}$ = the set of words $\{L, FL, FFL, \ldots\} = \{F^nL \mid n \geqq 0\}$, $L_{If} = L$, and $F_{If}(F^nL) = F^{n+1}L$. Thus a free interpretation is determined completely by $T_{If}$. The *string* of $I^f$, denoted $s(I^f)$, is that infinite binary sequence in which the $n$th member ($n \geqq 0$) is the value of $T_{If}(F^nL)$.

Assume an effective numbering of all program schemes $P_0, P_1, P_2 \ldots$.

Each $I$ defines a *computation* for $P_i$ in an obvious way (which we do not bother to formalize). We assume that the input $L_I$ is present on entry in *all* registers of $P_i$ (each $P_i$ can use only a finite number of registers, being a finite graph). In its computation under $I$, $P_i$ performs a sequence of evaluations – i.e., applications of $F_I$ or $T_I$ to an argument. It may (a) reach a STOP node, in which case it *converges to* $x$ (where $x$ is the final contents of $L_1$), or (b) at some point attempt an undefined evaluation, in which case it *sticks* on this evaluation, or (c) do neither, in which case it *strongly diverges*. We use *diverges* to mean 'strongly diverges or sticks'.

Note that a computation has only one input and one output. This simplifies the presentation, but all of what follows can be generalized to a set of inputs and a set of outputs. We can also generalize to more than one function and test letter – not necessarily monadic. We have allowed copying instructions ($L_i := L_j$) though these are excluded in Luckham, Park and Paterson (1967). However, this does not affect any results quoted.

We now define some partial orderings on program schemes.

$P_i \leqq^p P_j$ means $\forall I \forall x [P_i$ converges to $x \Rightarrow P_j$ converges to $x]$

$P_i \leqq^{p'} P_j$ means $\forall I [P_i$ converges $\Rightarrow P_j$ converges$]$

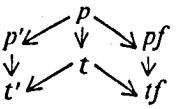
$P_i \leqq^{pf} P_j$ means $\forall I [I$ finite $\Rightarrow \forall x [P_j$ converges to $x \Rightarrow P_j$ converges to $x]]$

$P_i \leqq^t P_j$
$P_i \leqq^{t'} P_j$
$P_i \leqq^{tf} P_j$
} like $\leqq^p$, $\leqq^{p'}$, $\leqq^{pf}$ but restricting $I$ to total interpretations.

Each relation is clearly transitive and reflexive; we therefore have a corresponding set of equivalence relations – for example

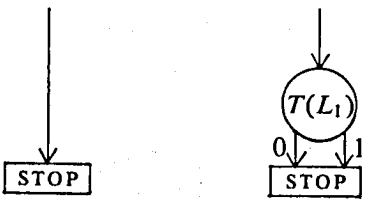
$$P_i \equiv {}^tP_j \Leftrightarrow P_i \leqq {}^tP_j \quad \text{and} \quad P_j \leqq {}^tP_i.$$

The following diagram of implications is easily verified (for $p \to t$ read $\leqq^p \Rightarrow \leqq^t$, etc.).




Luckham, Park and Paterson (1967) have shown that $P_i \leqq {}^{tf}P_j \not\Rightarrow P_i \leqq {}^tP_j$; on the other hand, in section 3 below, we prove that $\leqq^{pf} \Leftrightarrow \leqq^p$ – in fact the behaviour of a scheme under all partial finite interpretations determines its behaviour under all interpretations. Equivalence relations on program schemes under partial interpretations are further studied in Milner (1969), but the approach here is rather different from that paper.

It is easy to show that apart from $\leqq^{pf} \Rightarrow \leqq^p$, no implications hold in the above diagram other than those shown. One does not have to look far for counter-examples: the two schemes




are enough to show $\leqq^{t'} \not\Rightarrow \leqq^{p'}$, $\leqq^t \not\Rightarrow \leqq^p$, $\leqq^{tf} \not\Rightarrow \leqq^{pf}$.

We introduce some further notation. Assume an effective numbering of the finite sets of integers, $D_0, D_1, D_2, \ldots$, and of the partial recursive functions of one variable, $\phi_0, \phi_1, \phi_2, \ldots$, and denote the domain of $\phi_i$ by $W_i$, and the Turing machine which computes $\phi_i$ by $TM_i$. $N$ is the set of non-negative integers.

$\mathscr{P}$ is the class of partial functions $\phi : N \to N$.

$\mathscr{P}^{01}$ is the class of partial functions $\theta : N \to \{0, 1\}$.

$\mathscr{F}$, $\mathscr{F}^{01}$ are $\mathscr{P}$, $\mathscr{P}^{01}$ respectively restricted to total functions.

$\langle x, y \rangle$ is an effective onto 1–1 coding $N \times N \to N$. We consider a partial function $\phi$ to be $\{\langle x, y \rangle \mid \phi(x) = y\}$.

$\langle x_1, x_2, x_3 \ldots \rangle$ is short for $\langle \ldots \langle \langle x_1, x_2 \rangle, x_3 \rangle, \ldots \rangle$.

## 3. THE RELATIONS $\leqq^p$, $\leqq^{p'}$ AND RECURSIVE OPERATORS

We show that program schemes under partial interpretations may be regarded as a subclass of recursive operators (defined below). This also enables us to associate with each $P_i$ a recursive set, and the structure of these sets under inclusion is isomorphic to that of the $P_i$ under $\leqq^p$.

First, we can clearly restrict the universe of interpretations to (subsets of) $N$ – in other words the relation $\leqq^p$ is unchanged by this restriction (under

any $I$, a scheme can only 'look at' denumerably many individuals). We assume the restriction throughout this section.

*Definition.* The *partial recursive operator* associated with $W_i$ is a partial mapping $\Phi : \mathscr{P} \times \mathscr{P}^{01} \to \mathscr{P}$ defined by

$$\Phi(\psi, \theta) = \{\langle x, y\rangle \mid \exists u \exists v[\langle x, y, u, v\rangle \in W_i \text{ and } D_u \subset \psi,\ D_v \subset \theta]\}.$$

*See* Rogers (1967, para 9.8). His $\Phi$s have one argument, ours have two: we can clearly extend the definition to any number of arguments, or to non-monadic arguments. Intuitively, given any finite subsets of $\psi$ and $\theta$, $\Phi$ enumerates a subset (not necessarily finite) of the output function. Thus $\Phi$ is an extensional mapping – it deals with the actual argument-value pairs of functions, not with indices (in some numbering) of functions.

However, the output of $\Phi$ is in general a binary relation, which may not be single valued. The *domain* of $\Phi$ is that subset of $\mathscr{P} \times \mathscr{P}^{01}$ for which

$$\forall x \forall y_1 \forall y_2[\langle x, y_1\rangle \text{ and } \langle x, y_2\rangle \in \Phi(\psi, \theta) \Rightarrow y_1 = y_2].$$

$\Phi$ is a *recursive operator* if its domain is $\mathscr{P} \times \mathscr{P}^{01}$.

Now associate with each $P_i$ the r.e. set

$W_{p(i)} = \{\langle x, y, u, v\rangle \mid P_i \text{ converges to } y \text{ under the interpretation } (N, x, D_u, D_v)\}$.

Given $P_i$, we can effectively enumerate $W_{p(i)}$ (try every $\langle x, y, u, v\rangle$: accept it if $P_i$ converges to $y$!), and this ensures that $p$ is recursive. We call $W_{p(i)}$ the *p-set* of $P_i$.

*Theorem* 3.1

$W_{p(i)}$ is recursive, and there is a fixed recursive function $c$ such that $\forall i[\phi_{c(i)}$ *is the characteristic function of* $W_{p(i)}]$.

*Proof.* For any $z = \langle x, y\ u, v\rangle$, decide whether $z \in W_{p(i)}$ as follows: Compute $P_i$ under $I = (N, x, D_u, D_v)$. One of the following must occur

(a) $P_i$ converges to some $y'$. Then $y' = y \Leftrightarrow z \in W_{p(i)}$.

(b) $P_i$ sticks on some evaluation. Then $z \notin W_p(i)$.

(c) $P_i$ strongly diverges. Then $z \notin W_{p(i)}$.

Moreover, we can decide which after a bounded number of steps. For the computation can only adopt one of a finite number $k$ of 'states', where

$$k = \text{number of registers} \times \text{number of nodes} \times \text{size of } D_u,$$

so that if the computation lasts $k+1$ steps some state must have been repeated and we know that $P_i$ must strongly diverge cyclically. Thus we have a characteristic function $\phi_{c(i)}$ for $W_{p(i)}$, and since the function is obtained effectively from $P_i$, $c$ must be recursive.

Now under any semi-interpretation $I = (N, F_I, T_I)$ it is clear that we obtain from $P_i$ a partial function which contains $\langle x, y\rangle$ iff for some $D_u \subset F_I$ and $D_v \subset T_I$, we have $\langle x, y, u, v\rangle \in W_{p(i)}$. Thus $P_i$ is a recursive operator defined by

$$P_i(F_I, T_I) = \{\langle x, y\rangle \mid \exists u \exists v[\langle x, y, u, v\rangle \in W_{p(i)} \text{ and } D_u \subset F_I,\ D_v \subset T_I]\}.$$

$P_i$ is recursive since $y$, if it exists, is uniquely determined by $x, F_I, T_I$.

The question immediately arises: can we characterize the subclass $\{P_i\}$ of all recursive operators $\Phi: \mathscr{P} \times \mathscr{P}^{01} \to \mathscr{P}$ by properties of the class of recursive sets $\{W_{p(i)}\}$? Not every recursive set is in the class: the $W_{p(i)}$ have special properties. But it is unlikely that recursive function theory will provide a full description of the class, any more than it does for, say, the context-free languages – another class of recursive sets – since all recursive sets (except those which are finite or have finite complement) are recursively isomorphic, and recursive function theory is mainly concerned with properties of sets which are invariant under recursive isomorphism. However, we mention some simple algebraic properties of the class in section 5.

*Theorem* 3.2

$$P_i \leqq^p P_j \Leftrightarrow P_i \leqq^{pf} P_j \Leftrightarrow W_{p(i)} \subset W_{p(j)}.$$

*Proof.*

(1) $P_i \leqq^p P_j \Rightarrow P_i \leqq^{pf} P_j$: immediate.

(2) $P_i \leqq^{pf} P_j \Rightarrow W_{p(i)} \subset W_{p(j)}$: immediate from definition of $W_{p(i)}$.

(3) $W_{p(i)} \subset W_{p(j)} \Rightarrow P_i \leqq^p P_j$:

Assume $W_{p(i)} \subset W_{p(j)}$. Suppose $\langle x, y \rangle \in P_i(F_I, T_I)$ – i.e., $P_i$ converges to $y$ under $I = (N, x, F_I, T_I)$. Then from definition of $P_i$ as a recursive operator we have for some $u$, $v$

$$\langle x, y, u, v \rangle \in W_{p(i)} \text{ and } D_u \subset F_I,\ D_v \subset T_I.$$

Hence by hypothesis $\langle x, y, u, v \rangle \in W_{p(j)}$, so $\langle x, y \rangle \in P_j(F_I, T_I)$, and $P_i \leqq^p P_j$ follows.

Thus the structure of the $P_i$ under $\leqq^p$ or $\leqq^{pf}$ is isomorphic to that of the recursive sets $W_{p(i)}$ under inclusion. The equivalence of $\leqq^p$, $\leqq^{pf}$ (in contrast to $\leqq^t$, $\leqq^{tf}$) indicates the greater simplicity of properties of schemes when partial interpretations are allowed. Another indication is in the first part of the following theorem.

*Theorem* 3.3

(1) $P_i \not\leqq^p P_j$ is partially decidable.

(2) $P_i \leqq^p P_j$ is not partially decidable.

*Proof.*

(1) We have $P_i \not\leqq^p P_j \Leftrightarrow W_{p(i)} \not\subset W_{p(j)} \Leftrightarrow \exists x[\phi_{c(i)}(x) = 1 \text{ and } \phi_{c(j)}(x) = 0]$.
But the latter is partially decidable, since $\phi_{c(i)}$, $\phi_{c(j)}$ are recursive.

(2) Let $D$ be a fixed program scheme that diverges under all interpretations. Then $P_i \leqq^p D \Leftrightarrow P_i \leqq^t D$ ($\Rightarrow$ is immediate: $\Leftarrow$ holds since, if $P_i \not\leqq^p D$ then $P_i$ must converge under some interpretation, and hence also under some total interpretation). We may then use a result of Luckham, Park and Paterson (1967) that the latter is partially undecidable. (The proof is a simple application of the Turing machine simulator: see the Appendix.)

All the theorems of this section can be restated for $\leqq^{p'}$ instead of $\leqq^p$. Corresponding to $W_{p(i)}$ we define the *p'-set* of $P_i$:

$$W_{p'(i)} = \{\langle x,u,v \rangle | P_i \textit{ converges under } I = (N,x,D_u,D_v)\}.$$

**Discussion**

The relation $P_i \equiv^p P_j$ is stronger than $P_i \equiv^t P_j$; what does it require of $P_i$, $P_j$? It is easy to see that it requires that under any $I$, if $P_i$ and $P_j$ converge they perform precisely the same set of evaluations (though not necessarily in the same sequence). For if not, we can restrict $I$ in such a way that one sticks while the other still converges. In fact, if the above condition holds and $P_i \equiv^t P_j$ then $P_i \equiv^p P_j$.

Similarly, if $P_i \leqq^p P_j$ then under any $I$ if $P_i$ converges $P_j$ must perform no evaluations not performed by $P_i$; in a sense $P_j$ is more efficient than $P_i$. However, there is no restriction on the number of times each evaluation is performed. One may try to relate the above ideas to Paterson's (1967) *liberal* schemes (a scheme is liberal if it never performs the same evaluation more than once under any free interpretation); certainly given a fixed scheme $B$, the most efficient $A \equiv^t B$ (if such an $A$ exists) is one which is liberal, and also such that

$$\forall C[C \equiv^t B \Rightarrow C \leqq^p A]. \qquad (*)$$

This suggests the question: is there for each $B$ an $A$ satisfying (*)? We do not pursue this.

Just as recursion equations are an alternative to programs in defining functions, so recursion equation schemes (as, for example, defined in McCarthy 1963) are an alternative to program schemes in defining functional operators. All the results of this section can be extended without change to recursion equation schemes, though the proofs differ in some details.

## 4. THE RELATIONS $\leqq^t$, $\leqq^{t'}$ AND RECURSIVE FUNCTIONALS

We now attempt a similar characterization in terms of sets of the behaviour of program schemes under *total* interpretations. Since $\leqq^p \Rightarrow \leqq^t$, $W_{p(i)} \subset W_{p(j)}$ is a sufficient (but not necessary) condition for $P_i \leqq^t P_j$. It is possible, but not very helpful, to express a necessary and sufficient condition for $P_i \leqq^t P_j$ in terms of the $p$-sets. But we now show how to associate with each $P_i$ an r.e. set $W_{t(i)}$ such that

$$P_i \leqq^t P_j \Leftrightarrow W_{t(i)} \subset W_{t(j)}.$$

We consider only free interpretations in this section, using a result of Luckham, Park and Paterson (1967) that the relation $\leqq^t$ is unchanged if we restrict to free interpretations.

We are concerned with finite binary sequences, and we assume an effective numbering of these: $S_0, S_1, S_2, \ldots$

*Definition.* $S_k < S_j$, or $S_j > S_k$ means that $S_k$ is an initial segment of $S_j$ (this includes the case $S_k = S_j$). We also write $S_k < I^f$, or $I^f > S_k$ to mean that $S_k$ is an initial segment of $S(I^f)$, and then we say that $I^f$ is an *extension* of $S_k$.

We now define $W_{t(i)}$, the *t-set* of $P_i$, as follows:

$$W_{t(i)} = \{\langle k, n\rangle \mid P_i \textit{ converges to } F^n L \textit{ on every extension of } S_k\}.$$

To justify this definition (and show that $t$ is a recursive function), we only

need to apply König's infinity lemma to the finitely branching tree of execution sequences of $P_i$, pruned of all branches inconsistent with $S_k$. This ensures that if $P_i$ converges on every $I^f > S_k$, then we will discover this fact after a finite number of steps in enumerating the tree (i.e., the tree is finite). Paterson (1967) uses the same argument but without the constraint of $S_k$ – to show that 'always converging' is a partially decidable property of program schemes.

Now regarding $I^f$ (specified by $s(I^f)$) as a 0, 1 – valued function on the domain of free interpretations coded into the integers, we can regard $W_{t(i)}$ as defining a *recursive functional* $\Phi : \mathscr{F}^{01} \to N$ as follows:

$$\Phi(I^f) = \{n \mid \exists k[\langle k, n\rangle \in W_{t(i)} \text{ and } S_k < I^f]\}.$$

Rogers (1967, para. 15.3) provides a formal definition of partial recursive and recursive functionals; our use of $<$ and initial segments is an inessential difference from his definition. Our $\Phi$ is recursive, not just partial recursive, since $\Phi(I^f)$ is at most a singleton.

The fact that programs schemes under free interpretations are a subclass of the recursive functionals gives the motivation for characterizing this subclass by determining properties of the $t$-sets.

*Theorem* 4.1

$$P_i \leqq {}^tP_j \Leftrightarrow W_{t(i)} \subset W_{t(j)}.$$

*Proof.* ($\Rightarrow$) Assume $P_i \leqq {}^tP_j$.

Then $\langle k, n\rangle \in W_{t(j)} \Rightarrow P_i$ converges to $F^nL$ on each $I^f > S_k$
$\Rightarrow P_j$ converges to $F^nL$ on each $I^f > S_k$
$\Rightarrow \langle k, n\rangle \in W_{t(j)}$.

Hence $W_{t(i)} \subset W_{t(j)}$ since $k, n$ were arbitrary.

($\Leftarrow$) Assume $W_{t(i)} \subset W_{t(j)}$.

Suppose under $I^f$ $P_i$ converges to $F^mL$. Then some $S_k < I^f$ defines $P_i$'s execution sequence uniquely, so $\langle k, m\rangle \in W_{t(i)}$, so by assumption $\langle k, m\rangle \in W_{t(j)}$. Hence $P_j$ converges to $F^mL$ under every extension of $S_k$, and in particular under $I^f$.

Hence $P_i \leqq {}^tP_j$, since $I^f$ was arbitrary.

We now look for some properties of the $t$-sets. First, some standard definitions and results from recursive function theory: *see* Rogers (1967).

$W_j \leqq_T W_j$ (for $\leqq_T$ read '*is Turing reducible to*') iff

$\forall z$ [*The question* '$z \in W_i$?' *can be settled by asking some finite number of questions of the form* '$y \in W_j$?']

$W_i \leqq_m W_j$ (for $\leqq_m$ read '*is many-one reducible to*') via the recursive function $f$
iff $\forall z[z \in W_i \Leftrightarrow f(z) \in W_j]$

$W_i \leqq_1 W_j$ (for $\leqq_1$ read '*is one-one reducible to*') via $f$
iff $W_i \leqq_m W_j$ via $f$ and $f$ is 1-1 recursive.

The three relations are transitive and reflexive; the equivalence classes induced are known as *T-degrees, m-degrees* and *1-degrees*. It is well known

that $\leqq_1 \Rightarrow \leqq_m \Rightarrow \leqq_T$ but the reverse implications do not hold. There is a maximum 1-degree in each $m$-degree. Moreover, $W_i \equiv_1 W_j$ iff there is a recursive permutation $f$(i.e., $f: N \to N$ is recursive, $1-1$ and onto) such that $z \in W_i \Leftrightarrow f(z) \in W_j$. In this case $W_i$, $W_j$ are *recursively isomorphic.* Recursive function theory is concerned mainly with properties which are invariant under recursive isomorphism.

*Theorem* 4.2

There is a $t$-set in every $m$-degree, except that of $N$. In fact if $W_i \neq N$ then there is a $t$-set $W_j$ such that

$$W_i \leqq_1 W_j,\ W_j \leqq_m W_i.$$

*Proof.* See the Appendix.

*Corollary*

There is a $t$-set in the maximum 1-degree in every $m$-degree, except that of $N$.

*Proof.* Choose $A$ in such a maximum 1-degree. Then, by the theorem, there is a $t$-set $B$ such that $A \leqq_1 B$, $B \leqq_m A$, so also $B \equiv_m A$. But since the 1-degree of $A$ is maximum, $\forall C[C \equiv_m A \ \& \ A \leqq_1 C \Rightarrow C \equiv_1 A]$. Hence $B \equiv_1 A$.

The $t$-sets under inclusion capture the structure of the $P_i$ under $\leqq^t$. Equally, the structure of the $P_i$ under $\leqq^{t'}$ is captured by the $t'$-sets under inclusion, where

$$W_{t'(i)} = \{k \mid P_i \textit{ converges on all } I^f > S_k\}.$$

We can easily extend Theorem 4.2 also, to show that there is a $t'$-set in every $m$-degree (even that of $N$!).

Is there a $t'$-set (or $t$-set) in every recursive isomorphism class (that is, in every 1-degree)? No; it is easy to show that no $t'$-set (or $t$-set) is finite and non-empty. The following theorem is more interesting (and is true also for $t$-sets).

*Definition.* A r.e. set $A$ is *hypersimple* iff $\bar{A}$ is infinite but there is no infinite r.e. set of disjoint finite sets each intersecting $\bar{A}$.

Post proved the existence of such sets. Dekker (1954) showed the existence of one in every non-recursive $T$-degree. The property of hypersimplicity is clearly invariant under recursive isomorphism.

*Theorem* 4.3

No $t'$-set is hypersimple.

*Proof.* Suppose $W_{t'(i)} \neq N$($N$ is trivially not hypersimple). Then define the following sequence $E_j$ of finite sets of initial segments.

$$E_1 = \{0, 1\}$$
$$E_2 = \{00, 01, 10, 11\}$$
$$E_3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$$
$$-\ -\ -\ -\ -$$
$$-\ -\ -\ -\ -$$

Let $E_j^*$ be the correspondence sequence, each segment $S_k$ replaced by its index. Then clearly each $E_j^*$ contains a $k \notin W_{t'(i)}$, otherwise $P_i$ would converge

on *all* extensions of *all* members of $E_j^*$ – i.e., would always converge. But then $W_{t'(i)}=N$, contrary to hypothesis. But the $E_j^*$ are disjoint, and the sequence is clearly r.e. Hence $W_{t'(i)}$ is not hypersimple.

*Corollary*

There are non-recursive 1-degrees containing no $t'$-set (by the invariance of hypersimplicity under recursive isomorphism).

Now on the evidence of Theorems 4.2 and 4.3 a $t'$-set falls in a 1-degree which tends to be high (w.r.t. $\leqq_1$) in its $m$-degree (for example, hypersimplicity is hereditary downwards under $\leqq_1$). An attractive hypothesis is that the 1-degree of every $t'$-set is maximum in its $m$-degree: we would then have characterized the $t'$-sets completely up to recursive isomorphism. But this hypothesis is false. For it is known that the 1-degree of a *simple* set (that is, a r.e. set whose complement is infinite but contains no infinite r.e. set) is not maximum in its $m$-degree; Paterson (1967) has constructed a program scheme which diverges under some $I$ but under no recursive $I$ ($I$ is recursive iff $F_I$, $T_I$ are recursive), and the $t'$-set of this scheme may be shown to be simple. We do not give a proof, but anyone who inspects Paterson's elegant construction in detail should be able to supply it. It is also true that $W_{t'(i)}$ simple $\Rightarrow P_i$ converges under some $I$ but under no recursive $I$, though the converse is not clear.

Thus a characterization of the $t'$-sets up to recursive isomorphism may be complex. The results in this section hold unchanged for recursive equation schemata, and one may speculate whether such a characterization will distinguish between the $t'$-sets for program schemes and those for recursive equation schemes. But in any case, as remarked in section 3 about the $p, p'$-sets, a *full* characterization of the $t, t'$-sets will probably not be provided by recursive function theory alone.

We conclude this section with a proof that the equivalence problem for program schemes under total interpretations (or more generally the problem '$P_i \leqq^t P_j$?') is of the same degree (in fact the same $m$-degree) of unsolvability as the equivalence problem for all r.e. sets (or more generally, the problem '$W_i \subset W_j$?'). This means that the $T$-degree is $0''$, the second jump degree [*see* Rogers (1967, para. 13.1)]. The result is due to Martin Weiner (1969), though the proof given here (in the Appendix) uses rather different machinery.

*Theorem* 4.4

If $\Pi=\{\langle i,j\rangle \mid P_i \leqq^t P_j\}$ and $\Omega=\{\langle x,y\rangle \mid W_x \subset W_y\}$ then $\Pi \leqq_m \Omega$ and $\Omega \leqq_1 \Pi$.

*Proof.* See the Appendix. The first result is immediate by $t$-sets. The second requires use of the Turing machine simulator.

## 5. SIMPLE LATTICE-THEORETIC PROPERTIES OF $\leqq^{t'}$, $\leqq^{p'}$

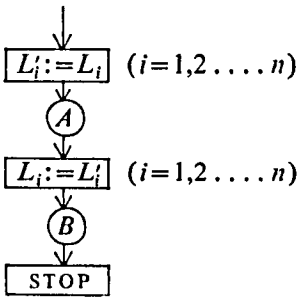
We now consider the structure of the $\equiv^{t'}$ ($\equiv^{p'}$) equivalence classes of program schemes under $\leqq^{t'}$ ($\leqq^{p'}$). It turns out that we have a distributive lattice for $\leqq^{t'}$, but only a lower semi-lattice for $\leqq^{p'}$. The same results can easily be shown for $\leqq^t$ ($\leqq^p$) if we restrict to all schemes $\leqq^t$ ($\leqq^p$) some

arbitrary fixed scheme. We do not assert that the same results hold for recursion equation schemes, though they probably do.

We need only the following concepts of lattice theory [*see*, for example, MacLane and Birkhoff (1967)]:

*Definition.* A partial ordering $\leqq$ on a set $S$ is a *lattice* if for all $A, B \in S$ there is an unique least upper bound (lub) $A \vee B$ and unique greater lower bound (glb) $A \wedge B$ in $S$; the lattice is distributive if always $(A \vee B) \wedge (A \vee C) = A \vee (B \wedge C)$ and $(A \wedge B) \vee (A \wedge C) = A \wedge (B \vee C)$; the partial ordering is a *lower* (*upper*) *semi-lattice* if there is at least a glb (lub).

We first define two simple binary operators *with, then* on program schemes. Roughly, *A with B* does *A* and *B* in parallel, and converges when *either* converges; *A then B* does *A* first, then *B*, so only converges when *both* converge. *A then B* is simply

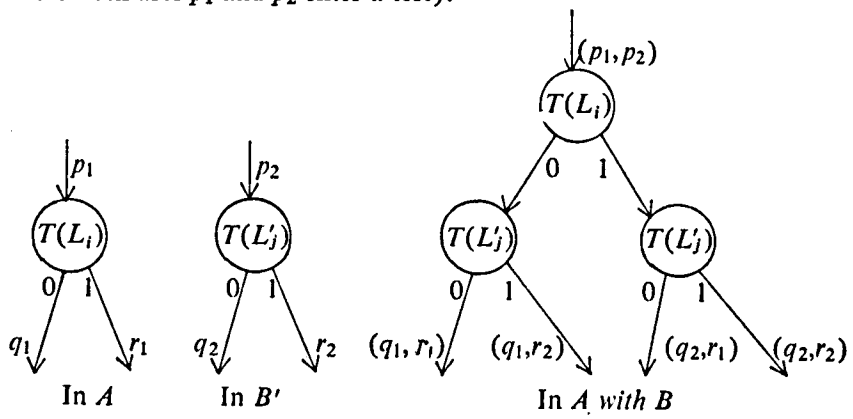



where *A* uses *n* registers. *A with B* is a little more complex, but intuitively straightforward. We form it like the Cartesian product of two finite automata thus:

1. Form $B'$ by priming all the $L_i$ in *B*.
2. Label the arcs of *A* 1, 2, ... (with 1 on the entry arc) and $B'$ similarly.
3. *A with B* has among its arcs an arc labelled $(p_1, p_2)$ for each arc labelled $p_1$ in *A*, $p_2$ in *B*, and these arcs are connected as follows:

(a) If arc $p_1$ or arc $p_2$ enters STOP, so does arc $(p_1, p_2)$.

(b) Otherwise, we have the following in *A with B* (we illustrate only the case where both arcs $p_1$ and $p_2$ enter a test):



4. The entry of *A with B* leads through a sequence of assignments $Li := L_i$ (for $i=1, 2, \ldots, n$ where $n$ is the larger of the numbers of registers used in $A$, $B$) and then to arc (1, 1).

Thus, *A with B* will always perform evaluations of $A$, $B$ alternately on separate register sets, and will converge when either converges.

The following properties are simply proved

1. *A with* $B \equiv^{t'} B$ *with* $A$, *A then* $B \equiv^{p'}$ (hence also $\equiv^{t'}$) *B then A.*
2. *with, then* are well defined on $\equiv^{t'}$ equivalence classes, and *then* is well defined on $\equiv^{p'}$ equivalence classes.

*Definition.* By $t'(A)$ ($p'(A)$) we mean the $t'$-set ($p'$-set) of the scheme $A$.

*Notation.* Conv $(A, I)$ means '$A$ converges under interpretation $I$'.

*Theorem* 5.1.

(1) The $\equiv^{t'}$ classes form a distributive lattice under $\leqq^{t'}$ with *then*, *with* as $\wedge$, $\vee$.

(2) $t'(A \textit{ then } B) = t'(A) \cap t'(B)$.

(3) $t'(A) \cup t'(B) \subset t'(A \textit{ with } B)$, but equality does not always hold.

*Proof.* (1) It is clear that

$$\text{Conv}(A \textit{ with } B, I^f) \Leftrightarrow \text{Conv}(A, I^f) \textit{ or } \text{Conv}(B, I^f)$$
$$\text{Conv}(A \textit{ then } B, I^f) \Leftrightarrow \text{Conv}(A, I^f) \textit{ and } \text{Conv}(B, I^f).$$

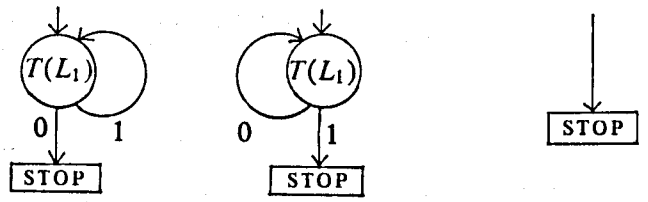
Hence for any $C$, if $A \leqq^{t'} C$ and $B \leqq^{t'} C$ we have for all $I^f$:

$$\text{Conv}(A \textit{ with } B, I^f) \Rightarrow \text{Conv}(C, I^f)$$

and so *A with* $B \leqq^{t'} C$. But this ensures that *A with B* is the lub of $A$, $B$. A similar argument shows that *A then B* is the glb, and the distributivity of the lattice follows from the distributivity of *and*, *or*.

(2) $k \in t'(A) \cap t'(B) \Leftrightarrow \text{Conv}(A, I^f)$ and $\text{Conv}(B, I^f)$ for all $I^f > S_k$
$\Leftrightarrow \text{Conv}(A \textit{ then } B, I^f)$ for all $I^f < S_k$
$\Leftrightarrow k \in t'(A \textit{ then } B)$.

(3) A similar argument to (2) holds for forward implication only, giving $t'(A) \cup t'(B) \subset t'(A \textit{ with } B)$. That equality fails is shown by the following simple example (using familiar notation of regular sets to describe a $t'$-set – or rather the set of binary strings indexed by a $t'$-set):




$A$ $B$ $C[\equiv^{t'} A \textit{ with } B]$

For we have that

$$t'(A) \cup t'(B) = 0(0+1)^* + 1(0+1)^* = (0+1)^* - \lambda = t'(C) - \lambda$$

(where $\lambda$ is the empty string).

It follows from Theorem 5.1 that the $t'$-sets form a distributive lattice with $\cap$ as $\wedge$, but the $\vee$ operation is not union. In fact the $t'$-sets are closed under $\cap$ but not under $\cup$.

Finally, we turn to the $\equiv^{p'}$ classes under $\leqq^{p'}$. An argument like Theorem 5.1, but considering all partial interpretations, easily gives Theorem 5.2(1) and (2), and we do not give details. But *A with B* will not do as a lub: in fact it is not an upper bound at all, since for example *A with B* may fail to converge when *A* converges if it meets a *B*-evaluation which is undefined. An attempt to modify the construction for *with* must fail, by Theorem 5.2 (3).

*Theorem* 5.2

(1) The $\equiv^{p'}$ classes form a lower semi-lattice under $\leqq^{p'}$ with *then* as $\wedge$.

(2) $p'(A \textit{ then } B) = p'(A) \cap p'(B)$.

(3) The $\equiv^{p'}$ classes do not form a lattice under $\leqq^{p'}$.

*Proof.* (1), (2) see above.

(3) See the Appendix. It is shown that the existence of a lub gives a solution to the Turing machine halting problem.

It follows that the $p'$-sets also form a lower semi-lattice under $\cap$, but are not closed under $\cup$.

## 6. CONCLUSION

We have tried to characterize the behaviour of program schemes in terms of r.e. sets, and have only been partly successful. It remains both to describe these sets fully up to recursive isomorphism in the case of the $t$-sets, and also to find some complete description of both $p$-sets and $t$-sets which will give an independent characterization of the $\equiv^{p}$ classes and $\equiv^{t}$ classes of program schemes. If this is done we should be able also to characterize interesting subclasses of program schemes, and perhaps find a large subclass in which equivalence is (at least partially) decidable.

As previously remarked, the method is applicable to other evaluation mechanisms – e.g., to recursion equation schemes under some particular evaluation procedure. Each evaluation mechanism will be characterized by its corresponding class of $p$-sets, or $t$-sets. We may consider also non-deterministic evaluation mechanisms where (roughly) a scheme is defined only under those interpretations in which all terminating computations yield the same result. In this case our functionals and operators are only partial recursive, not recursive. Another interesting question is – what happens if we allow equality as a special (always defined) binary predicate? This may be thought analogous to permitting copying instructions $(L_i := L_j)$ – but in fact is likely to make much more difference.

**Acknowledgements**

This work was done under a grant from the Science Research Council. It clearly owes much to Luckham, Park and Paterson (1967). I am grateful to my colleagues at Swansea and to David Park for illuminating discussions; also to Martin Weiner for Theorem 4.4.

## REFERENCES

Dekker, J.C.E. (1954) A theorem on hypersimple sets, *Proc. Am. Math. S.*, **5**, 791–6.

Luckham, D.C., Park, D.M.R. & Paterson, M.S. (1967) *On formalised computer programs.* Programming Research Group, University of Oxford.

McCarthy, J. (1963) A basis for a mathematical theory of computation, *Computer Programming and Formal Systems*, pp. 33–70 (eds Braffort P. & Hirschberg D.). Amsterdam: North Holland.

MacLane, S. and Birkhoff, G. (1967) *Algebra.* London: MacMillan.

Milner, R. (1969) Some equivalence relations on program schemes. *Research Memorandum No.* 5. Computer Science Department, University College of Swansea.

Paterson, M.S. (1967) Equivalence problems in a model of computation. Ph.D. thesis. Cambridge University.

Rogers, H., Jr (1967) *Theory of Recursive Functions and Effective Computability.* New York: McGraw-Hill.

Weiner, M. (1969) The degree of unsolvability of the problem of equivalence of two program schemes, *Research Memorandum No.* 9. Computer Science Department, University College of Swansea.

## APPENDIX

We describe here how from the Turing machine $TM_i$ which computes $\phi_i$ a program scheme $P_{w(i)}$ may be effectively constructed (i.e., $w$ is a recursive function) which in some sense mimics the behaviour of $\phi_i$. Full details are given by Paterson (1967); what follows is a briefer, intuitive description.

We also include here the proofs of 3 theorems which depend on the properties of $P_{w(i)}$.

### The Turing Machine simulator $P_{w(i)}$

Suppose that the states of $TM_i$ are a finite subset $Q$ of $\{q_0, q_1, \ldots\}$ and the tape alphabet is the finite set $\{a_0, a_1, \ldots, a_k\} = A$. Then a *configuration* of $TM_i$ is $Lq_ia_jR$ where $L$, $R \in A^*$ are the left, right non-blank portions of the tape, $q_i \in Q$, and $a_j \in A$ is the scanned symbol. We may restrict *initial configurations* to be of the form $q_0\bar{n}$ where $\bar{n}$ is some conventional representation in $A^*$ of the input integer $n$. Then a *configuration sequence* is a string in $A \cup Q \cup \{\varepsilon\}$ of the form $C_0\,\varepsilon\,C_1\,\varepsilon\,C_2\,\varepsilon\ldots$ where the $C_i$ are configurations ($C_0$ initial). The sequence is *admissible* if for all $j$, $C_{j+1}$ follows correctly from $C_j$ by the transition rules of $TM_i$. In this case the sequence is finite iff a terminating configuration is reached.

We imagine these sequences coded in binary. Then $P_{w(i)}$ will behave as follows on any free interpretation $I^f$ (unlike our usual schemes, $P_{w(i)}$ has two exits, *accept* and *reject*):

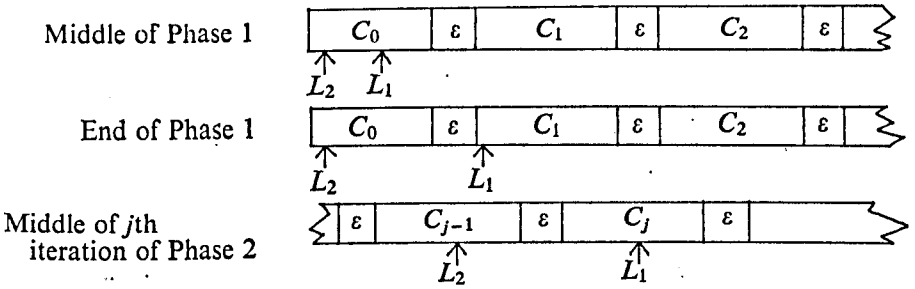
If no finite initial segment of $s(I^f)$ is (the binary coding of) an initial configuration, $P_{w(i)}$ will *diverge*: otherwise

If a finite initial segment of $s(I^f)$ is an admissible (terminating) configuration sequence, $P_{w(i)}$ will *accept*:

If $s(I^f)$ is an admissible (non-terminating) configuration sequence, $P_{w(i)}$ will *diverge*:

In any other case (i.e., where $s(I^f)$ is an inadmissible configuration sequence, or no configuration sequence at all), $P_{w(i)}$ will *reject*.

$P_{w(i)}$ has just two registers, and uses them like the heads of a two-headed, one-tape, read-only finite automaton, examining $s(If)$ (analogous to the tape of the automaton) in a one-way scan, starting together at the beginning and moving along it by applying $L_1 := F(L_1)$ to move the first head, $L_2 := F(L_2)$ to move the second. First (Phase 1) $L_1$ checks for an initial configuration; thereafter $L_1$ and $L_2$ advance together ($L_1$ leading), always checking for a proper consecutive pair of configurations. Thus:




The flow chart of $P_{w(i)}$ is summarized in figure 1. The feasibility of the detailed construction of Phase 2 depends on the fact that the change from $C_j$ to $C_{j+1}$ is a small local change. For simplicity, Phase 1 is made to diverge on all inadmissible initial configurations, though the only inadmissibility which it cannot detect (and reject) in a finite number of steps is that consisting of (the encoding of) an infinite 'integer'.

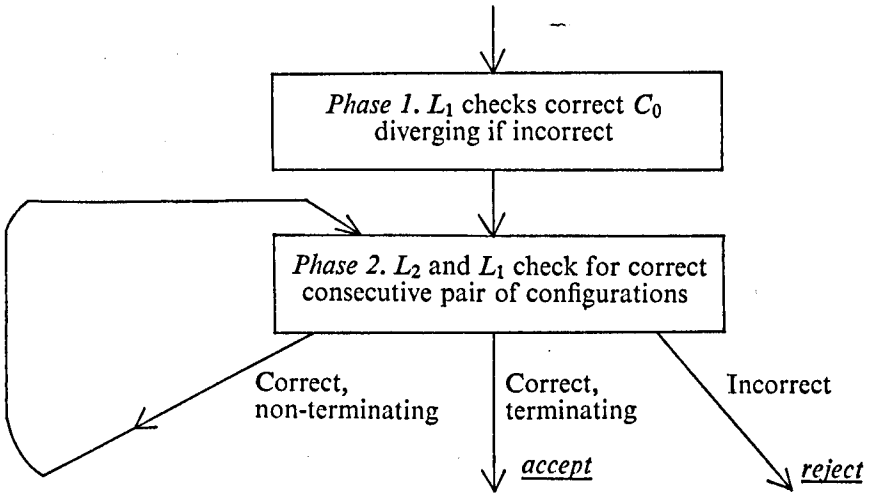



Figure 1. The flow chart of $P_{w(i)}$.

*Theorem* 4.2

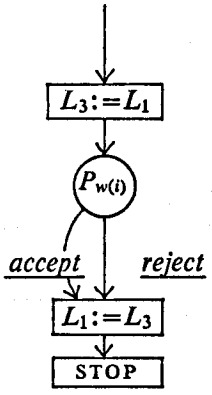
For each $W_i \neq N$ there is a $W_{f(i)}$ which is a $t$-set, and such that

$$W_i \leqq_1 W_{f(i)},\ W_{f(i)} \leqq_m W_i.$$

(Note that we do not show $f$ recursive, so we do not prove that $W_{f(i)}$ is *effectively* obtainable from $W_i$.)

*Proof.* If $W_i = \emptyset$, then we can take $W_{f(i)}$ to be the $t$-set of any always diverging scheme, for then $W_{f(i)} = \emptyset$.

If $W_i \neq \emptyset$ and $\neq N$, there is some $y_0 \in W_i$, $y_1 \in \overline{W_i}$. Now take $W_{f(i)}$ to be the $t$-set of the following scheme




(1) To prove $W_i \leqq_1 W_{f(i)}$. Take any integer $n$. Let $S_{g(n)}$ be the binary encoding of the initial configuration $q_0\bar{n}$. Clearly $g$ is a 1–1 recursive function.

Now we have

$n \in w_i \Leftrightarrow P_{w(i)}$ diverges on no $I^f > S_{g(n)}$ (there is no admissible configuration sequence).

$\Leftrightarrow \langle g(n), 0\rangle \in W_{f(i)}$ (since the constructed scheme gives output $L$ under any $I^f$).

Thus $W_i \leqq_1 W_{f(i)}$ via the (clearly 1–1) recursive function $\lambda n \,.\, \langle g(n), 0\rangle$.

(2) To prove $W_{f(i)} \leqq_m W_i$. Take any $x = \langle k, m\rangle$. We can effectively decide from $k$ which of the following alternatives holds:

(a) There is no (encoding of an) initial configuration $< S_k$. In this case $P_{w(i)}$ can diverge in Phase 1 on an $I^f > S_k$, so $\langle k, m\rangle \notin W_{f(i)}$.

(b) (a) is false, but $S_k$ is not an initial segment of an admissible configuration sequence. So $P_{w(i)}$ rejects every $I^f > S_k$, and we have

$$m = 0 \Leftrightarrow \langle k, m\rangle \in W_{f(i)}.$$

(c) (a) and (b) are false. $S_k$ is an initial segment of some admissible configuration sequence with initial configuration $q_0\overline{n(k)}$, where $n$ is a recursive function. In this case, if $m = 0$,

$$\langle k, m\rangle \in W_{f(i)} \Leftrightarrow P_{w(i)} \text{ converges on every } I^f > S_I$$
$$\Leftrightarrow n(k) \in W_i$$

and clearly if $m \neq 0$, $\langle k, m\rangle \notin W_{f(i)}$.

Now, remembering that $y_0 \in W_i$, $y_1 \notin W_i$, define a recursive $h$ as follows:

$h(x) =$ **if** $m \neq 0$ **then** $y_1$
**else if** (a) **then** $y_1$
**else if** (b) **then** $y_0$

**else if** (c) **then** $n(k)$ *where* $x = \langle k, m \rangle$.

From the preceding remarks, we have

$$x = \langle k, m \rangle \in W_{f(i)} \Leftrightarrow h(x) \in W_i, \text{ so } W_{f(i)} \leqq_m W_i \text{ via } h.$$

*Theorem* 4.4

The relation $P_i \leqq {}^tP_j$ is in the same $m$-degree of solvability as the relation $W_x \subset W_y$. In fact:

(1) $\{\langle i, j \rangle \mid P_i \leqq {}^tP_j\} \leqq_m \{\langle x, y \rangle \mid W_x \subset W_y\}$.

(2) $\{\langle x, y \rangle \mid W_x \subset W_y\} \leqq_1 \{\langle i, j \rangle \mid P_i \leqq P_j\}$.

*Proof.* (1) We know that $P_i \leqq {}^tP_j \Leftrightarrow W_{t(i)} \subset W_{t(j)}$ (Theorem 4.1) and we can therefore decide the former relation if we can decide $W_x \subset W_y$ for $\langle x, y \rangle = \langle t(i), t(j) \rangle$.

(2) We can use the Turing machine simulator. The aim, given $W_x$, $W_y$, is to construct effectively two schemes $P_{f(x,y)}$, $P_{g(x,y)}$ such that

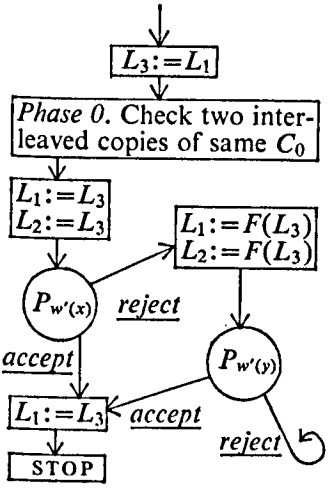
$$W_x \subset W_y \Leftrightarrow P_{f(xy)} \leqq {}^tP_{g(x,y)}.$$

We are prevented from using $P_{w(x)}$, $P_{w(y)}$ (or simple modifications of them) as the two schemes since the admissible configuration sequences of $TM_x$, $TM_y$ will in general bear no resemblance to one another even when $W_x \subset W_y$. Instead we use two schemes, *each* of which is a combination of simple modifications of $P_{w(x)}$ and $P_{w(y)}$.

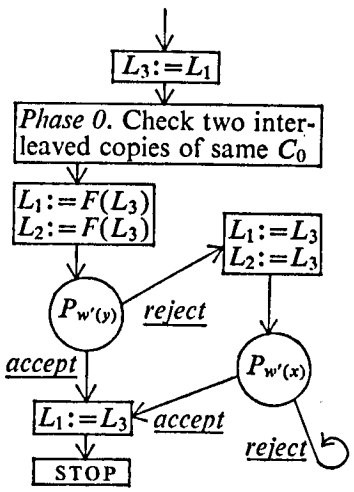
We modify $P_{w(x)}$ in a simple way to make it concerned only with *alternate* members of $s(I^j)$; this we do by doubling up $L_1 := F(L_1)$, and similarly doubling up $L_2 := F(L_2)$, wherever they occur in $P_{w(x)}$. Call the resulting scheme $P_{w'(x)}$; $P_{w'(y)}$ is formed likewise.

$P_{f(x,y)}$ and $P_{g(x,y)}$ are illustrated in figure 2.

$P_{f(x,y)}$ performs as follows: first, using $L_1$ only (Phase 0) it examines




The scheme $P_{f(x,y)}$




The scheme $P_{g(x,y)}$

Figure 2

$s(I^f)$ to ensure that it starts with two interleaved copies of the *same* initial configuration $q_0\bar{n}$ for some $n$; if not it diverges. If correct, it returns to the start of $s(I^f)$ and applies $P_{w'(x)}$ to the *even* members of $s(I^f)$; on rejection (and only then) it returns to the start and applies $P_{w'(y)}$ to the *odd* members of $s(I^f)$; on rejection it enters some loop.

$P_{g(x,y)}$ performs exactly alike but with $P_{w'(x)}$, $P_{w'(y)}$ and *even*, *odd* interchanged. (There is some duplication between the work of Phase 0 and the work of Phase 1 in $P_{w'(x)}$, $P_{w'(y)}$; this inefficiency simplifies the construction.)

We now assert that $W_x \subset W_y \Leftrightarrow P_{f(x,y)} \leqq^t P_{g(x,y)}$. Thus, the 1–1 reducibility is shown, since $f$, $g$ are clearly 1–1 recursive functions. To prove the assertion:

($\Rightarrow$). Suppose $P_{f(x,y)} \not\leqq^t P_{g(x,y)}$. Then under some $I^f$ $P_{g(x,y)}$ diverges and $P_{f(x,y)}$ converges. $P_{g(x,y)}$ may diverge in only 4 ways:

(a) In Phase 0.

(b) In $P_{w'(y)}$.

(c) In $P_{w'(x)}$.

(d) On rejection by $P_{w'(x)}$.

But in cases (a), (c), (d) $P_{f(x,y)}$ will also diverge, so $P_{g(x,y)}$ must diverge in $P_{w'(y)}$ under $I^f$. Since $P_{f(x,y)}$ converges, $P_{w'(x)}$ must accept. But then $n \in W_x$, $n \notin w_y$ where $q_0\bar{n}$ is the initial configuration. Therefore $W_x \not\subset W_y$.

($\Leftarrow$). Suppose $W_x \not\subset W_y$. Then for some $n$, $n \in W_x$, $n \notin W_y$. Now consider an $I^f$ whose even numbers have as an initial segment the *finite* admissible configuration sequence of $TM_x$ starting with $q_0\bar{n}$, and whose odd members are the *infinite* admissible configuration sequence of $TM_y$ starting with $q_0\bar{n}$. Clearly $P_{f(x,y)}$ accepts in $P_{w'(x)}$ and $P_{g(x,y)}$ diverges in $P_{w'(y)}$. Therefore $P_{f(x,y)} \not\leqq^t P_{g(x,y)}$.

*Theorem* 5.2 (3)

The $\equiv^{p'}$ classes do not form a lattice under $\leqq^{p'}$.

*Proof.* We know that $K = \{x \mid \phi_x(x) \text{ converges}\}$ is not recursive. Now let $\phi_k = \lambda x \,.\, \phi_x(x)$. Consider the schemes in figure 3.

It is clear that $A$, $B \leqq^{p'} C$. However, we show that the existence of a *least* upper bound for $A$, $B$ implies $K$ recursive.

Note that the above schemes start $P_{w(k)}$ on the second, not the first digit (0 or 1) in $s(I^f)$. For any $I^f$, let $I^{f'}$ be such that $s(I^{f'}) = [s(I^f)$ with first digit removed]. Call an $I^f$ *unacceptable* if $P_{w(k)}$ rejects $I^{f'}$. Otherwise $I^f$ is *acceptable*. Denote by $I^f_z$ any acceptable $I^f$ for which $I^{f'}$ starts with an initial configuration $C_0 = q_0\bar{z}$.

Now suppose $M$ is lub for $A$, $B$. Then $M$ has the following properties:

(1) $M$ converges on all $I^f_z$, $z \in K$ (since $B \leqq^{p'} M$).

(2) $M$ converges on all unacceptable $I^f$ (since $A \leqq^{p'} M$).

(3) $M$ strongly diverges on all $I^f_z$, $z \notin K$ (since $M \leqq^{p'} C$).

We can now show the following:

*Lemma* 1

$M$ *never* evaluates $T_I(L_I)$ when $I$ is $I^f_z$ and $z \in K$ (proof below).

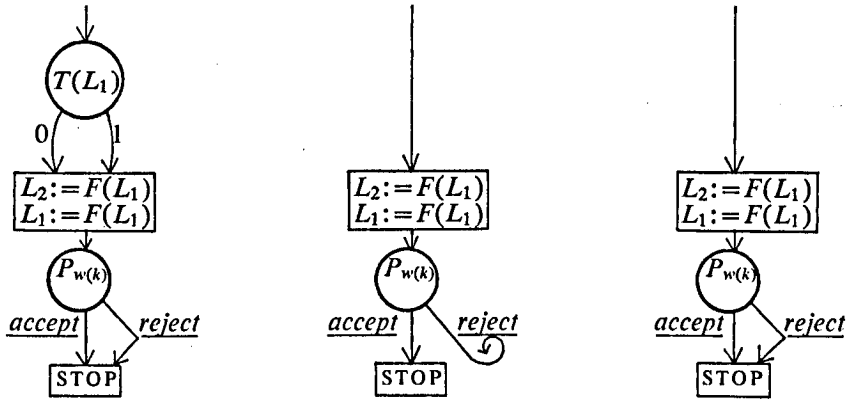



Figure 3

*Lemma* 2

*M always* evaluates $T_I(L_I)$ at some point, when $I$ is $I'_z$ and $z \notin K$ (proof below).

Hence we have the following decision procedure for $z \in K$: execute $M$ under $I=I'_z$. If $z \in K$, $M$ will converge. If $z \notin K$, $M$ will at some point evaluate $T_I(L_I)$. One or other must occur, never both. Thus the existence of $M$ implies that $K$ is recursive, and so there can be no lub for $A$, $B$.

*Proof of Lemma* 1. Suppose $M$ evaluates $T_I(L_I)$ under some $I=I'_z$, $z \in K$. Then by rendering $T_I(L_I)$ undefined we obtain a partial interpretation $I'$ in which $\mathrm{Conv}(B, I')$ but $\sim\mathrm{Conv}(M, I')$. [Note that $T_I(L_I)$ is never evaluated in $B$.] This contradicts $B \leqq^{p'} M$.

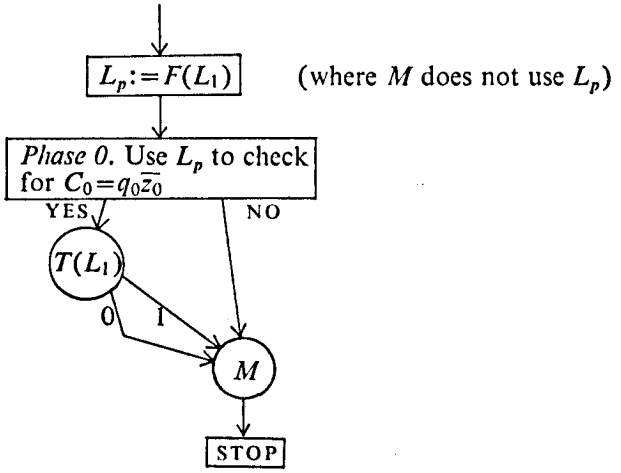



Figure 4. The scheme $M'$.

*Proof of Lemma* 2. Suppose that for some $z_0 \notin K$, $M$ never evaluates $T_I(L_I)$ under $I=I^{\prime}_z$. Now construct $M'$ to do $M$, after first looking for $q_0\bar{z}_0$ at the start of $s(I^{f'})$, and only then performing $T(L_1)$.

We show that $M'$ is a strictly lesser upper bound than $M$.

(1) $M' \leqq^{p'} M$. Immediate.

(2) $M \not\leqq^{p'} M'$. For define $I$ as follows: let $I$ be an inadmissible $I^f$ where $s(I^{f'})$ starts with $q_0\bar{z}_0$, but make $T_I(L_I)$ undefined. Then $\mathrm{Conv}(M, I)$, $\sim\mathrm{Conv}(M', I)$.

(3) $A \leqq^{p'} M'$. This follows from $A \leqq^{p'} M$, together with the fact that Phase 0 of $M'$ only performs evaluations also performed in Phase 1 of $P_{w(k)}$. (The latter must look at least as far along $s(I^{f'})$ as Phase 0, and must test every digit.)

(4) $B \leqq^{p'} M'$. This follows for the same reason as (3), together with the fact that $M'$ only does $T_I(L_I)$ when $\sim\mathrm{Conv}(B, I)$, since $z_0 \in K$.

But $M$ is a lub, hence for every $z \notin K$, $M$ evaluates $T_I(L_I)$ under $I^{\prime}_z$.

# 3

# Fixpoint Induction and Proofs of Program Properties

David Park
School of Computer Science
University of Warwick

Given a computer program and a certain property, there are now a number of techniques (Ashcroft 1969, Cooper 1969, Manna 1969, Manna and Pnueli 1969) for obtaining in a formal system an expression whose formal proof or disproof constitutes a verification that the program has the property in question. The techniques stemming from the ideas of Floyd (1967), developed and formalized by Manna (1969), are particularly interesting in that a wide variety of properties of programs turn out to be representable as validity or satisfiability problems in first-order predicate calculi; and in this form established mechanical theorem-proving techniques may, one hopes, be relevant to automating the verification process.

This paper describes one approach to developing a general theory relevant to the formalization and proof of arbitrary properties of computer programs. From this theoretical point of view, we are interested in the following questions:

(1) *Representability*. In contrast to the results of Manna (1969), it is known from previous theoretical work (Luckham, Park and Paterson 1967, Paterson 1968) that some comparatively straight-forward properties of programs (e.g., the 'strong' equivalence property between program schemas, namely that they compute the same partial functions on all interpretations) cannot easily be represented as validity or satisfiability problems in a first-order language. More precisely, the fact (Luckham, Park and Paterson 1967) that neither strong equivalence nor strong non-equivalence is partially decidable implies the following assertion: that *a formal system in which strong equivalence problems are representable as validity/satisfiability problems is necessarily incomplete with respect to provability*, i.e., that the valid formulas of such a formal system are not recursively enumerable. This eliminates from consideration the pure first-order predicate calculus, although a 'pure' predicate calculus of some sort is a natural candidate for representing properties of 'pure' program schemas, i.e., schemas as considered in Luckham, Park and

Paterson (1967), where no constraints are imposed on interpretations of the basic functions and tests involved. With this in mind, Cooper (1969) showed that strong equivalence problems for program schemas can in fact be expressed as validity problems in the pure second-order calculus (taking validity in the narrow sense to be defined in section 1, as against 'Henkin-validity'; we are concerned with validity in first-order structures, with no constraints placed on the ranges of predicate variables). We will see here that this representation can be regarded as a 'natural' one, in that there are straightforward characterizations of the partial functions computed by schemas in terms of 'convergence formulas' which can be defined in a second-order calculus.

(2) *Proof Strategies.* Following Cooper (1969), we are concerned with general strategies for proving the second-order sentences obtained by carrying out the representation in terms of convergence formulas. In particular we are interested in finding specialized derived rules of the second-order calculus, particularly rules permitting us to deduce second-order from first-order formulas. The insight necessary here is in fact a very simple one, of the relationship between the Fixpoint Theorem of lattice theory (the Knaster–Tarski theorem), and a general form of argument by mathematical induction (which we call *Fixpoint Induction*) which is applicable to properties of minimal fixpoints, and applicable therefore to convergence formulas, which characterize the graphs of the partial functions computed by schemas in terms of the lattice-theoretic notion. This simple idea is nevertheless a very important one theoretically, since both the 'recursion induction' rule of McCarthy (1963), and analogs of the proof methods of Floyd (1967), are justifiable as direct applications of the Fixpoint Induction principle.

Some further elementary results concerning fixpoints turn out to imply both generalizations of and alternatives to the Manna–Pnueli (1969) methods associating first-order satisfiability/validity results with properties of programs. Finally, we present a stronger form of the Fixpoint Induction principle, which turns out to permit 'nested' induction arguments of various sorts.

We have chosen here to obtain these results for *equation schemas*, sets of formalized recursive function definitions of the sort considered by Manna and Pnueli (1969). There is a well-known general method (*see* McCarthy 1963, Luckham, Park and Paterson 1967) for characterizing equivalences between program schemas in terms of equivalences between equation schemas; the application of these proof methods to equivalence and other properties of program schemas can be obtained by following through this characterization. Our motivation for preferring to deal with equation schemas lies in the fact that they model computer programs involving (possibly recursive) subroutine calls, which program schemas do not do. On the other hand, we could also have chosen to deal with *nondeterministic* equation schemas, which generalize the sort of equation schema considered

here by permitting one to define 'multiple valued' functions (using, e.g., the basic function *amb* described in McCarthy (1963)), since the relations that non-deterministic schemas can be regarded as computing can also be characterized by convergence formulas.

## 1. PRELIMINARIES

In this section we set up basic terminology and notation for *equation schemas*, our program-like objects, and for their associated second-order predicate calculi, in which their properties are to be formalized. An alternative but essentially equivalent scheme of notation is given in Manna and Pnueli (1969).

We want to consider as formal objects sets of recursive function definitions expressed in terms of various 'basic' functions, predicate and constants, in the spirit of McCarthy (1963). We refer to the symbols for these basic objects as a *formal basis* for *equation schemas* constructed in terms of them. The following notational styles will be used for elements of formal bases:

*basic function symbols*: $f, g, f_0, g_0$, etc.
*basic predicate symbols*: $P, Q, P_0, Q_0$, etc.
*individual constants*: $T, F, a, b, a_0, b_0$, etc.

It will be assumed that the two individual constant symbols '$T$', '$F$' are present in any formal basis (for ease of exposition; this device permits recursive predicates to be defined in the same form as recursive functions; '$T$', '$F$' are to denote representations of the truth-values 'true', 'false').

*Terms* over a given formal basis involve symbols of the basis, various delimiters, and also symbols for *derived functions* and *individual variables*, using the following styles:

*derived functions*: $\phi, \psi, \phi_0, \psi_0$, etc.
*individual variables*: $x, y, z, x_0, y_0, z_0$, etc.

(*Note.* In order to define various substitution operations for variables in terms or in formulas, it is convenient to give some of them a privileged status; in the case of individual variables, we use the set $\{x_0, x_1, x_2, \ldots\}$; other symbols for individual variables can be regarded, when they occur, as metavariables ranging over this set; the occurrences of such a metavariable within a formula, say, are to be regarded as replaced consistently by some $x_i$ which does not otherwise occur in the formula.)

With each basic predicate, basic function and derived function symbol is associated in context a degree $\geqslant 0$, which will not be indicated explicitly here. When $\phi$, say, has degree $n$, we say it is an $n$-ary derived function symbol, and similarly for the other categories.

*Definition* 1.1. The class of terms (over some formal basis) is the smallest class of strings satisfying the following:

(1) Any individual variable or basic constant is a term.

(2) If $\tau_1, \tau_2, \tau_3$ are terms, so is

**if** $\tau_1$ **then** $\tau_3$ **else** $\tau_3$.

(3) If $\tau_1, \tau_2, \ldots \tau_n$ are terms, so are

(a) $f(\tau_1, \tau_2, \ldots \tau_n)$, any $n$-ary basic function $f$.

(b) $P(\tau_1, \tau_2, \ldots \tau_n)$, any $n$-ary basic predicate $P$.

(c) $\phi(\tau_1, \tau_2, \ldots \tau_n)$, any $n$-ary derived function $\phi$.

(*Note.* In examples we also use conventionally infixed basic functions and predicates, e.g. '=', '+', '*', etc; these are clearly inessential departures from (3)).

An $N$-ary *equation schema* is a sequence of $N$ *recursion equations*, of the form:

$$\left.\begin{aligned} &\phi_0(x_1, x_2, \ldots, x_{n_0}) \Leftarrow \tau_0 \\ &\phi_1(x_1, x_2, \ldots, x_{n_1}) \Leftarrow \tau_1 \\ &\qquad\vdots \\ &\phi_{N-1}(x_1, x_2, \ldots, x_{n_{N-1}}) \Leftarrow \tau_{N-1} \end{aligned}\right\} \qquad (1.1.1)$$

each $\tau_i$ being a term, and each $\phi_i$ being an $n_i$-ary derived function symbol, $0 \leqslant i < N$. Individual variables and derived function symbols mentioned in $\tau_i$ are restricted to $x_1, x_2, \ldots x_{n_i}, \phi_0, \phi_1, \ldots \phi_{N-1}$ respectively.

An *interpretation* $I$ of a formal basis is given by specifying a *domain* $D$, $n$-ary functions $f^I: D^n \to D$ for each $n$-ary basic function $f$, $n$-ary relations $P^I \subseteq D^n$ for each $n$-ary basic predicate $P$, and elements $c^I \in D$ for each individual constant $c$ of the formal basis. (Interpretations are therefore 'first-order relational structures' of the sort investigated in the model theory of first-order predicate calculi.)

Given such an interpretation $I$, the schema (1.1.1.) determines $N$ partial functions $\phi_i^I: D^n \to D$; obtained by following the standard evaluation rules for recursion equations (as in McCarthy 1963). In the formulation assumed here $n$-ary functions and $n$-ary predicates can be used interchangeably; a predicate is to be treated as a function with the range $\{T^I, F^I\}$ of truth-value representations, in the natural way; the value of a conditional expression is to be undefined unless the value of the first term is $T^I$ or $F^I$. The only requirement on $T^I$, $F^I$ will be that they are distinct elements of $D$.

With each formal basis is associated the vocabulary of a second-order predicate calculus with identity, with the same basic function symbols, basic predicate symbols and individual constants. Formulas of the calculus involve also a standard set of connectives, quantifiers, delimiters, etc., with individual variables and predicate variables. We use the following styles for variables:

*individual variables*: $x, y, z, x_0, y_0, z_0$, etc. (as for schemas)

*predicate variables*: $X, Y, Z, X_0, Y_0, Z_0$, etc.

(*Note.* The variables $\{x_0, x_1, x_2 \ldots\}$, $\{X_0, X_1, X_2 \ldots\}$ have a privileged status, as explained above in connection with individual variables in equation schemas. Predicate variables have degrees in context, as for basic predicates.)

The formation rules of this second-order calculus are obtained by extending

those of a standard first-order calculus with equality so as to permit quantification over predicate variables. The rules are assumed to permit compound terms involving individual variables and constants and basic functions. An account of the fundamental properties of second-order calculi is given in Church (1956).

We summarize below the notions that are important here. It should be emphasized that the semantic notions involved are 'naive', in that they are defined by reference to 'first-order' interpretations only, without the consideration of possible restrictions on the ranges of predicate variables which is needed for the definition of *Henkin-validity*.

The rules for *satisfaction* of formulas are straightforward extensions of the corresponding rules for a first-order language; they specify when a formula is said to be satisfied in an interpretation $I$ by an assignment of elements of the domain $D$ to individual variables and of relations over $D$ to predicate variables – e.g., a formula $(\exists X)\mathscr{F}$ is satisfied if there is an assignment which satisfies $\mathscr{F}$ in $I$, which agrees with the given assignment on all variables except possibly $X$, and which assigns to $X$ some $n$-ary relation over $D$, where $X$ is $n$-ary; $(\forall X)\mathscr{F}$ is satisfied if $(\exists X)\sim\mathscr{F}$ is not satisfied. A formula which does not involve quantification over a predicate variable is said to be *first-order*. A formula can have free predicate variables as well as free individual variables. Formulas without free variables of either sort are *closed formulas*. Satisfaction of closed formulas depends only on the interpretation, and is clearly independent of the particular assignment chosen.

If a formula $\mathscr{F}$ has free variables at most $x_0, x_1, \ldots, x_{n-1}, X_0, X_1, \ldots, X_{N-1}$, it has a $\langle n, N\rangle$-*extension* in $I$, which is the set of $\langle a_0, a_1, \ldots, a_{n-1}, A_0, A_1 \ldots A_{N-1}\rangle$ such that $\mathscr{F}$ is satisfied by (some or all assignments) assigning $a_i$ to $x_i$, $0\leqslant i<n$, and $A_j$ to $X_j$, $0\leqslant j<N$. A tuple in the $\langle n, N\rangle$-extension of $\mathscr{F}$ is said to *satisfy* $\mathscr{F}$. The $\langle n, 0\rangle$-*extension*, if any, of $\mathscr{F}$ is the *n-extension* (or simply *extension*) of $\mathscr{F}$ in $I$.

In general one is interested in constraining interpretation of the formal basis for equation schemas; we assume that these constraints are in the form of a set of closed formulas which are *axioms* for the associated second-order predicate calculus (which then becomes an applied predicate calculus). In any case, we must assume the following as an axiom

$$[T\neq F]. \tag{1.1.2}$$

*Permissible interpretations* of the calculus are just those which satisfy all of the given axioms for the calculus ((1.1.2) then ensures that in every permissible interpretation $I$, $T^I\neq F^I$).

A formula $\mathscr{F}$ is *valid* in the calculus if it is satisfied by all assignments in all permissible interpretations. $\mathscr{F}$ is *satisfiable* if $\sim\mathscr{F}$ is not valid. Two formulas $\mathscr{F}$, $\mathscr{G}$, are equivalent (written $\mathscr{F}\equiv\mathscr{G}$), if they are satisfied by just the same assignments in all permissible interpretations of the calculus.

It is well known that second-order predicate calculi with infinite permissible interpretation are incomplete with respect to provability, with the above

definition of 'validity'; i.e., that there is no effective notion of 'provability' whereby all and only the valid formulas of such a calculus are provable. This makes it necessary to treat with slight caution 'semantic' theorems of the sort to be proved here, insofar as they claim to establish 'derived rules' of the calculus. Speaking very strictly, the logical status of the rule is not made completely clear by this sort of argument, which would need to be rephrased in terms of Henkin-validity to establish derived rules in a formal development of the calculus. In fact this rephrasing can be carried through for the metatheorems described below. (It would be astonishing if this were not so.)

*Predicate calculus notation.* (Omitting standard abbreviation).

1. Script letters $\mathscr{C}, \mathscr{D}, \mathscr{E}, \mathscr{F}$, etc., standard for arbitrary formulas of the calculus. Where it is unambiguous, we use a predicate variable $X$, say, to stand for the formula $X(x_0, x_1, \ldots, x_{n-1})$, if $X$ is $n$-ary.

2. Given a formula $\mathscr{F}$, and terms $\tau_0, \tau_1, \ldots, \tau_{n-1}$, $\mathscr{F}(\tau_0, \tau_1, \ldots, \tau_{n-1})$ is to be the result of simultaneously substituting $\tau_i$ for all free occurrences of $x_i$ in $\mathscr{F}$, $0 \leqslant i < N$.

3. Given formulas $\mathscr{F}, \mathscr{G}_0, \mathscr{G}_1, \ldots, \mathscr{G}_{,N-1}$ $\mathscr{F}(\mathscr{G}_0, \mathscr{G}_1, \ldots, \mathscr{G}_{N-1})$ is to be the result of

(a) by a suitable alphabetic change of bound variables of $\mathscr{F}$, ensuring that no bound variable of $\mathscr{F}$ has free occurrences in $\mathscr{G}_i$, $0 \leqslant i < N$; followed by

(b) simultaneously substituting $\mathscr{G}_i(\tau_0, \tau_1 \ldots \tau_n)$ for all subformulas of the form $X_i(\tau_0, \tau_1 \ldots \tau_n)$ with $X_i$ free, $0 \leqslant i < N$.

*Example*

$$\text{if } \mathscr{F} \equiv (\forall X_1)[X_0(x_1) \rightarrow (\forall x_1)[X_1(x_0, x_1) \wedge X_0(x_1)]]$$
$$\text{then } \mathscr{F}(X_1(x_1, x_0), X_1(x_0, x_1))(f(a), g(a))$$
$$\equiv (\forall X_2)[X_1(g(a), g(a)) \rightarrow (\forall x_2)[X_2(f(a), x_2) \wedge X_1(g(a), x_2)]]$$

4. Boldface letters are used to indicate argument lists of some (unspecified) length, e.g., $\mathscr{F}(\mathbf{Y})(\mathbf{y})$ abbreviates $\mathscr{F}(Y_0, Y_1 \ldots Y_{N-1})(y_0, y_1, \ldots, y_{n-1})$, for some $N, n \geqslant 0$. Unless otherwise stated, $N, n$ are assumed sufficiently large that $\mathscr{F}$ has no free variables of the form $x_i$, $i \geqslant n$, or $X_j$, $j \geqslant N$. (Note that $\mathscr{F} \equiv \mathscr{F}(\mathbf{X})(\mathbf{x})$ etc. with this convention.) Similarly $(\forall \mathbf{x})\mathscr{F}$, $(\exists \mathbf{X})\mathscr{F}$ etc. abbreviate formulas $(\forall x_0)(\forall x_1) \ldots (\forall x_{n-1})\mathscr{F}$, $(\exists X_0)(\exists X_1) \ldots (\exists X_{N-1})\mathscr{F}$, etc., with the same implicit conventions concerning $n$, $N$ as above.

## 2. THE FIXPOINT THEOREM

This result from lattice theory [Knaster–Tarski theorem (Tarski 1955)] is of fundamental importance if one is interested in setting up a study of programs as 'mathematical' objects, e.g., if one is interested in proving theorems about the results of running programs. (The relevance of the lattice theory result is also emphasized in the recent work of Scott and de Bakker (private communication), and Hans Bekić.) The significance of the Knaster–Tarski theorem in the current context is twofold:

(a) it provides a straightforward 'set-theoretic' characterization of the partial function computed by a program, as opposed to the more familiar 'constructive' characterization obtained by describing in detail a particular evaluating mechanism. (The relevance to recursive function theory is well known to recursion-theorists, in that the Knaster–Tarski theorem is related to a 'weak' form of the Recursion Theorem, see, e.g., Rogers (1967, Chapter 11). The 'weakness' is that the fixpoint is defined nonconstructively; in the form given here its definition is also impredicative.)

(b) the characterization obtained has a direct association with various principles of (mathematical) induction: one can regard some existing methods of proving facts about programs, e.g., recursion induction, the methods of Floyd (1967), as being dependent upon this association.

In fact we will need the fixpoint results only as applied to complete boolean algebras – e.g., to the algebra of all subsets of some set. (2.1, 2.2 generalize to complete lattices, 2.3 to uniquely complemented complete lattices.)

Let $\Lambda = \langle L, ', \cup, \cap, 0, 1 \rangle$ be a complete boolean algebra; define the partial ordering $l \subseteq m$ by $l \cap m = l$, for $l, m \in L$; since $\Lambda$ is complete, then any $M \subseteq L$ has a l.u.b.$\cup M$, and g.l.b.$\cap M$ with respect to '$\subseteq$' ($\cup \emptyset = 0$, $\cap \emptyset = 1$).

A map $c: L \to L$ is *monotone* if, whenever $l \subseteq m$, then also $c(l) \subseteq c(n)$, i.e., if $c$ preserves the partial ordering '$\subseteq$'.

Define, for any map $c: L \to L$,

$$Conv(c) = \cap \{l \mid c(l) \subseteq l\} \tag{2.0.1}$$

Note that it follows immediately from (2.0.1) that

$$c(l) \subseteq l \text{ implies } Conv(c) \subseteq l. \tag{2.0.2}$$

The Fixpoint Theorem is the result that, if $c$ is monotone, then $Conv(c)$ is a 'fixpoint' of $c$; an immediate corollary is that $Conv(c)$ is the *minimal* fixpoint of $c$ with respect to '$\subseteq$'.

**2.1.** *Fixpoint Theorem (Knaster–Tarski)*

2.1.1. $c(Conv(c)) = Conv(c)$

2.1.2. $Conv(c) = \cap \{l \mid c(l) = l\}$

*Proofs*

2.1.1. The usual proof can be put as follows:

(a) suppose $c(l) \subseteq l$; then $Conv(c) \subseteq l$, from (2.0.2); then $c(Conv(c)) \subseteq c(l)$, since $c$ is monotone; but $c(l) \subseteq l$ by hypothesis; so $c(Conv(c)) \subseteq l$. This holds for all $l \in \{l \mid c(l) \subseteq l\}$; therefore $c(Conv(c)) \subseteq \cap \{l \mid c(l) \subseteq l\} = Conv(c)$.

(b) conversely, from (a), $c(Conv(c)) \subseteq Conv(c)$; therefore $c(c(Conv(c))) \subseteq c(Conv(c))$, since $c$ is monotone; so $Conv(c) \subseteq c(Conv(c))$, by (2.0.2).

2.1.2. From (2.0.1), $Conv(c) = \cap \{l \mid c(l) \subseteq l\} \subseteq \cup \{l \mid c(l) = l\}$
Conversely, from 2.1.1. $Conv(c) = c(Conv(c))$; therefore $Conv(c) \in \{l / c(l) = l\}$; so $Conv(c) \supseteq \cap \{l \mid c(l) = l\}$.

*Note.* There is an alternative well-known characterization of $Conv(c)$ if $c$ is also *continuous*, in the sense that whenever $l_0 \subseteq l_1 \subseteq l_2 \subseteq \ldots$, then

$$c\left(\bigcup_{i=0}^{\infty} l_i\right) = \bigcup_{i=0}^{\infty} c(l_i).$$

For continuous monotone $c$,

$$Conv(c) = \bigcup_{i=0}^{\infty} c^i(0). \tag{2.1.1}$$

(Continuity is needed for the identity to hold; to show $\bigcup_{i=0}^{\infty} c^i(0) \subseteq Conv(c)$ does not require continuity.) The maps obtained below from equation schemas are in fact continuous; there are plausible general grounds for regarding continuity as an essential property of maps used to characterize computability for functions in any practical sense. Remarkably, however, continuity does not enter into the arguments used below, which depend solely on monotonicity.

The property (2.0.2) turns out to be the source of the Fixpoint Induction principle described below. In fact we will describe a stronger version, which provides a condition which is necessary as well as sufficient to establish $Conv(c) \subseteq l$ (it is easy to find counter examples to the converse of (2.0.2)).

**2.2.** *If $c$ is monotone, then for all $l \in L$ $(Conv(c) \cap c(l)) \subseteq l$ if and only if $Conv(c) \subseteq l$*

*Proof.* $Conv(c) \subseteq l$ implies $(Conv(c) \cap c(l)) \subseteq l$, trivially. Conversely, suppose $(Conv(c) \cap c(l)) \subseteq l$; let $m = (Conv(c) \cap l)$; since $m \subseteq Conv(c)$ and $c$ is monotone, $c(m) \subseteq c(Conv(c)) = Conv(c)$ by 2.1.1. Also, since $m \subseteq l$, $c(m) \subseteq c(l)$; therefore $c(m) \subseteq Conv(c) \cap c(l)$; but $Conv(c) \cap c(l) \subseteq l$, by hypothesis, and $Conv(c) \cap c(l) \subseteq Conv(c)$, trivially; therefore $c(m) \subseteq Conv(c) \cap l = m$; so $Conv(c) \subseteq m$, by (2.0.2); but $m \subseteq l$; so finally $Conv(c) \subseteq l$.
Note that (2.0.2) follows immediately from 2.2.

By dualizing the Fixpoint Theorem, an expression is obtained for the *maximal* fixpoint of a monotone map $c$. Given $c : L \to L$, define the *dual* map $c^* : L \to L$ by $c^*(l) = c(l')'$ (dashes indicate complementation). Note that $c^*$ is monotone if and only if $c$ is monotone.

**2.3.** *If $c$ is monotone, then*

2.3.1. $Conv(c^*)' = \cup \{l \mid c(l) \subseteq l\} = \cup \{l \mid c(l) = l\}$

2.3.2. $c(Conv(c^*)') = Conv(c^*)'$

2.3.3. $Conv(c^*) \cap Conv(c) = 0$

2.3.4. $Conv(c^*) \cup Conv(c) = 1$ *if and only if* $Conv(c)$ *is the unique fixpoint of* $c$.

(Proofs are straightforward, applying 2.1 to $c^*$.)

From 2.3.1, 2.3.2, $Conv(c^*)'$ is the maximal fixpoint of $c$. In the case that $\Lambda$ is the algebra of subsets of a set $D$, 2.3.3 shows that $c$ determines a three-fold partition of $D$, into mutually disjoint sets $Conv(c)$, $Conv(c^*)$, $(Conv$

$(c)' \cap Conv(c^*)')$. From 2.3.4 the last set is empty just in the case that $c$ has a unique fixpoint $Conv(c)$.

## 3. CONVERGENCE FORMULAS. FIXPOINT INDUCTION

*Notation.* For any set $D$, $\Lambda_n(D)$ is the boolean algebra of $n$-ary relations over $D$.

Given a predicate calculus formula $\mathscr{C}$ with free variables at most $x_0, x_1, \ldots x_{n-1}$, $X_0$, $X_0$ being $n$-ary, we can obtain, for any interpretation $I$ with domain $D$, a map $c$ on $\Lambda_n(D)$ which corresponds to $\mathscr{C}$ as follows:
for $A \subseteq D^n$, $c(A) = \{\langle a_0, a_1 \ldots a_{n-1}\rangle | \langle a_0, a_1 \ldots a, A_{n-1}\rangle$ *satisfies* $\mathscr{C}\}$.
In that case the fixpoints of $c$ are then just the solutions in $I$ to the 'recursion formula'

$$(\forall \mathbf{x})[\mathscr{C}(X)(\mathbf{x}) \leftrightarrow X(\mathbf{x})]. \tag{3.0.1}$$

The condition that $c$ be monotone (on every $I$) in the sense of section 2 is then just the following property of $\mathscr{C}$, for $N=1$;

**3.1.** *Definition* $\mathscr{C}$ *is monotone in each* $X_i$ $0 \leqslant i < N$ *if the formula*

$$\bigwedge_{i=0}^{N-1} (\forall \mathbf{x})[X_i(\mathbf{x}) \rightarrow Y_i(\mathbf{x})] \rightarrow (\forall \mathbf{x})[\mathscr{C}(\mathbf{X})(\mathbf{x}) \rightarrow \mathscr{C}(\mathbf{Y})(\mathbf{x})]$$

*is valid.*

Suppose $\mathscr{C}$ involves just the quantifiers $\forall, \exists$ and the connectives $\wedge, \vee, \sim$. An occurrence of $X_i$ in $\mathscr{C}$ is *positive* if it occurs in no subformula (or in an even number of subformulas) of the form $\sim \mathscr{F}$.

The following rule suffices to establish monotonicity for the cases considered here:

**3.2.** *Suppose* $\mathscr{C}$ *involves just* $\forall, \exists, \wedge, \vee, \sim$, *and all occurrences of* $X_i$ *in* $\mathscr{C}$ *are positive,* $0 \leqslant i < N$, *then* $\mathscr{C}$ *is monotone in each* $X_i$, $0 \leqslant i < N$.

(3.2 is established by induction on the number of connectives in $\mathscr{C}$.)

The relation over $D$, $Conv(c) = \cap\{A | c(A) \subseteq A\}$ is then the extension in $I$ of the following formula:

$$Conv(\mathscr{C})(\mathbf{x}) \equiv (\forall X)[(\forall \mathbf{y})[\mathscr{C}(X)(\mathbf{y}) \rightarrow X(\mathbf{y})] \rightarrow X(\mathbf{x})]. \tag{3.2.1}$$

The following are then immediate from 2.1

**3.3.** *If* $\mathscr{C}$ $(X)$ *is monotone in* $X$, *then*

3.3.1. $\mathscr{C}(Conv(\mathscr{C})) \equiv Conv(\mathscr{C})$

3.3.2. $Conv(\mathscr{C})(\mathbf{x}) \equiv (\forall X)[(\forall \mathbf{y})[\mathscr{C}(X)(\mathbf{y}) \leftrightarrow X(\mathbf{y})] \rightarrow X(\mathbf{x})]$.

More generally, we are concerned with finding solutions for $\mathbf{X}$ in $I$ to a system of simultaneous recursion formulas:

$$\left.\begin{array}{l} (\forall \mathbf{x})[X_0(\mathbf{x}) \leftrightarrow \mathscr{C}_0(\mathbf{X})(\mathbf{x})] \\ (\forall \mathbf{x})[X_1(\mathbf{x}) \leftrightarrow \mathscr{C}_1(\mathbf{X})(\mathbf{x})] \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \\ (\forall \mathbf{x}[X_{N-1}(\mathbf{x}) \leftrightarrow \mathscr{C}_{N-1}(\mathbf{X})(\mathbf{x})] \end{array}\right| \tag{3.3.1}$$

each $X_i$ being $n_i$-ary, say, $0 \leqslant i < N$.

We can regard the formulas $\mathscr{C}_i$, $0 \leqslant i < N$, as together determining a map $c$ on a certain boolean algebra, in this case a direct product of relation algebras on $D$

$$\Lambda = \Lambda_{n_0}(D) \times \Lambda_{n_1}(D) \times \ldots \times \Lambda_{n_{N-1}}(D).$$

The elements of $L$, the domain of $\Lambda$, are $N$-tuples

$$\langle \mathbf{A} \rangle = \langle A_0, A_1, \ldots A_{N-1} \rangle \text{ with } A_i \subseteq D^{n_i},\ 0 \leqslant i < N.$$

The operations on $\Lambda$ are defined coordinatewise, e.g.,

$$\langle \mathbf{A} \rangle \cup \langle \mathbf{B} \rangle = \langle A_0 \cup B_0, A_1 \cup B_1, \ldots, A_{N-1} \cup B_{N-1} \rangle$$
$$\langle \mathbf{A} \rangle \cap \langle \mathbf{B} \rangle = \langle A_0 \cap B_0, A_1 \cap B_1, \ldots, A_{N-1} \cap B_{N-1} \rangle \text{ etc.,}$$

and $\langle \mathbf{A} \rangle \subseteq \langle \mathbf{B} \rangle$ if and only if each $A_i \subseteq B_i$, $0 \leqslant i < N$.

It is well known that any direct product of complete boolean algebras is itself a complete boolean algebra, with e.g.,

$$\cap S = \langle \mathbf{A} \rangle \text{ where } A_i = \cap \{B_i | \langle B \rangle \in S \textit{ for some } \{B_j | j \neq i\}\}$$

for any $S \subseteq L$. (Actually $\Lambda$ is isomorphic to the algebra of subsets of a 'disjoint union' of $D^{n_i}$, $0 \leqslant i < N$, e.g., to $\Lambda_1(\{\langle i, a \rangle | 0 \leqslant i < N \textit{ and } a \in D^{n_i}\})$.

The map $c : L \to L$ determined by the $\mathscr{C}_i$ is then

$$c(A) = \langle c_1(\mathbf{A}), c_2(\mathbf{A}), \ldots c_{N-1}(\mathbf{A}) \rangle$$

where $c_i(\mathbf{A}) = \{\langle a_0, a_1 \ldots a_{n_i-1} \rangle | \langle a_0, a_1 \ldots a_{n_i-1}, A_0, A_1 \ldots A_{N-1} \rangle \textit{ satisfies } \mathscr{C}_i \text{ on } I\}$.

$c$ is monotone on $\Lambda$ if and only if, for all $\mathbf{A}$, $\mathbf{B}$

$$\langle \mathbf{A} \rangle \subseteq \langle \mathbf{B} \rangle \Rightarrow c(\langle \mathbf{A} \rangle) \subseteq c(\langle \mathbf{B} \rangle)$$

i.e., if and only if

$$A_j \subseteq B_j,\ 0 \leqslant j < N \Rightarrow c_i(\langle \mathbf{A} \rangle) \subseteq c_i(\langle \mathbf{B} \rangle),\ 0 \leqslant i < N.$$

But from the definition of $c_i$, and definition 3.1, this holds in every interpretation $I$ if and only if each formula $\mathscr{C}_i$ is monotone in each $X_j$, $0 \leqslant i < N$, $0 \leqslant j < N$.

It remains to translate into predicate calculus terms the lattice-theoretic definition of *Conv*. Since $Conv(c) \in L$, we can write

$$Conv(c) = \langle Conv_0(c), Conv_1(c), \ldots Conv_{N-1}(c) \rangle,$$

with $Conv_i(c) \subseteq D^{n_i}$. Since $Conv(c) = \cap \{\langle \mathbf{A} \rangle | c(\langle \mathbf{A} \rangle) \subseteq \langle \mathbf{A} \rangle\}$ we have

$$Conv_i(c) = \cap \{A_i | c(\langle \mathbf{A} \rangle) \subseteq \langle \mathbf{A} \rangle \textit{ for some } \{A_k | k \neq i\}\}$$
$$= \cap \{A_i | c_j(\langle \mathbf{A} \rangle) \subseteq A_j,\ 0 \leqslant j < N, \textit{for some } \{A_k | k \neq i\}\}$$

But then, writing '$\mathscr{C}$' for '$\mathscr{C}_0, \mathscr{C}_1, \ldots, \mathscr{C}_{N-1}$', if we define

$$Conv_i(\mathscr{C})(\mathbf{x}) \equiv (\forall \mathbf{X}) \left[ \bigwedge_{j=0}^{N-1} (\forall \mathbf{y}) [\mathscr{C}_j(\mathbf{X})(\mathbf{y}) \to X_j(\mathbf{y})] \to X_i(\mathbf{x}) \right] \tag{3.3.2}$$

then the extension of $Conv_i(\mathscr{C})$ in $I$ is just $Conv_i(c)$.
(Note that (3.3.2) reduces to (3.2.1) for $N = 1$).
The generalization of 3.3 is then the following:

**3.4.** *If each $\mathscr{C}_i$ is monotone in each $X_j$, $0 \leqslant i < N$, $0 \leqslant j < N$, then, for $0 \leqslant i < N$*

3.4.1. $\mathscr{C}_i(Conv_0(\mathscr{C}), Conv_1(\mathscr{C}), \ldots, Conv_{N-1}(\mathscr{C})) \equiv Conv_i(\mathscr{C})$

3.4.2. $Conv_i(\mathscr{C})(\mathbf{x}) \equiv (\forall \mathbf{X})\left[\bigwedge_{j=0}^{N-1}(\forall \mathbf{y})[\mathscr{C}_j(\mathbf{X})(\mathbf{y}) \leftrightarrow X_j(\mathbf{y})] \rightarrow X_i(\mathbf{x})\right]$.

In words, the formulas $Conv_i(\mathscr{C})$, $0 \leqslant i < N$ provide a general solution for $X_i$, $0 \leqslant i < N$, in the simultaneous recursion formulas (3.3.1), and this solution set is 'minimal' in the rather strong sense provided by 3.4.2.

We refer to formulas of the form $Conv(\mathscr{C})$, $Conv_i(\mathscr{C})$ as *convergence formulas.*

In addition to providing solutions to recursion formulas, convergence formulas can be used to abbreviate a number of important mathematical and logical properties of interpretations. For example, consider the following definitions of formulas $\mathscr{C}$ in the appropriate predicate calculi.

$$\mathscr{C}_A(X)(x) \equiv [x=0 \vee x(x-1)] \tag{3.4.1}$$

$$\mathscr{C}_{WO}(X)(x) \equiv (\forall y)[y<x \rightarrow X(y)] \tag{3.4.2}$$

$$\mathscr{C}_L(X) \equiv [\mathrm{atom}(x) \vee [X(\mathrm{car}(x)) \wedge X(\mathrm{cdr}(x))]]. \tag{3.4.3}$$

We can make the following remarks about these formulas:

1 In each case $\mathscr{C}(X)$ is monotone in $X$, by 3.2.

2 For each version of $\mathscr{C}$, the extension of $Conv(\mathscr{C})$ is some interesting subset of the domain of an interpretation:

(a) In any interpretation containing the integers, and in which '0', '$-$' receive their standard interpretation, the extension of $Conv(\mathscr{C}_A)$ is just the non-negative integers.

(b) In any interpretation in which '$<$' is a total ordering, the relation $Conv(\mathscr{C}_{WO})$ characterizes the set of elements in the maximal well-ordered initial segment of the ordering '$<$'. Consequently, $(\forall x)\ Conv(\mathscr{C}_{WO})(x)$ holds of the interpretation if and only if '$<$' is a well-ordering of the domain. (This can be seen by direct manipulation of the formula, obtaining

$$(\forall x)Conv(\mathscr{C}_{WO})(x) \equiv (\forall X)[(\exists x)X(x) \rightarrow (\exists z)[X(z) \wedge (\forall y)[X(y) \rightarrow \sim y<z]]]$$

which formalizes the conventional assertion of well-orderedness.) $\mathscr{C}_{WO}$, incidentally, determines a map $c$ on subsets of the domain which is monotone but not necessarily continuous in the sense of section 2 (the order-type of $\bigcup_{i=0}^{\infty} c^i(\phi)$ with respect to '$<$' is at most $\omega$, whereas $Conv(c)$ corresponds to arbitrarily large infinite ordinals for suitable interpretations, contra (2.1.1)).

(c) In the case of any interpretation satisfying the axioms for 'atom', 'car', 'cdr', 'cons' specified by McCarthy (1963), the extension of $Conv(\mathscr{C}_L)$ cuts out a subset which is, in effect (strictly, which is isomorphic to), a class of $S$-expressions over an alphabet which is the extension of the predicate 'atom'. Such an interpretation might be the class of all list structures over a fixed set of 'atoms', modulo the equivalence relation defined by the LISP function '*equal*'. Note that $Conv(\mathscr{C}_L)$ then fails for list structure containing infinite paths, e.g., with cycles of pointers.

3 In each case the closed formula $(\forall x)Conv(\mathscr{C})(x)$ can be regarded as the assertion of some (mathematical) induction principle:

(a) $(\forall x)Conv(\mathscr{C}_A)(x)$
$\equiv (\forall x)(\forall X)[(\forall y)[[y=0 \vee X(y-1)]\rightarrow X(y)]\rightarrow X(x)]$
$\equiv (\forall X)[[X(0)\wedge(\forall x)[[x\neq 0\wedge X(x-1)]\rightarrow X(x)]]\rightarrow(\forall x)X(x)]$

is a formulation of the standard induction axiom for arithmetic.

(b) $(\forall x)Conv(\mathscr{C}_{WO})(x)$
$\equiv (\forall X)[(\forall x)[(\forall y)[y<x\rightarrow X(y)]\rightarrow X(x)]\rightarrow(\forall x)X(x)]$

formalizes a principle of 'course-of values' induction, or of transfinite induction, depending on the interpretation of '$<$'.

(c) $(\forall x)Conv(\mathscr{C}_L)(x)\equiv(\forall X)[[(\forall x)[\mathrm{atom}(x)\rightarrow X(x)]$
$\wedge(\forall x)[[X(\mathrm{car}(x))\wedge X(\mathrm{cdr}(x))]\rightarrow X(x)]]$
$\rightarrow(\forall x)X(x)]$

formalizes a principle of 'structural induction' on LISP *S*-expressions (or on finite-path list structure); a special case of the principle used by Painter (1967) and Burstall (1969). One can deduce directly from the lattice-theoretic definition of *Conv* a principle which resembles a general form of mathematical induction. In its lattice-theoretic form this is (2.0.2).

**3.5.** (*Fixpoint Induction: Preliminary Form*)

3.5.1. *For any formulas* $\mathscr{F}$, $\mathscr{C}$, *whenever*

$$(\forall \mathbf{x})[\mathscr{C}(\mathscr{F})(\mathbf{x})\rightarrow\mathscr{F}(\mathbf{x})] \tag{3.5.1}$$

*is satisfied then so is*

$$(\forall \mathbf{x})[Conv(\mathscr{C})(\mathbf{x})\rightarrow\mathscr{F}(\mathbf{x})] \tag{3.5.2}$$

3.5.2* *For any formulas* $\mathscr{F}_i$, $\mathscr{C}_i$, $0\leqslant i<N$, *whenever*

$$(\forall \mathbf{x})[\mathscr{C}_i(\mathscr{F})(\mathbf{x})\rightarrow\mathscr{F}_i(\mathbf{x})] \tag{3.5.3}$$

*is satisfied, for each* $0\leqslant i<N$, *then so is each formula*

$$(\forall \mathbf{x})[Conv_i(\mathscr{C})(\mathbf{x})\rightarrow\mathscr{F}_i(\mathbf{x})] \tag{3.5.4}$$

*for* $0\leqslant i<N$.

3.5.1, 3.5.2 can of course be formulated as 'derived rules' of any standard axiomatic development of a second-order calculus, as can the rules stated below; the derivations are particularly simple in the above case, as should be clear on substituting for *Conv*, $Conv_i$ according to their definitions.

3.5.1 is closely related to principles of mathematical induction, in the sense that, with $\mathscr{C}\equiv\mathscr{C}_A$, say, (3.5.1) expresses precisely the standard premises for an argument by arithmetic induction. Formula (3.5.2) is then the appropriate conclusion, bearing in mind that in any appropriate interpretation the extension of $Conv(\mathscr{C}_A)$ is just the set of non-negative integers, as remarked above. One can look at the axiom of arithmetic induction $(\forall x)Conv(\mathscr{C}_A)(x)$, in this context, as restricting the domain of any interpretation to this set.

* Scott and de Bakker (private communication) formulated this independently as an axiom for a 'relational theory', with similar intentions to the author's. They also appreciated the connection of 3.5.2 with the methods of Floyd.

*Note.* If an 'induction axiom' $(\forall x)Conv(\mathscr{C})(x)$ holds, then the converse of 3.5.1 is clearly also true. However the converse of 3.5.1 does not hold in general. For a trivial example, take $\mathscr{C}(X)(x) \equiv X(f(x))$; then $Conv(\mathscr{C}) \equiv$ *false*, the identically false formula (as can be deduced by applying 3.5.1 with $\mathscr{F} \equiv$ *false*); but then, with $\mathscr{F} \equiv P(x_0)$, eq. 3.5.2 is valid, but not eq. 3.5.1, which is the formula $(\forall x_0)[P(f(x_0)) \rightarrow P(x_0)]$.

We will return to this point in section 6.

## 4. CONVERGENCE FORMULAS AND EQUATION SCHEMAS

It should be clear that, if the partial functions computed by equation schemas can be characterized by convergence formulas, then 3.5.1 and 3.5.2 embody useful principles for establishing their properties in terms of the characterization. In fact 3.5.2 for first-order formulas $\mathscr{C}_i$ turns out to embody a principle very similar to that used by Floyd on program schemas (Floyd 1967). (The extension of Floyd's method to equation schemas is already implicit in Manna and Pneuli (1969).) On the other hand, in the case that both $Conv(\mathscr{C})$, $\mathscr{F}$ characterize graphs of partial functions, 3.5.1 is another form of the 'recursion induction' principle devised by McCarthy (1963).

In this section, we show how to obtain, from an equation schema

$$\left.\begin{aligned} &\phi_0(x_1, x_2, \ldots, x_{n_0}) \Leftarrow \tau_0 \\ &\phi_1(x_1, x_2, \ldots, x_{n_1}) \Leftarrow \tau_1 \\ &\qquad\vdots \\ &\phi_{N-1}(x_1, x_2, \ldots, x_{n_{N-1}}) \Leftarrow \tau_{N-1} \end{aligned}\right\} \qquad (4.0.1)$$

first-order formulas $\mathscr{C}_0, \mathscr{C}_1, \ldots, \mathscr{C}_{N-1}$, monotone in each predicate variable $X_j$, such that the convergence formulas $Conv_i(\mathscr{C})$ characterize the partial functions computed by (4.0.1), in the following sense:

**4.1.** *Let $\phi_i^I$, $0 \leqslant i < N$, be the partial functions on $I$ corresponding to* (4.0.1); *then the $(n_i+1)$-extension of $Conv_i(\mathscr{C})$ on $I$ is the graph of $\phi_i^I$, viz.*

$$\{<x_0, x_1, \ldots, x_{n_i}> \mid \phi_i^I(x_1, x_2, \ldots, x_{n_i}) = x_0\}.$$

There is a close relationship between the formulas obtained in this way, and those obtained by following the method of Manna and Pnueli (1969). 4.1 will follow as a direct consequence of their results.

Corresponding to each subterm $\sigma$ of the right-hand side of eq. 4.0.1 we will obtain a formula $\mathscr{C}_\sigma$, by induction on the formation rules 1.1 for terms; the individual variable $x_0$ is, so to speak, to be the result of evaluating $\sigma$:

(1) If $\sigma$ is an individual variable or constant, then

$$\mathscr{C}_\sigma(x_0) \equiv [x_0 = \sigma] \qquad (4.1.1)$$

(2) If $\sigma$ is of the form

**if** $\sigma_1$ **then** $\sigma_2$ **else** $\sigma_3$

then

$$\mathscr{C}_\sigma(x_0) \equiv [[\mathscr{C}_{\sigma_1}(T) \wedge \mathscr{C}_{\sigma_2}(x_0)] \vee [\mathscr{C}_{\sigma_1}(F) \wedge \mathscr{C}_{\sigma_3}(x_0)]]. \qquad (4.1.2)$$

(3) If $\sigma$ has one of the forms

(a) $f(\sigma_1, \sigma_2, \ldots, \sigma_n)$, $f$ a basic function symbol.
(b) $P(\sigma_1, \sigma_2, \ldots, \sigma_n)$, $P$ a basic predicate symbol.
(c) $\phi_i(\sigma_1, \sigma_2, \ldots, \sigma_n)$, $0 \leqslant i < N$, $n = n_i$.

then

$$\mathscr{C}\sigma(x_0) \equiv (\exists y)\left[\bigwedge_{j=1}^{n} \mathscr{C}_{\sigma_j}(y_i) \wedge \mathscr{F}\right] \tag{4.1.3}$$

where $\mathscr{F}$ depends on which of (a), (b), (c) is the case, as follows:

(a) $\mathscr{F} \equiv [x_0 = f(y_1, y_2, \ldots y_n)]$ (4.1.4)

(b) $\mathscr{F} \equiv [[x_0 = T \wedge P(y_1, y_2, \ldots, y_n)] \vee [x_0 = F \wedge \sim P(y_1, y_2, \ldots y_n)]]$ (4.1.5)

(c) $\mathscr{F} \equiv X_i(x_0, y_1, y_2 \ldots y_n)$. (4.1.6)

*Example.* If $\sigma$ is the term

**if** $P(x_1)$ **then if** $Q(\phi_2(x_2))$ **then** $\phi_1(x_1, x_2)$ **else** $f(\phi_0(x_1, x_2))$
**else** $x_1$

then (after simplification)

$$\begin{aligned}\mathscr{C}_\sigma(x_0) \equiv [[P(x_1) \wedge [[(\exists y_1)[X_2(y_i, x_2) \wedge Q(y_1)] \wedge X_1(x_0, x_1, x_2)] \\ \vee [(\exists y_1)[X_2(y_1, x_2) \wedge \sim Q(y_1)] \\ \wedge (\exists y_1)[X_0(y_1, x_1, x_2) \wedge x_0 = f(y_1)]]]] \\ \vee [\sim P(x_1) \wedge x_0 = x_1]].\end{aligned}$$

(The simplification needed to reduce $\mathscr{C}_0$ to such a form involves just repeated substitutions according to the following equivalences, for arbitrary formulas $\mathscr{F}$, terms $\tau$:

$$(\exists x)[x = \tau \wedge \mathscr{F}(x)] \equiv \mathscr{F}(\tau); [\mathscr{F} \vee \textbf{true}] \equiv [\tau = \tau] \equiv \textbf{true};$$
$$[\mathscr{F} \wedge \textbf{false}] \equiv [T = F] \equiv [F = T] \equiv \textbf{false}; [\mathscr{F} \wedge \textbf{true}] \equiv [\mathscr{F} \vee \textbf{false}] \equiv \mathscr{F}.$$

Note that $[T \neq F]$ is assumed an axiom.)

The construction is completed by taking $\mathscr{C}_i \equiv \mathscr{C}_{\tau_i}$, $0 \leqslant i < N$.

We can now note the following points about the formulas $\mathscr{C}_i$ constructed:
(1) The formulas $\mathscr{C}_i$ are first order, since no step involves quantification over a predicate variable.
(2) Each $\mathscr{C}_i$ is monotone in each $X_j$, by 3.2, since the only negated subformulas are those introduced at step 3(b), and these involve no $X_j$.

Manna and Pnueli (1969) show how to obtain formulas $W_{\phi_i}(\mathbf{Q})$ closely related to the formulas $\mathscr{C}_i$ obtained above. In fact it can be shown, using standard predicate-calculus manipulations and the axiom $[T \neq F]$, that, for each $i$, $W_{\phi_i}(\mathbf{X}) \equiv \mathscr{C}_i(\mathbf{X})$, where

$$\mathscr{C}_i(\mathbf{X}) \equiv (\forall \mathbf{x})[\mathscr{C}_i(\mathbf{X})(\mathbf{x}) \rightarrow X_i(\mathbf{x})], \quad 0 \leqslant i < N$$

and the $\mathscr{C}_i$ are the formulas constructed above. One incidental advantage of the Manna–Pnueli formulas $W_{\phi_i}$ is that they avoid the use of equality (unless equalities are used in the terms $\tau_i$); also they arrive directly at a more compact form of $\mathscr{C}_i$ than that specified above.

4.1 follows from a suitable adaption of Lemmas 1, 2 of Manna and Pneuli (1969) generalized to the case $N \geqslant 1$, which we restate here as 4.2 (without proof):

**4.2.** (*Manna–Pnueli*) *Given an interpretation I, if*

$A_i = \{\langle x_0, x_1, \ldots, x_{n_i}\rangle \mid \phi_i^I(x_1, x_2, \ldots, x_{n_i}) = x_0\}$ *is the graph of* $\phi_i^I$, *then*:

4.2.1. $\langle A_0, A_1 \ldots A_{N-1}\rangle$ *satisfies* $\mathscr{C}_i$, $0 \leqslant i < N$.

4.2.2. *if* $\langle B_0, B_1 \ldots B_{N-1}\rangle$ *satisfies* $\mathscr{C}_i$ *for* $0 \leqslant i < N$ *then* $B_i \supseteq A_i$, $0 \leqslant i < N$.

Now let $C_i$ be the extension of $Conv_i(\mathscr{C})$, and note from the definition (3.3.2) of $Conv_i$

$$Conv_i(\mathscr{C})(\mathbf{x}) \equiv (\forall \mathbf{X})\left[\bigwedge_{j=1}^{N-1} \hat{\mathscr{C}}_j(\mathbf{X}) \rightarrow X_i(\mathbf{x})\right], \quad 0 \leqslant i < N.$$

From this form of $Conv_i$ and 4.2.1, it follows that $C_i \supseteq A_i$, $0 \leqslant i < N$. Conversely, from the Fixpoint Theorem 3.4.1, $\langle C_0, C_1, \ldots C_{N-1}\rangle$ satisfies $\mathscr{C}_i$; therefore from 4.2.2, $C_i \supseteq A_i$, $0 \leqslant i < N$, therefore $A_i = C_i$, $0 \leqslant i < N$. This, in outline, establishes 4.1.

To clarify the construction, and to illustrate the point that Fixpoint Induction is essentially a general version of the proof-method of Floyd (1967), consider the following simple example of an equation schema, over the appropriate formal basis:

$$\phi_0(x_1, x_2) \Leftarrow \textbf{if } x_1 \leqslant 0 \textbf{ then } 0 \textbf{ else } \phi_1(x_1, x_2, x_2)$$
$$\phi_1(x_1, x_2, x_3) \Leftarrow \textbf{if } x_3 \leqslant 0 \textbf{ then } \phi_0(x_1 - 1, x_2) \textbf{ else } \phi_1(x_1, x_2, x_3 - 1) + 1.$$

We assume the corresponding applied predicate calculus involves the usual axioms for the arithmetic operations and relations, say for arithmetic on the real numbers, and also the axioms for first-order arithmetic restricted to some predicate $N^+(x)$, intended to formalize the property of being a non-negative integer. The formulas corresponding to this schema can be taken as:

$$\mathscr{C}_0 \equiv [x_1 \leqslant 0 \wedge x_0 = 0] \vee [x_1 > 0 \wedge X_1(x_0, x_1, x_2, x_2)] \tag{4.2.1}$$

$$\mathscr{C}_1 \equiv [x_3 \leqslant 0 \wedge X_0(x_0, x_1 - 1, x_2)] \vee [x_3 > 0 \wedge (\exists y)[X_1(y, x_1, x_2, x_3 - 1) \wedge x_0 = y + 1]]$$
$$\equiv [x_3 \leqslant 0 \wedge X_0(x_0, x_1 - 1, x_2)] \vee [x_3 > 0 \wedge X_1(x_0 - 1, x_1, x_2, x_3 - 1)]. \tag{4.2.2}$$

We want to show that $\phi_0(x_1, x_2)$ over the non-negative integers, if convergent, computes the product function $(x_1 * x_2)$. Define $\mathscr{F}_0$, $\mathscr{F}_1$, by

$$\mathscr{F}_0 \equiv [N^+(x_1) \wedge N^+(x_2)] \rightarrow x_0 = x_1 * x_2 \tag{4.2.3}$$

$$\mathscr{F}_1 \equiv [N^+(x_1) \wedge N^+(x_2) \wedge N^+(x_3) \wedge x_1 > 0] \rightarrow x_0 = ((x_1 - 1) * x_2) + x_3 \tag{4.2.4}$$

A formalization of the required conclusion, by 4.1, is

$$(\forall \mathbf{x})[Conv_0(\mathscr{C}_0, \mathscr{C}_1)(\mathbf{x}) \rightarrow \mathscr{F}_0(\mathbf{x})].$$

From the Fixpoint Induction principle 3.5.2, it is sufficient to prove the two formulas

$$(\forall \mathbf{x})[\mathscr{C}_0(\mathscr{F}_0, \mathscr{F}_1)(\mathbf{x}) \rightarrow \mathscr{F}_0(\mathbf{x})]$$
$$(\forall \mathbf{x})[\mathscr{C}_1(\mathscr{F}_0, \mathscr{F}_1)(\mathbf{x}) \rightarrow \mathscr{F}_1(\mathbf{x})].$$

Expanding these formulas by substitution for eqs 4.2.1–4.2.4, it is sufficient to prove the following four formulas:

$(\forall \mathbf{x})[[x_1 \leqslant 0 \wedge x_0 = 0] \rightarrow [[N^+(x_1) \wedge N^+(x_2)] \rightarrow x_0 = x_1 * x_2]]$

$(\forall \mathbf{x})[[x_1 > 0 \wedge [[N^+(x_1) \wedge N^+(x_2)] \rightarrow x_0 = ((x_1 - 1) * x_2) + x_2]] \rightarrow [[N^+(x_1) \wedge N^+(x_2)] \rightarrow x_0 = x_1 * x_2]]$

$(\forall \mathbf{x})[[x_3 \leqslant 0 \wedge [[N^+(x_1 - 1) \wedge N^+(x_2)] \rightarrow x_0 = (x_1 - 1) * x_2]] \rightarrow \rightarrow [[N^+(x_1) \wedge N^+(x_2) \wedge N^+(x_3) \wedge x_1 > 0] \rightarrow x_0 = ((x_1 - 1) * x_2) + x_3]]$

$(\forall \mathbf{x})[[x_3 > 0 \wedge [[N^+(x_1) \wedge N^+(x_2) \wedge N^+(x_3 - 1) \wedge x_1 > 0] \rightarrow (x_0 - 1) = ((x_1 - 1) * x_2) + (x_3 - 1)]] \rightarrow [[N^+(x_1) \wedge N^+(x_2) \wedge N^+(x_3) \wedge x_1 > 0] \rightarrow x_0 = ((x_1 - 1) * x_2) + x_3]]$.

These can all be proved straightforwardly from the standard axioms assumed for the calculus. Formal proofs of these would then establish the property claimed for $\phi_0$.

Before exhibiting the relationship of Fixpoint Induction to recursion induction, we first point out the general relationship between the value of a term $\sigma$ in a particular 'environment' and satisfaction of the formula $\mathscr{C}_\sigma$ corresponding to $\sigma$.

Suppose $\sigma$ involves at most the derived function symbols $\phi_0, \phi_1 \ldots \phi_{N-1}$ and the variables $x_1, x_2, \ldots, x_n$. Given an interpretation $I$, consider the value of $\sigma$ in an 'environment' in which $\phi_i$ takes the value $\alpha_i$, $0 \leqslant i < N$, and $x_j$ takes the value $a_j$, $1 \leqslant j < N$, $\alpha_i$ being a partial function over the domain of $I$.

Following the standard evaluation rules for terms, denote the value (if any) of $\sigma$ in this environment by $\sigma(\boldsymbol{\alpha}, \mathbf{a})$. The following lemma can be proved by induction on the formation rules 1.1 for terms:

**4.3.** *Let $A_i$ be the graph of $\alpha_i$, then*

$$\langle a_0, a_1, \ldots, a_n, A_0, A_1, \ldots, A_{N-1} \rangle \textit{ satisfies } \sigma \textit{ if and only if } \sigma(\boldsymbol{\alpha}, \mathbf{a}) = a_0.$$

It follows that any equation

$$\sigma(\boldsymbol{\alpha}, \mathbf{a}) = \tau(\boldsymbol{\alpha}, \mathbf{a})$$

holds for all $\mathbf{a}$ 'strongly' (i.e., whenever either side is defined) if and only if

$$(\forall \mathbf{x})[\mathscr{C}_\sigma(\mathbf{X})(\mathbf{x}) \leftrightarrow \mathscr{C}_\tau(\mathbf{X})(\mathbf{x})]$$

is satisfied by $\langle A_0, A_1, \ldots, A_{N-1} \rangle$.

We can now indicate the relationship of the Fixpoint Induction principle 3.5.1 to recursion induction. Suppose $\mathscr{F}$, $\mathscr{G}$ to be (first or second order) formulas whose extensions in $I$ are graphs of partial functions $\alpha^I$, $\beta^I$, and that $\alpha^I$, $\beta^I$ satisfy strongly the recursion equation

$$\phi(x_1, x_2 \ldots x_n) = \tau \tag{4.3.1}$$

on $I$. The conclusion of the recursion induction rule as stated by McCarthy (1963) is that $\alpha^I$, $\beta^I$ are equivalent on the domain of $\phi^I$. We must show that this can be seen to follow also from 3.5.1.

From the remark following 4.3, since $\alpha^I$ satisfies the equation, the formula

$$(\forall \mathbf{x})[\mathscr{C}_\tau(\mathscr{F})(\mathbf{x}) \leftrightarrow \mathscr{C}_{\phi(x_1, x_2, \cdots x_n)}(\mathscr{F})(\mathbf{x})]$$

is satisfied in $I$. Therefore, substituting for the equivalence $\mathscr{C}_{\phi(x_1, x_2, \ldots, x_n)} \equiv X_0(x_0, x_1, \ldots, x_n)$ and weakening the statement,

$$(\forall \mathbf{x})[\mathscr{C}_\tau(\mathscr{F})(\mathbf{x}) \rightarrow \mathscr{F}(\mathbf{x})]$$

is satisfied. But then from 3.5.1, so is

$$(\forall \mathbf{x})[Conv\,(\mathscr{C}_\tau)(\mathbf{x}) \rightarrow \mathscr{F}(\mathbf{x})]$$

and similarly, so is

$$(\forall \mathbf{x})[Conv(\mathscr{C}_\tau)(\mathbf{x}) \rightarrow \mathscr{G}(\mathbf{x})].$$

But these formulas express the property that $\alpha^I$, $\beta^I$, extend $\phi^I$, on all $I$, i.e., that $\alpha^I$, $\beta^I$ are equivalent on the domain of $\phi^I$, which was the required conclusion.

It is worth pointing out that the same reasoning establishes a slight strengthening of the recursion induction rule, viz. that $\alpha^I$, $\beta^I$ need only satisfy eq. 4.3.1 for those values of variables for which $\tau$ is defined.

## 5. ELIMINATION OF SECOND-ORDER QUANTIFIERS

Many interesting properties of particular equation schemas can clearly be expressed in terms of the convergence formulas constructed for them by the method of Section 4. In particular the 'strong equivalence' relation between equation schemas is now representable straightforwardly as the validity of a closed formula of the form

$$(\forall \mathbf{x})[Conv_i(\mathscr{C})(\mathbf{x}) \leftrightarrow Conv_j(\mathscr{D})(\mathbf{x})]. \tag{5.0.1}$$

On the other hand the wide variety of properties considered by Manna and Pnueli (1969) are also expressible in terms of the satisfaction of various closed formulas constructed from convergence formulas.

In this section we consider some rules for simplifying formulas involving convergence formulas by permitting various manipulations involving the second-order quantifiers introduced by convergence formulas.

This turns out to be one way of looking at the reductions to first-order validity problems obtained by Manna and Pnueli (1969).

Manna and Pnueli consider only validity and satisfiability properties relative to a fixed interpretation of the formal basis. This has the effect that, while their validity results generalize straightforwardly to validity results over the class of interpretations satisfying the axioms of the calculus, those results which represent properties of schemas in terms of 'satisfiability' properties of formulas do not generalize to 'satisfiability' results in the sense of section 1 above. (The property in question, generalized over all permissible interpretations of the basis, is *not* established by showing that the negation of the formula is not valid, but rather by proving as a theorem the second-order existential closure of this formula.)

The following result is a direct corollary of the Fixpoint Theorem 3.4.1.

**5.1.** *If each* $\mathscr{C}_i(\mathbf{X})$ *is monotone in each* $X_j$, $0 \leqslant i < N$, $0 \leqslant j < N$, *then whenever*

$$(\forall \mathbf{X})\left[\bigwedge_{i=0}^{N-1}[(\forall \mathbf{x})[\mathscr{C}_i(\mathbf{X}) \rightarrow X_i(\mathbf{x})] \rightarrow \mathscr{F}(\mathbf{X})\right] \tag{5.1.1}$$

*is satisfied, then so is*

$$\mathscr{F}(Conv_0(\mathscr{C}), Conv_1(\mathscr{C}), \ldots, Conv_{N-1}(\mathscr{C})) \tag{5.1.2}$$

5.2 below covers an important variety of cases in which the converse of 5.1 holds; the equivalence (5.2.1) ≡ (5.2.2) can be seen as the principal source of the Manna–Pnueli reductions to first-order validity problems.

**5.2.** *If both* $\mathscr{F}(\mathbf{X})$, $\mathscr{C}_i(\mathbf{X})$ *are monotone in each* $X_j$, $0 \leqslant i < N$, $0 \leqslant j < N$, *then*

$$\mathscr{F}(Conv_0(\mathscr{C}), Conv_1(\mathscr{C}), \ldots, Conv_{N-1}(\mathscr{C})) \tag{5.2.1}$$

$$\equiv (\forall \mathbf{X})\left[\bigwedge_{i=0}^{N-1} (\forall \mathbf{x})[\mathscr{C}_i(\mathbf{X})(\mathbf{x}) \rightarrow X_i(\mathbf{x})] \rightarrow \mathscr{F}(\mathbf{X})\right] \tag{5.2.2}$$

$$\equiv (\forall \mathbf{X})\left[\bigwedge_{i=0}^{N-1} (\forall \mathbf{x})[\mathscr{C}_i(\mathbf{X})(\mathbf{x}) \leftrightarrow X_i(\mathbf{x})] \rightarrow \mathscr{F}(\mathbf{X})\right] \tag{5.2.3}$$

*Proof.* (a) If (5.2.2) is satisfied, so is (5.2.3), trivially.
(b) If (5.2.3) is satisfied, so is (5.2.1), from 5.1.
(c) From 3.5.2, with $\mathscr{F}_i \equiv X_i$, the formula

$$\bigwedge_{i=0}^{N-1} (\forall \mathbf{x})[\mathscr{C}_i(\mathbf{X})(\mathbf{x}) \rightarrow X_i(\mathbf{x})] \rightarrow \bigwedge_{i=0}^{N-1} (\forall \mathbf{x})[Conv_i(\mathscr{C})(\mathbf{x}) \rightarrow X_i(\mathbf{x})]$$

is valid, and from monotonicity of $\mathscr{F}$

$$\bigwedge_{i=0}^{N-1} (\forall \mathbf{x})[Conv_i(\mathscr{C})(\mathbf{x}) \rightarrow X_i(\mathbf{x})] \rightarrow [\mathscr{F}(Conv_0(\mathscr{C}), Conv_1(\mathscr{C}), \ldots Conv_{N-1}(\mathscr{C}) \rightarrow \mathscr{F}(\mathbf{X})]$$

is valid also. Formula (5.2.2) follows from these two formulas and (5.2.1).

5.2 provides, in effect, a method of extracting the initial universal quantifiers from positive occurrences of convergence formulas, which conserves validity properties.

The first-order formula constructed by the Manna–Pnueli method which is valid if and only if '$\phi^I$ is correct with respect to $\mathscr{E}$, $\mathscr{G}$ *on all* $I$' is equivalent to eq. 5.2.2, omitting the second-order quantifiers, with $\mathscr{C}_i$ as constructed in section 4, and with

$$\mathscr{F} \equiv (\forall \mathbf{x})[\mathscr{E}(\mathbf{x}) \rightarrow (\exists x_0)[X_i(\mathbf{x}) \wedge \mathscr{G}(\mathbf{x})]].$$

Note that $\mathscr{F}$ is monotone in $X_i$, so that 5.2 is indeed applicable. The formula constructed is therefore valid if and only if

$$(\forall \mathbf{x})[\mathscr{E}(\mathbf{x}) \rightarrow (\exists x_0)[Conv_i(\mathscr{C})(\mathbf{x}) \wedge \mathscr{G}(\mathbf{x})]]$$

is valid, and this assertion expresses, by 4.1, just the property claimed for $\phi_i$. The form (5.2.3) provides a stronger form of the Manna–Pnueli formula.

## 6. STRONGER FIXPOINT INDUCTION

In conclusion, we give the predicate calculus version of the lattice-theory theorem 2.2, as an example of a very general rule which suggests itself through the rather abstract approach taken here:

**6.1 Fixpoint induction: stronger form**

If each $\mathscr{C}_i$ is monotone in each $X_j$, $0 \leqslant i < N$, $0 \leqslant j < N$, then for any $\mathscr{F}$,

$$\bigwedge_{i=0}^{N-1} (\forall \mathbf{x})[Conv_i(\mathscr{C})(\mathbf{x}) \rightarrow [\mathscr{C}_i(\mathscr{F})(\mathbf{x}) \rightarrow \mathscr{F}_i(\mathbf{x})]] \tag{6.1.1}$$

$$\equiv \bigwedge_{i=0}^{N-1} (\forall \mathbf{x})[Conv_i(\mathscr{C})(\mathbf{x}) \rightarrow \mathscr{F}_i(\mathbf{x})] \tag{6.1.2}$$

As a deduction rule, the intention is to deduce eq. 6.1.2 from eq. 6.1.1.

In addition to incorporating the preliminary form 3.5, 6.1 permits various 'nested' induction arguments in the process of establishing the formulas $[\mathscr{C}_i(\mathscr{F})(\mathbf{x}) \rightarrow \mathscr{F}_i(\mathbf{x})]$.

Such arguments turn out to be particularly useful in establishing strong equivalence properties between schemas. In particular, the equivalence theorems in Cooper can be derived using 6.1, and with rather less 'guesswork' than is needed for Cooper's own proofs. (His 'difficult proof' (Cooper 1969, p. 14) can be established this way with no particular difficulty.) Further examples, however, make us doubt whether such proofs can be mechanized in practice without some powerful heuristics for use with 6.1, and these remain to be formulated.

**Acknowledgements**

My interest in this area was originally aroused by the work David Cooper describes in *Machine Intelligence 4*. I am grateful both to him and to Robin Milner for many stimulating discussions in this area. Robin Milner showed me the correct proof of 2.2. The work described was supported by a grant from the Science Research Council.

## REFERENCES

Ashcroft, E.A. (1969) Functional programs as axiomatic theories. *Centre for Computing & Automation Report No. 9*, Imperial College, London.

Burstall, R.M. (1969) Proving properties of programs by structural induction. *Computer Journal*, **12**, 41–8.

Church, A. (1956) *Introduction to Mathematical Logic*. Princeton University Press.

Cooper, D.C. (1969) Program scheme equivalence and second-order logic. *Machine Intelligence 4*, pp. 3–15 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Floyd, R.W. (1967) Assigning meanings to programs. *AMS Applied Mathematics Symposia* **19**, 19–32.

Luckham, D.C., Park, D.M.R. & Paterson, M.S. (1967) *On formalized computer programs*. Programming Research Group, Oxford University.

Manna, Z. (1969) Properties of programs and the first-order predicate calculus. *J. Ass. comput. Mach.*, **16**, 244–55.

Manna, Z. & Pnueli, A. (1969) Formalization of properties of recursively defined functions. *ACM Symposium on Theory of Computing*, 201–10.

McCarthy, J. (1963) A Basis for a Mathematical Theory of Computation. *Computer Programming and Formal Systems* (eds Braffort, P. & Hirschberg, D.). Amsterdam: North Holland.

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.

Paterson, M.S. (1968) Program schemata. *Machine Intelligence 3*, pp. 19–31 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Rogers, H. (1967) *Theory of Recursive Functions and Effective Computability*. New York: McGraw-Hill.

Tarski, A. (1955) A lattice-theoretical fixpoint theorem and its applications. *Pacific J. of Maths*, **5**, 285–309.

4

# Formal Description of Program Structure and Semantics in First Order Logic

R. M. Burstall
Department of Machine Intelligence and Perception
University of Edinburgh

## 1. INTRODUCTION

This paper shows how the semantics of a programming language can be defined by a set of sentences of first order logic. A set of sentences describing a major part of ALGOL 60 is exhibited. A method is given for describing a program, e.g., an ALGOL program, by another set of first order sentences. When these sentences are set beside the sentences defining the language it is possible to infer, using the ordinary rules of logic, what the outcome of the program will be for any initial conditions. Thus one can prove correctness and termination.

The main idea can be grasped by reading the first three sections of this paper only. The remainder works out the details for most of the main features of ALGOL 60 to demonstrate the feasibility of applying the technique to a real programming language. The only background assumed is some knowledge of ALGOL 60 and of the predicate calculus.

Recent work by McCarthy and Painter (1967), Painter (1967), Floyd (1967), Hoare (1969), Manna (1969), Manna and McCarthy (1970), Green (1969), Allen, Ashcroft and Florentin (1969), and others has enabled one to prove that a program in some simplified ALGOL-like language terminates, and is correct i.e., that for given input conditions a specified output condition is obtained. Ashcroft (1970) has used these methods to cover most of ALGOL 60.

Manna's method consists of a set of rules which, given a program and a specified input condition, produce a sentence of first order functional calculus which is unsatisfiable if and only if execution of the program terminates and conforms to a specified output condition. These rules are given in mathematical/computing terminology and describe how the sentence of first order calculus is to be constructed by examining the various parts of the program. Floyd (1967) pointed out that such rules embody the semantics of the programming language concerned.

The technique put forward here is to use a very simple set of rules to obtain a set of sentences of first order calculus which just describe the structure of any given program, i.e., they say what statements and expressions the program contains and how these statements and expressions are connected together. Another set of sentences of first order calculus defines the semantics of the programming language in use. From these two sets of sentences, one describing the particular program being considered and one describing the programming language semantics, the ordinary rules of logic can be used to infer further sentences which describe the execution of the program and hence its outcome. These latter sentences play the same role as Manna's sentence of first order calculus, and when a suitable input condition is adjoined to them the termination and correctness of the program may be proved.

Thus the preliminary part of the business of proving the correctness of a program, namely obtaining a description of its execution, has here been almost completely formalized itself (excluding only a stage corresponding to a slightly elaborated syntax analysis). A principal effect of this is to enable one to state the semantics of a programming language in first order functional calculus. It is hoped that this work will make it easier to extend correctness proof techniques to more elaborate languages and that it will provide a useful tool for formal language definition along the lines suggested by Strachey (1965), and workers at IBM Vienna such as Lauer (1968).

The method of describing program execution used here is based on that of C. Green (1969), itself derived from McCarthy's work on 'situations, actions and causal laws' (1963). I have used the idea of specifying the execution of a program by a set of pairs each consisting of a point in the program and a state of the store. The importance of this notion was suggested to me by Landin (1970). The technique used for assignment is essentially that of Burstall (1968), reduced to first order logic.

The method is demonstrated by applying it to a subset of ALGOL 60 which includes simple kinds of blocks, arrays, recursive procedures and side-effects. The language is built up in stages each introducing new features. We add a final stage introducing list processing. We impose a number of inessential restrictions, such as procedures having only one parameter, to keep the exposition as brief as possible.

## 2. NOTATION

We use the usual notation for first order functional calculus, with the following slight extensions for readability.

(i) Variables not bound by a quantifier are assumed to be universally quantified.

(ii) We often write two-place predicates as infixes, thus we write $x\ R\ y$ for $R(x, y)$ and $x\ not\ R\ y$ for $not(R(x, y))$.

(iii) Similarly for the usual binary functions of mathematics, thus $x=y$ for $equal(x, y)$ and $x+y$ for $sum(x, y)$.

(iv) Parentheses are implied in the usual way e.g., we write *p and q⇒r and not s* for *(p and q)⇒(r and (not s))*.

(v) When a set of sentences are given they are separated by full stops.

We do not use a many-sorted logic, but for the reader's guidance we specify the sorts of the objects, predicates and functions concerned.

$p \subseteq s_1 \times \ldots \times s_n$ means that $p$ is an $n$-place relation with arguments of sorts $s_1$ to $s_n$.

$f: s_1 \times \ldots \times s_n \rightarrow s_{n+1}$ means that $f$ is an $n$-place function with arguments of sorts $s_1$ to $s_n$, and result of sort $s_{n+1}$.

Where several predicates or several functions have the same sort we write $p, q, r \subseteq \ldots$ or $f, g, h: \ldots$.

## 3. ELEMENTARY ALGOL: EXPRESSIONS, ASSIGNMENTS, CONDITIONAL STATEMENTS AND GOTO

### 3.1 Structure of elementary programs

The first step is to take the given program and derive a set of first order sentences to describe its structure. A numbered point is associated with the start and finish of each statement of the program and these points are related to each other by predicates. Expressions and statements are converted into terms and these terms are related to each other by predicates. The process should be quite easy to mechanize with the aid of a context-free syntax analyser. It is specified informally by giving examples. A formal method is suggested in the appendix. The sorts, predicates, functions, and constants needed are as follows.

Sorts. *Points*, *statements*, *labels*, *expressions*, *identifiers* and *numerals*. *Identifiers* and *numerals* are subsorts of *expressions*. (*Points* are marked positions in programs before and after statements.)

Predicates, functions, and constants

*identifier* ⊆ *expressions*
*numeral* ⊆ *expressions*
*has* ⊆ *points* × *statements*
*follows* ⊆ *points* × *points*
*labels* ⊆ *labels* × *points*
*assignment: identifiers* × *expressions* → *statements*
*ifthen: expressions* × *points* → *statements*
*goto: labels* → *statements*
*plus, minus, times, equal: expressions* × *expressions* → *expressions*
**j,k,l**, etc. ∈ *identifiers*.

(We use bold type for identifiers to avoid confusion with logical variables. They are logical constants. Please note the distinction.)

*zero, one, two* etc. ∈ *numerals*.

(We could obviously construct identifiers and numerals in terms of letters,

digits and, say, concatenation, and so avoid using an infinite supply of constants. We eschew this technicality.)

Axioms

*identifier*(**j**). *identifier*(**k**). etc. **j**≠**k**. **k**≠**l**. **j**≠**l**. etc.

*numeral*(*zero*). *numeral*(*one*). etc.

### 3.2. Example of structural description of an elementary program

The program computes $2^n$. The program is given on the left with the points numbered (1), (2), .... The corresponding set of sentences of first order functional calculus is given on the right. This example is introduced as an informal specification of the rules for obtaining the descriptive sentences. Later we sketch a proof of correctness of this program.

| | |
|---|---|
| (*1*) *j*:=*0*; | *p1 has assignment*(**j**, *zero*). |
| (*2*) *k*:=*1*; | *p2 follows p1. p2 has assignment*(**k**, *one*). |
| *loop:* (*3*) **if** *j*=*n* **then** | **loop** *labels p3. p3 follows p2.* |
| | *p3 has if then*(*equal*(**j**, **n**), *p4*). |
| (*4*) **goto** *out*; | *p4 has goto*(**out**). |
| (*5*) | *p5 follows p4.* |
| (*6*) *k*:=*2*×*k*; | *p6 follows p3. p6 has assignment*(**k**, *times*(*two*, **k**)). |
| (*7*) *j*:=*j*+*1*; | *p7 follows p6. p7 has assignment*(**j**, *plus*(**j**, *one*)). |
| (*8*) **goto** *loop*; | *p8 follows p7. p8 has goto*(**loop**). |
| *out:* (*9*) | **out** *labels p9. p9 follows p8.* |

*Notes*

(a) We insert numbered points so that each statement has a point before it and one after it. The points go where ALGOL labels could go, but also, for example, after the statement following a **then**, e.g., (5) above.

(b) We say a point *follows* another if it comes before the next statement in the sequence, ignoring inner statements depending on a **then**, e.g., (6) follows (3) above.

### 3.3 Semantics of elementary programs

We now express the semantics of the elementary subset of ALGOL which we have used as a set of first order sentences. We need the ideas of a value, here an integer or truth value, and a state, here just the values of the variables. Each state is associated with a certain point in the program.

Sorts: *values*, *states*.

Predicates, functions and constants

*at*⊆*states*×*points*

*next*: *states*→*states*

*val*: *expressions*×*states*→*values*

*jumpsto*⊆*states*×*points*

*numeralval*: *numerals*→*values*

+, −, ×: *values*×*values*→*values*

*true*, *false*, *0*, *1*, *2*, etc. ∈ *values*

Axioms: We assume the usual axioms of arithmetic and feel free to quote elementary theorems of arithmetic.

Axioms about semantics

Expressions

$$numeral(m) \Rightarrow val(m, s) = numeralval(m)$$
$$numeralval(zero) = 0.\ numeralval(one) = 1, \text{ etc.}$$
$$val(plus(e, e'), s) = val(e, s) + val(e', s)$$
$$val(minus(e, e'), s) = val(e, s) - val(e', s)$$
$$val(times(e, e'), s) = val(e, s) \times val(e', s)$$
$$val(e, s) = val(e', s) \Rightarrow val(equal(e, e'), s) = true$$
$$val(e, s) \neq val(e', s) \Rightarrow val(equal(e, e'), s) = false$$

Assignment

$$p\ has\ assignment(i, e)\ and\ p'\ follows\ p\ and\ s\ at\ p \Rightarrow next(s)\ at\ p'$$
$$and\ val(i, next(s)) = val(e, s)$$
$$and[identifier(i')\ and\ i' \neq i \Rightarrow val(i', next(s)) = val(i', s)]$$

Definition of jumpsto

$$s\ jumpsto\ p \Rightarrow next(s)\ at\ p$$
$$and[identifier(i) \Rightarrow val(i, next(s)) = val(i, s)]$$

Goto

$$p\ has\ goto(l)\ and\ l\ labels\ p'\ and\ s\ at\ p \Rightarrow s\ jumpsto\ p'$$

If . . . then

$$p\ has\ ifthen(e, p')\ and\ p''\ follows\ p\ and\ s\ at\ p \Rightarrow$$
$$[val(e, s) = true \Rightarrow s\ jumpsto\ p']$$
$$and\ [val(e, s) = false \Rightarrow s\ jumpsto\ p'']$$

$$p\ has\ ifthen(e, p')\ and\ p''\ follows\ p\ and\ p'''\ follows\ p'\ and\ s'''\ at\ p''' \Rightarrow s'''\ jumpsto\ p''$$

### 3.4 Proving correctness and termination of an elementary program

To illustrate the technique for correctness proofs we now prove that the program given above does indeed compute $2^n$.

We use two abbreviations:

(a) $s^*$ for next($s$), $s^{**}$ for next(next($s$)) etc.

(b) $\mathbf{i}_s$ for val($\mathbf{i}$, $s$)

*Initial conditions*

$$s_1\ at\ p_1 \text{ and } \mathbf{n}_{s_1} = v$$

*Theorem*

$$(\exists s)[s\ at\ p9\ and\ k_s = 2^v]$$

*Proof.* We use the structural sentences describing each step in the program together with the appropriate semantic sentences to obtain a proof outlined thus:

(*1*) $s_1^*\ at\ p2.\ \mathbf{j}_{s_1^*} = 0.\ \mathbf{n}_{s_1^*} = v.$

(using the sentences associated with the first statement together with the axiom for assignment).

(2) $s_1^{**}$ *at p3.* $\mathbf{k}_{s_1^{**}}=1.\ \mathbf{j}_{s_1^{**}}=0.\ \mathbf{n}_{s_1^{**}}=\nu.$

(3) Define $H$ an induction hypothesis for the loop

$$H(\alpha)\Leftrightarrow[\alpha\leqslant\nu\Rightarrow(\exists s)[s \textit{ at p3 and } \mathbf{j}_s=\alpha \textit{ and } \mathbf{k}_s=2^\alpha \textit{ and } \mathbf{n}_s=\nu]]$$

(i) $H(0)$ follows from (2) putting $s=s_1^{**}$ and using $2^0=1$.

(ii) Assume $H(\alpha)$ and prove $H(\alpha+1)$.

If $\alpha\geqslant\nu$ then $\alpha+1>\nu$ and $H(\alpha+1)$ is trivially true.

If $\alpha<\nu$ then by $H(\alpha)$ there is an $s$ such that

$$s \textit{ at p3. } \mathbf{j}_s=\alpha.\ \mathbf{k}_s=2^\alpha.\ \mathbf{n}_s=\nu.$$

$$val(equal(\mathbf{j},\mathbf{n}),s)=false. \tag{3.1}$$

$$s^* \textit{ at p6. } \mathbf{j}_{s^*}=\alpha.\ \mathbf{k}_{s^*}=2^\alpha.\ \mathbf{n}_{s^*}=\nu. \tag{3.2}$$

$$s^{**} \textit{ at p7. } \mathbf{j}_{s^{**}}=\alpha.\ \mathbf{k}_{s^{**}}=2\times2^\alpha.\ \mathbf{n}_{s^{**}}=\nu. \tag{3.3}$$

$$s^{***} \textit{ at p8. } \mathbf{j}_{s^{***}}=\alpha+1.\ \mathbf{k}_{s^{***}}=2\times2^\alpha.\ \mathbf{n}_{s^{***}}=\nu. \tag{3.4}$$

$$s^{****} \textit{ at p3. } \mathbf{j}_{s^{****}}=\alpha+1.\ \mathbf{k}_{s^{****}}=2\times2^\alpha.\ \mathbf{n}_{s^{****}}=\nu. \tag{3.5}$$

Using $2\times2^\alpha=2^{\alpha+1}$ this gives $H(\alpha+1)$.

Thus $H(\alpha)$ holds for all $\alpha$ by mathematical induction on $H$.

(4) Put $\alpha=\nu$ getting $H(\nu)$. Then there is some $s_2$ such that

$$s_2 \textit{ at p3. } \mathbf{j}_{s_2}=\nu.\ \mathbf{k}_{s_2}=2^\nu.\ \mathbf{n}_{s_2}=\nu.$$

$$val(equal(\mathbf{j},\mathbf{n}),s_2)=true.$$

(5) $s_2^*$ *at p4.* $\mathbf{k}_{s_2^*}=2^\nu.$

(6) $s_2^{**}$ *at p9.* $\mathbf{k}_{s_2^{**}}=2^\nu.$ (end of proof)

The proof is tedious but straightforward. The induction hypothesis must be supplied by a little human ingenuity. To obtain a proof using a first order theorem-prover the instance of the mathematical induction schema corresponding to this induction hypothesis would have to be supplied.

## 4. BLOCKS WITH INTEGER DECLARATIONS

We extend the language to allow blocks with integer declarations. The rules for describing structure are extended. The semantic sentences are extended and some of them are replaced by more elaborate versions.

### 4.1 Structure of programs with blocks

Predicates

*declaredat* $\subseteq$ *identifiers* $\times$ *points*

*begins* $\subseteq$ *points* $\times$ *points*

*ends* $\subseteq$ *points* $\times$ *points.*

### 4.2 Example of a program with blocks

The following program does no useful computation. It is chosen to illustrate the scope rules.

| | |
|---|---|
| (1) **begin integer** *j, k;* | *i declaredat p1*⇔*i*=**j** *or i*=**k**. |
| (2) *j*: =*0;* | *p2 begins p1. p2 has assignment*(**j**, *zero*). |
| (3) *k*: =*1;* | *p3 follows p2. p3 has assignment*(**k**, *one*). |
| (4) **begin integer** *k;* | *p4* follows *p3. i declaredat p4*⇔*i*=**k**. |
| (5) *k*: =*2;* | *p5 begins p4. p5 has assignment*(**k**, *two*). |
| (6) *j*: =*j*+*k;* | *p6 follows p5. p6 has assignment*(**j**, *plus*(**j**, **k**)). |
| (7) | *p7 follows p6. p7 ends p4.* |
| **end;** | |
| (8) | *p8 follows p4. p8 ends p1.* |
| **end** | |
| (9) | *p9 follows p1.* |

#### 4.3 Semantics of programs with blocks

We introduced two new sorts, blocks and unique names. Blocks are partially ordered by the relation '*inside*'. A unique name stands for an identifier together with the block in which it is declared. The scope rules demand that each identifier be associated with the appropriate unique name. An assignment now assigns a value to a unique name, rather than to an identifier. We call this value the '*uval*' of the unique name.

Sorts: *blocks, uniquenames.*

Predicates and functions

*blockof: points*→*blocks*

*justinside, inside*⊆*blocks*×*blocks*

*localto*⊆*identifiers*×*blocks*

*uniquename: identifiers*×*blocks*→*uniquenames*

(given an identifier and the block in which it is used, produces the appropriate unique name)

*uval: uniquenames*×*states*→*values.*

(the value associated with a unique name)

*assign*⊆*uniquenames*×*values*×*states*×*states*

Axioms

Blocks

*b justinside b'*⇒*b inside b'*

*b inside b' and b' inside b"*⇒*b inside b"*

*b inside b'*⇒*b*≠*b'*

*p' begins p*⇒
*blockof*(*p'*) *justinside blockof*(*p*)
*and* [*i localto blockof*(*p'*)⇔*i declaredat p*]

*p' follows p or p has ifthen*(*e, p'*)⇒
*blockof*(*p'*) = *blockof*(*p*)

Unique names

*i*≠*i'*⇒*uniquename*(*i, b*) ≠ *uniquename*(*i', b*)

*b inside b' and i localto b*⇒*uniquename*(*i, b*) ≠ *uniquename*(*i', b'*)

*b justinside b' and i not localto b⇒*
*uniquename(i, b) = uniquename(i, b')*

Value associated with a unique name

*s at p⇒*
*[identifier(i)⇒val(i, s) = uval(uniquename(i, blockof(p)), s)]*

Definition of assign

*assign(u, v, s, s')⇒*
*uval(u, s') = v*
*and [u' ≠ u⇒uval(u', s') = uval(u', s)]*

Assignment (replaces previous axiom)

*p has assignment(i, e) and p' follows p and s at p⇒*
*next(s) at p'*
*and assign(uniquename(i, blockof(p)), val(e, s), s, next(s))*

Definition of jumpsto (replaces previous definition)

*s jumpsto p⇒*
*next(s) at p and [uval(u, next(s)) = uval(u, s)]*

Block beginning

*p' begins p and s at p⇒s jumpsto p'*

Block ending

*p'' ends p and p''' follows p and s'' at p''⇒*
*s'' jumpsto p'''*

## 5. SIMPLE NON-TYPE PROCEDURES

We assume non-type, non-recursive procedures with just one parameter and that called by value. The restriction to one parameter is not important. We adopt it merely to avoid some tedious processing of *n*-tuples in matching actual to formal parameters. In fact the way we handle scopes of variables would allow extension to any number.

### 5.1 Structure of procedures

New predicates and functions

*proceduredecl: identifiers × identifiers × points→statements*
(the arguments are: procedure identifier, formal parameter identifier, first point of body)
*call: identifiers × expressions→statements*

### 5.2 Example of a program with procedures

This and subsequent example programs are intended to show the method of obtaining descriptive sentences. No proof of their outcome is given but the reader may like to consider the inferences which might be drawn using the semantic axioms below.

(*1*) **begin integer** *j;* *i declaredat p1⇔i* = **j** *or i* = **g.**

| | |
|---|---|
| (2) **procedure** $g(k)$; **value** $k$; | *p2 begins p1.* |
| | *p2 has proceduredecl*(**g**,**k**,*p3*). |
| (3) $j:=j+k$; | *p3 follows p2.* |
| | *p3 has assignment*(**j**, *plus*(**j**, **k**)). |
| (4) | *p4 follows p3.* |
| (5) $j:=1$; | *p5 follows p2. p5 has assignment*(**j**, *one*). |
| (6) **begin integer** $j$; | *p6 follows p5. i declaredat p6⇔i=***j**. |
| (7) $j:=0$; | *p7 begins p6. p7 has assignment*(**j**, *zero*). |
| (8) $g(2)$ | *p8 follows p7. p8 has call*(**g**, *two*). |
| (9) **end** | *p9 follows p8. p9 ends p6.* |
| (10) **end** | *p10 follows p6. p10 ends p1.* |

*Notes.* The body of a procedure may be a block rather than a simple statement. The previous rules apply.

We handle the procedure declaration as the first statement of the block.

### 5.3 Semantics of procedures

New sort: *procedures* (a subsort of *values*). *Points* become a subsort of *values.*

New predicates and functions

*procedure: uniquenames→procedures*
*formalparameter: procedures→uniquenames*
*entry, exit: procedures→points*
*linkname: procedures→uniquenames*

*Notes.* (a) Each textual occurrence of a procedure declaration produces a procedure.

(b) Each procedure has a link name, which is a unique name. At any point during execution of the procedure the contents of the link name is the point to which the procedure must return (obviously this does not permit recursion).

Axioms

procedure declaration

*p has proceduredecl*($f$, $i$, $p'$) *and p″ follows p and p‴ follows p′*
*and s at p and* $\pi$=*procedure*(*uniquename*(*f*, *blockof*(*p*))⇒
*s jumpsto p″*
*and formalparameter*($\pi$)=*uniquename*(*i*, *blockof*(*p′*))
*and entry*($\pi$)=*p′*
*and exit*($\pi$)=*p″*.
*p has proceduredecl* (*f*, *i*, *p′*)⇒
*blockof* (*p′*) *justinside blockof* (*p*)
*and* [*i′ localto blockof* (*p′*)⇔*i′*=*f or i′*=*i*].

procedure entry

*p has call*(*f*, *e*) *and p′ follows p and s at p*
*and procedure*(*uniquename*(*f*, *blockof*(*p*)))=$\pi$⇒
*next*(*next*(*s*)) *at entry*($\pi$)

*and assign(formalparameter*$(\pi)$, *val*$(e, s)$, $s$, *next*$(s)$)
*and assign(linkname*$(\pi)$, $p'$, *next*$(s)$, *next*(*next*$(s)$))

procedure exit

*exit*$(\pi) = p$ *and* $s$ *at* $p$ *and uval(linkname*$(\pi), s) = p' \Rightarrow$
$s$ *jumpsto* $p'$

## 6. RECURSIVE NON-TYPE PROCEDURES

We maintain the restriction to one parameter, and that called by value, but allow recursion. We have to introduce the notion of an 'activation' of a procedure and the notion of a 'cell' i.e., a unique name together with an activation, what Strachey calls a 'left-hand value'. The value associated with a unique name is now defined as the contents of the cell associated with it in the current activation. Procedures as parameters are not allowed; for this we would have to associate an activation with a procedure.

### 6.1 Structure of recursive procedures

No change is needed to allow recursion. Indeed it could be argued that we were remiss in specifically disallowing it in the previous axioms.

### 6.2 Example of a program with a recursive procedure

| | |
|---|---|
| (*1*) **begin integer** $j$; | *i declaredat p1* $\Leftrightarrow i = \mathbf{j}$. |
| (*2*) **procedure** $g(k)$; **value** $k$; | *p2* begins *p1*. |
| | *p2 has proceduredecl*(**g**, **k**, *p3*). |
| (*3*) **begin** | *i not declaredat p3*. |
| (*4*) **if** $k=0$ **then** | *p4 begins p3*. |
| | *p4 has ifthen*(*equal*(**k**, *zero*), *p5*). |
| (*5*) **goto** *out*; | *p5 has goto*(**out**). |
| (*6*) | *p6 follows p5*. |
| (*7*) $g(k-1)$; | *p7 follows p4*. |
| | *p7 has call*(**g**, *minus*(**k**, *one*)). |
| (*8*) $j := k \times j$; | *p8 follows p7*. |
| | *p8 has assignment*(**j**, *times*(**k**, **j**)). |
| *out:* (*9*) | **out** *labels p9. p9 follows p8*. |
| **end** | *p9 ends p3*. |
| (*10*) | *p10 follows p3*. |
| (*11*) $j := 1$; | *p11 follows p2*. |
| | *p11 has assignment*(**j**, *one*). |
| (*12*) $g(1)$ | *p12 follows p11. p12 has call*(**g**, *one*). |
| (*13*) **end** | *p13 follows p12. p13 ends p1*. |

### 6.3 Semantics of recursive procedures

New sorts: *activations*, *links* (*links* are a subsort of *values*), *cells*.
New predicates and functions

*justafter*, *after* $\subseteq$ *activations* $\times$ *activations*

*cell: uniquenames × activations → cells*
*contents: cells × states → values*
*linkname: procedures → uniquenames*
*link: points × activations → links*

Axioms

Abbreviation: we write $next^2(s)$ for $next(next(s))$, similarly $next^3(s)$ etc.

Activations

*a justafter a' ⇒ a after a'.*
*a after a' and a' after a'' ⇒ a after a''.*
*a after a' ⇒ a ≠ a'.*

Values

$uval(u, s) = contents(cell(u, activation(s)), s)$

Definition of newactivation

*newactivation(s, s') ⇒*
*activation(s') justafter activation(s)*
*and contents(c, s') = contents(c, s)*

Definition of changeactivation

*changeactivation(a', s, s') ⇒*
*activation(s') = a'*
*and contents(c, s') = contents(c, s)*

Definition of assign

*assign(u, v, s, s') ⇒*
*activation(s') = activation(s)*
*and [c = cell(u, activation(s)) ⇒*
*contents(c, s') = v*
*and [c' ≠ c ⇒ contents(c', s') = contents(c', s)]]*

Definition of jumpsto

*s jumpsto p ⇒*
*next(s) at p*
*and activation(next(s)) = activation(s)*
*and contents(c, next(s)) = contents(c, s)*

Procedure entry

*p has call(f, e) and p' follows p and s at p*
*and procedure(uniquename (f, blockof(p))) = π ⇒*
*newactivation(s, next(s))*
*and assign(formalparameter(π), val(e, s), next(s), next²(s))*
*and assign(linkname(π), link(p', activation(s)), next²(s),*
*next³(s))*
*and next³(s) at p'*

Procedure exit

*exit(π) = p and s at p and link(p', a') = uval(linkname(π), s) ⇒*
*next(s) at p' and changeactivation(a', s, next(s))*

## 7. ARRAYS

We introduce integer arrays with just one subscript. As in the case of procedure parameters the restriction to one subscript is not important and is adopted simply to avoid the processing involved in handling $n$-tuples of subscripts. Array declarations resemble procedure declarations. A function *subscript* is introduced in the structure description for building up array expressions. We do not give axioms for procedures with array parameters, but the method used should cope quite easily with arrays called by value, given a function which assigns a type to each identifier.

### 7.1 Structure of arrays

New predicates and functions

*arraydecl:identifiers* × *expressions* × *expressions*→*statements*
*subscript: identifiers* × *expressions*→*expressions*

### 7.2 Example of a program with arrays

| | |
|---|---|
| (*1*) **begin integer** *j;* | *i declaredat p1*⇔*i*=**j** *or i*=**a**. |
| (*2*) **integer array** *a*[*0:2*]; | *p2 begins p1. p2 has arraydecl*(**a**, *zero*, *two*). |
| (*3*) *j*:=*0;* | *p3 follows p2. p3 has assignment*(**j**, *zero*). |
| (*4*) *a*[*j*]:=*1*; | *p4 follows p3.* |
| | *p4 has assignment*(*subscript*(**a**, **j**), *one*). |
| (*5*) *j*:=*a*[*j*]+*1* | *p5 follows p3.* |
| | *p5 has assignment*(**j**, *plus*(*subscript*(**a**, **j**), *one*)) |
| (*6*) **end** | *p6 follows p5. p6 ends p1.* |

### 7.3 Semantics of arrays

A function *elementname* is introduced which given an array and an integer produces a unique name. Thus the expressions *a*[*1*] and *a*[*2*] are treated rather as if they gave rise to new identifiers *a1* and *a2*, a viewpoint which underlies a number of ALGOL implementations.

New sorts: *arrays* (a subsort of *values*), *integers* (a subsort of *values*)
New predicates and functions

*array: identifiers* × *points*→*arrays*
*elementname: arrays* × *integers*→*uniquenames*
*upperbound: arrays*→*integers*
*lowerbound: arrays*→*integers*

Axioms

Element names

*elementname*(α, *n*) ≠ *uniquename*(*i*, *b*)
*elementname*(α, *n*) ≠ *linkname*(π)
*elementname*(α, *n*) = *elementname*(α′, *n*′)⇒α=α′ *and n*=*n*′

Array declaration

*p has arraydecl(i,e,e') and p' follows p and s at p and α=array(i,p)⇒*
*next(s) at p'*
*and assign(uniquename(i, blockof(p)), α, s, next(s))*
*and lowerbound(α) = val(e, s)*
*and upperbound(α) = val(e', s)*

Array expressions

*val(i, s) = α and val(e, s) = v*
*and lowerbound(α) ⩽ v and v ⩽ upperbound(α) ⇒*
*val(subscript(i, e), s) = uval(elementname(α, v), s)*

Assignment

Replace the previous axiom by the following two (the first being just a restriction of the previous axiom to assignments to identifiers).

*identifier(i) and p has assignment (i, e) and p' follows p and s at p⇒*
*next(s) at p'*
*and assign(uniquename(i, blockof(p)), val(e, s), s, next(s)).*
*p has assignment(subscript(a, e), e') and p' follows p and s at p⇒*
*next(s) at p'*
*and assign(elementname(α, val(e, s)), val(e', s), s, next(s))*

## 8. TYPE PROCEDURES AND SIDE-EFFECTS

We now consider type procedures, i.e., those which produce a result and may be used in an expression with possible side-effects. We will maintain the previous restriction to one parameter, and that called by value. Since we have some binary operations (plus and times) we will still encounter problems about the order of evaluation of sub-expressions of an expression. Declarations of type procedures are very like those of non-type procedures. To deal with side-effects we have to attach points, not just to statements, but also to expressions formed by applying a type procedure to an argument. We are interested in the states which arise at such points.

### 8.1 Structure of type procedures

New predicates and functions

*intproceduredecl: identifiers × identifiers × points → statements*
*application: identifiers × expressions → expressions*
*expr: points → expressions*

### 8.2 Example of a program with type procedures

| | |
|---|---|
| (*1*) **begin integer** *j;* | *i declaredat p1⇒i=*j *or i=*g. |
| (2) **integer procedure** *g*(*k*); | *p2 begins p1.* |
| | *p2 has intproceduredecl*(g, k, *p3*). |
| **value** *k;* | |
| (*3*) **begin** | *i not declaredat p3.* |

| | |
|---|---|
| (4) $j:=1$; | *p4 begins p3.* *p4 has assignment*(**k**, *one*). |
| (5) $g:=k$ | *p5 follows p4.* *p5 has assignment*(**g**, **k**). |
| (6) **end**; | *p6 follows p5. p6 ends p3.* |
| (7) | *p7 follows p3.* |
| (8) $j:=0$; | *p8 follows p2.* *p8 has assignment*(**j**, *zero*). |
| (9) $j:=^{(12)}g(^{(10)}g(0)+\ ^{(11)}g(j))$ | *p9 has assignment*(**j**, *expr*(p12)). *p12 has application* (**g**, *plus*(*expr*(*p10*), *expr*(*p11*)). *p10 has application*(**g**, *zero*). *p11 has application*(**g**, **j**). *p10 follows p9.* *p11 follows p10. p12 follows p11.* |
| (13) **end** | *p13 follows p12. p13 ends p1.* |

*Notes.* (a) The points have been attached to the sub-expressions in line (9) in the order in which they are to be evaluated. A better sentence which allows arbitrary order of evaluation would be

[*p10 follows p9 and p11 follows p10 and p12 follows p11*]
or [*p11 follows p9 and p10 follows p11 and p12 follows p10*]

If both orders of evaluation lead to the same result (here they do not) then this result can be proved in spite of the disjunction, since from

$P$ or $Q$. $P \Rightarrow R$. $Q \Rightarrow R$

we may deduce $R$.

If however they would lead to different results we may only deduce that the effect of the program is one or other of these.

(b) The attachment of points and formulation of sentences involving *follows* is more complicated here than for statements although it is easier if the reverse Polish form of the expression is borne in mind. Although the Appendix outlines the method for attaching points we leave quite a lot to the reader's imagination here.

### 8.3 Semantics of type procedures

We need some notion of the value of an expression which has a point attached to it, i.e., one involving application of a type procedure. We might well introduce the notion of a stack of intermediate results. However we have tried to avoid such machine-oriented devices and content ourself with introducing a unique name (stackname) associated with each point to hold the result of the expression attached to the point. These unique names (or rather the cells belonging to them) actually correspond to the positions on the stack, although in a many–one rather than one–one manner. In fact the stack mechanism economizes on such cells by writing new intermediate results

over previous ones. We sacrifice such economies for greater lucidity.

New predicates and functions:

*resultname: procedures→uniquenames*

*stackname: points→uniquenames*

Axioms

Values

*val(expr(p),s) = uval(stackname(p),s)* (an extra axiom)

Type procedure declaration

*p has intproceduredecl(f, i, p') and p'' follows p and p''' follows p'*
*and s at p and π = procedure(uniquename(f, blockof(p))) ⇒*
*s jumpsto p''*
*and formalparameter(π) = uniquename(i, blockof(p'))*
*and entry(π) = p' and exit(π) = p''*
*and resultname(π) = uniquename(f, blockof(p')).*
*p has proceduredecl (f, i, p') ⇒*
*blockof (p') justinside blockof (p)*
*and [i' localto blockof (p') ⇔ i' = f or i' = i]*

Type procedure entry

*s at p and p' follows p and p' has application(f, e) ⇒*
*[π = procedure(uniquename(f, blockof(p))) and v = val(e, s) ⇒*
*newactivation(s, next(s))*
*and assign(formalparameter(π), v, next(s), next³(s))*
*and assign(linkname(π), link(p', activation(s)), next²(s),*
*next³(s))*
*and next³(s) at entry(π)]*

Type procedure exit

*s at p and p = exit(π) and v = uval(resultname(π),s)*
*and link(p', a') = uval(linkname(π, s)) ⇒*
*contents(c, next(s)) = contents(c, s))*
*and activation(next(s)) = a'*
*and assign(stackname(p'), v, next(s), next²(s))*
*and next²(s) at p'*

## 9. LIST PROCESSING

It is worth going a little beyond the confines of ALGOL 60 to show how to handle lists with shared components. These will stand as a paradigm for more elaborate data structures such as 'records' or 'plexes'. We will just add elementary operations on lists to ALGOL 60 in the obvious way.

### 9.1 Structure of list processing programs

New predicates and functions

*head, tail, atom, null: expressions→expressions*
(*head* and *tail* are LISP '*car*' and '*cdr*')
*cons: expressions × expressions→expressions*
*nil ∈ expressions*

### 9.2 Example of a program with list processing

We choose a very simple program since no new syntax is involved. Note that *cons* produces a side effect and so must be treated rather as if it were a type procedure.

| | |
|---|---|
| (1) **begin list** *j, k;* | *i declaredat p1*⇔*i*=**j** *or i*=**k**. |
| (2) *j*: =(3)*cons*(*0, nil*); | *p2 begins p1. p2 has assignment*(**j**, *expr*(*p3*)). *p3 follows p2. p3 has cons*(*zero, nil*). |
| (4) *k*: =(5)*cons*(*1, j*); | *p4 follows p3. p4 has assignment*(**k**, *p5*). *p5 follows p4. p5 has cons*(*one*, **j**). |
| (6) *head*(*j*): =*2* | *p6 follows p5. p5 has assignment*(*head*(**j**), *two*). |
| (7) **end** | *p7 follows p6. p7 ends p1.* |

### 9.3 Semantics of list processing

New sorts: *nodes* (a subset of *values*). These stand for list cells.

New predicates and functions

*nextnode: nodes*→*nodes* (gives the next free node)
*laternode*⊆*nodes*×*nodes*
*fhead, ftail: nodes*→*values*
*fnil* ∈ *nodes*
*fnull*⊆*nodes*
*fatom*⊆*values*
*freenode: states*→*nodes* (gives the first free node of a state)

Axioms

Nodes

*laternode*(*nextnode*(*n*),*n*).
*laternode*(*n*″, *n*′) and *laternode*(*n*′, *n*)⇒*laternode*(*n*″,*n*)
*laternode*(*n*′, *n*)⇒*n*′≠*n*
*freenode*(*s*)≠*fnil*. *nextnode*(*n*)≠*fnil*.
(these ensure a set of distinct nodes different from *fnil*)

Null and atom

*not fatom*(*freenode*(*s*))
*not fatom*(*fnil*)
*fnull*(*n*)⇔*n*=*fnil*
*fatom*(*v*) *and not*(*fatom*(*n*))⇒*v*≠*n*
*fatom*(*true*). *fatom*(*false*). *numeral*(*n*)⇒*fatom*(*n*).

Values

*val*(*head*(*e*),*s*)=*fhead*(*val*(*e, s*))
Similarly for *tail*, *null* and *atom*.
*val*(*nil, s*) =*fnil*

Assignment

Add to the existing assignment and jumpsto axioms:
*andfhead*(*n, next*(*s*)) =*fhead*(*n, s*) *andftail*(*n, next*(*s*)) =*ftail*(*n, s*)
*and freenode*(*next*(*s*)) =*freenode*(*s*)

Add new axioms:

$assignhead(n, v, s, s') \Rightarrow$
$[n' \neq n \Rightarrow fhead(n', s') = fhead(n', s)]$
$and\ fhead(n, s') = v$
$and\ ftail(n', s') = ftail(n', s)$
$and\ freenode(s') = freenode(s)$
$and\ contents(c, s') = contents(c, s)$
$and\ activation(s') = activation(s).$
$s\ at\ p\ and\ p\ has\ assignment(head(e), e')\ and\ p'\ follows\ p \Rightarrow$
$next(s)\ at\ p'$
$and\ assignhead(val(e, s), val(e', s), s, next(s)).$
Similarly for *tail*.

Cons

$s\ at\ p\ and\ p'\ follows\ p\ and\ p'\ has\ cons(e, e')\ and\ freenode(s) = n \Rightarrow$
$next(s)\ at\ p'$
$and\ freenode(next(s)) = nextnode(n)$
$and\ fhead(n, next(s)) = val(e, s)$
$and\ ftail(n, next(s)) = val(e', s)$
$and\ [n' \neq n \Rightarrow fhead(n', next(s)) = fhead(n', s)$
$and\ ftail(n', next(s)) = ftail(n', s)]$
$and\ contents(c, next(s)) = contents(c, s)$
$and\ activation(next(s)) = activation(s)$

## 10. CONCLUSIONS

We have omitted quite a number of features of ALGOL 60: call by name, procedures and arrays as parameters, own variables and switches. A tentative look at these has revealed some complexity but no fundamental obstacle to applying the techniques used above. This is not surprising since, in a way, all that has been done here is to write an ALGOL interpreter using predicate calculus as a programming language, and any features which can be dealt with by a compiler should be capable of being handled in predicate calculus. Indeed we should be able to mirror in predicate calculus the various alternative abstract machines for running ALGOL (or other) programs.

One question which may have occurred to the reader is whether first order logic is the most lucid form of logic for defining a programming language, granted the advantage that it can be mechanized. It seems likely that a second or higher order formulation would make it easier to use existing mathematical concepts and theorems, for example by stating that relations like *inside* and *after* are orderings. Robinson (1970) has pointed out that it is possible to formulate higher order logic within first order logic so that there is some prospect of a mechanizable higher order system.

I have experimented a little with mechanization using, first, a resolution theorem prover written by Isobel Smith in the Metamathematics Unit,

Edinburgh University, and, second, a resolution proof checker written by D.B.Anderson of this department. The theorem prover was able to go through a couple of assignment statements but produced a lot of useless clauses from the definitions of the values of expressions. Reformulating some of the axioms with a transitive 'reduction' relation in place of equality helped, but it was out of the question to prove correctness of even the first program given in one go. The proof checker was slower but more manageable. I got through the proof of the program in section 3 with 26 semantic axiom clauses, 23 arithmetic clauses, 16 program description clauses and 197 resolution inferences, many of them just transitivity of equality. This was using clashes as well as binary resolution and a special rule to do transitivity of 'equals' more easily. I conclude that mechanization, at least by a proof checker, is possible but needs tricks to make it less impossibly tedious.

One advantage of this approach to language definition, if it can be mechanized, is that it would allow a language designer to 'debug' the formal definition of his language by trying it out on a number of test programs in that language and checking mechanically that the expected outcome can be inferred. In fact we can, in theory, push the nastiness of 'suck-it-and-see' debugging back from debugging individual programs to debugging the language definition.

**Acknowledgements**

These ideas were conceived while I was a visiting lecturer at the Computer Science Department, University of Helsinki, and they were worked out as part of the Machine Intelligence project at Edinburgh University, sponsored by the Science Research Council. I would like to thank Christopher Strachey for teaching me about the formal semantics of programming languages and Bernard Meltzer and his colleagues in the Meta-mathematics Unit at Edinburgh University for many helpful talks on theorem proving. Thanks are due to Miss Eleanor Kerse for typing the manuscript.

## REFERENCES

Allen, C.D., Ashcroft, E.A. & Florentin, J.J. (1969) Inferences from formal language definitions. Presented at the *ACM SIGPLAN Conf. on Language Definition*, at San Francisco.

Ashcroft, E.A. (1970) Mathematical logic applied to the semantics of computer programs. *Ph.D. thesis*, to be submitted to Imperial College, London.

Burstall, R.M. (1968) Semantics of assignment. *Machine Intelligence 2*, pp. 3–20 (eds Dale, E. & Michie D.). Edinburgh: Oliver and Boyd.

Floyd, R.W. (1967) Assigning meanings to programs. *Mathematical Aspects of Computer Science*, pp. 19–32. Providence, Rhode Island: Amer. Math. Soc.

Green, C.C. (1969) Applications of theorem proving to problem solving. *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pp. 219–39 (eds Walker, D.E. & Morton, L.M.). Washington, D.C.

Hoare, C.A.R. (1969) An axiomatic basis for computer programs. *Comm. Ass. comput. Mach.*, **12**, 576–80.

Landin, P.J. (1970) Program/machine symmetric automata theory. *Machine Intelligence 5*, pp. 99–120 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Lauer, P. (1968) Formal definition of ALGOL 60. *Technical Report* 25.088, IBM Laboratory, Vienna.

Manna, Z. (1969) Properties of programs and the first order predicate calculus. *J. Ass. comput. Mach.*, **16**, 2.

Manna, Z. & McCarthy, J. (1970) Partial function logic and its applications. *Machine Intelligence 5* (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

McCarthy, J. (1963) Situations, actions and causal laws. *Stanford Artificial Intelligence Memo. No. 2*, reprinted in *Semantic Information Processing* (1968) (ed. Minsky, M.). Cambridge, Mass: MIT Press.

McCarthy, J. & Painter, J.A. (1967) Correctness of a compiler for arithmetic expressions. *Mathematical Aspects of Computer Science*, pp. 33–41. Providence, Rhode Island: Amer. Math. Soc.

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.

Popplestone, R.J. (1967) Private communication.

Robinson, J.A. (1970) A note on mechanizing higher order logic. *Machine Intelligence 5* (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Strachey, C. (1965) Towards a formal semantics. *Formal Description Languages for Computer Programming* (ed. Steel, T.B., Jr.). Amsterdam: North Holland.

## APPENDIX

### Obtaining the description of the program structure

We have shown, mainly by example, how a program may be converted into a set of first order sentences which describe its structure. The business of syntax analysis is well understood, and we can use context-free grammars to specify a transformation from a program to an analysis tree (or 'abstract expression') in a quite formal manner. There is however a slight gap in the above description of the method, in that the analysis tree is converted into first order sentences using some rules which have not been formally stated. Suppose for example that we have used a syntax analyser to describe the program

$$j:=0;$$
$$k:=1;$$
$$j:=k+1;$$

by using a two-place relation *isprogram* between programs and their analysis trees, thus:

*P isprogram semicolon*(*assignment*(**j**, *zero*), *semicolon*(*assignment*(**k**, *one*), *assignment*(**j**, *plus*(**k**, *one*)))).

Now we would like statements:

*p*1 *has assignment*(**j**, *zero*). *p*2 *follows p*1. *p*2 *has assignment*(**k**, *one*). etc.

since this form seems a good deal more convenient and intuitive for developing the semantic axioms.

The basic problem is this. Given some abstract expression (analysis tree) how can we systematically tag each subexpression with a marker and produce first order sentences showing the relationships between the markers on the subexpressions? There is a simple trick for doing this as the following example shows.

Suppose we have one associative binary function written '$\cdot$', and a set of atomic constants $a, b, c$ etc. A typical expression is

$$(a \cdot b) \cdot (a \cdot d)$$

which is of course equal to $a \cdot (b \cdot (a \cdot d))$.

We use a 2-place predicate *tags: points* $\times$ *expressions*, and, as before, *follows* $\subseteq$ *points* $\times$ *points* and *has* $\subseteq$ *points* $\times$ *atomic expressions*. We also postulate a function *succ: points* $\rightarrow$ *points* (to give us an infinite supply of points).

We use the axioms

(*i*) *p tags* $\sigma$ *and* $\sigma = \alpha \cdot \sigma'$ *and atomic*$(\alpha) \Rightarrow$
*p has* $\alpha$ *and succ*$(p)$ *follows p and succ*$(p)$ *tags* $\sigma'$.

and

(*ii*) *p tags* $\alpha$ *and atomic*$(\alpha) \Rightarrow$
*p has* $\alpha$.

Now the statement

*p1 tags*$(a \cdot b) \cdot (a \cdot d)$.

together with the associative axiom immediately yield

*p1 tags*$(a \cdot (b \cdot (a \cdot d))$.
*p1 has a . succ*$(p1)$ *follows p1.* *succ*$(p1)$ *tags* $b \cdot (a \cdot d)$
*succ*$(p1)$ *has b . succ*$(succ(p1))$ *follows succ*$(p1)$.
*succ*$(succ(p2))$ *tags*$(a \cdot d)$. etc.

Ignoring the sentences with predicate *tags* we see that we have just the sentences needed to describe the structure of the expression in the required manner.

It should be clear how we can use this method to obtain the sentences describing a program from its analysis tree. We will require axioms to introduce *follows* and also *begins*, *ends* and *enters*, etc. We leave the details to the reader.

It is perhaps worth remarking that Popplestone (1967) has pointed out that the syntax analysis phase itself can be done with a theorem prover. Thus we can (as an exercise in formalism) describe the program simply as a string of basic symbols, using just a concatenation operation, and give first order sentences to describe all phases of processing the string to obtain the result of executing the program.

# 5

# A Program Machine Symmetric Automata Theory

P. J. Landin
Queen Mary College
University of London

**Abstract**

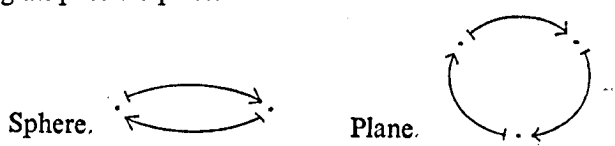
A computation is viewed as the generation of a subalgebra in a direct product of partial nondeterminate algebras. This leads to a definition of the operation $F_BA$ computed by one algebra, $A$, on another, $B$. There is an algebra-constructing operation by which any finite algebra can be derived from 'elementary' algebras. For a given $B$, $F_B$ maps this structure over algebras homomorphically to a structure over operations.

## INTRODUCTION

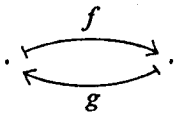
Suppose a directed graph with coloured edges is drawn on a plane, and another one drawn on the surface of a sphere. Imagine the sphere rolling, swivelling and slipping on the plane, but constrained in such a way that the point of contact follows a path through both graphs, moreover synchronizing the vertices and matching the colours. Started at a particular vertex-pairing this device might be unable to move, or there might be open to it just one direction, or it might have a choice. Of these three possibilities two lead to another vertex-pairing where the same three possibilities arise.

Consider the case when a choice never arises. Then the 'orbit' is given by a string of vertex-pairs. This might cycle on the plane, or on the sphere. It might cycle on both and if so the cycles need not synchronize. Cycling in either graph is avoided only because of a halt or because the graph is infinite (a possibility we tolerate).

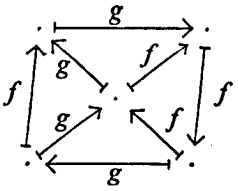
*Example* 1. The plane has a monochromatic 3-cycle and the sphere has a 2-cycle of the same colour. Wherever it is started the orbit will be a 6-cycle encountering all possible pairs.



*Example* 2. Two colours, fawn and green.




Sphere




Plane

Here each choice of starting pair leads to a determinate route, some halting and some cycling. But none of them visits all 10 possible pairs.

Such ball-on-plane systems are almost the same as the program-on-machine systems studied in this paper. The only serious difference is mentioned at the end of this introduction. Accordingly we introduce some terminology.

*Graph Terminology.* By a *graph* we always mean a directed graph with labelled (i.e., coloured) edges. Each edge has a *label*, a *source* vertex, and a *target* vertex. *Node* is a synonym for vertex. *Link*, *arrow*, *pointer* are synonyms for edge.

*Plane/Machine Terminology.* The plane-nodes are the *states.*

*Sphere/Program Terminology.* The sphere-nodes are the *instructions.* Thus in a program an operation can be thought of as *between* instructions not *at* them. In terms of an ALGOL 60 program the occurrences of semicolons (i.e. roughly the contexts that admit labels) are our instructions, the statements are our colours. The plane must then bear the transition diagram of the states of some 'ALGOL 60 machine'. (The ALGOL analogy is developed below.)

Rolling different balls on the same plane is like executing different programs on the same machine. In each such experiment a set of instruction–state pairs are visited. A particular instruction might touch many states, or just one, or none at all. A particular state might be touched by many instructions or just one, or none at all. The set of instructions that ever touch the plane might be the whole program or only part, or even just the initial instruction. Similarly with the set of states.

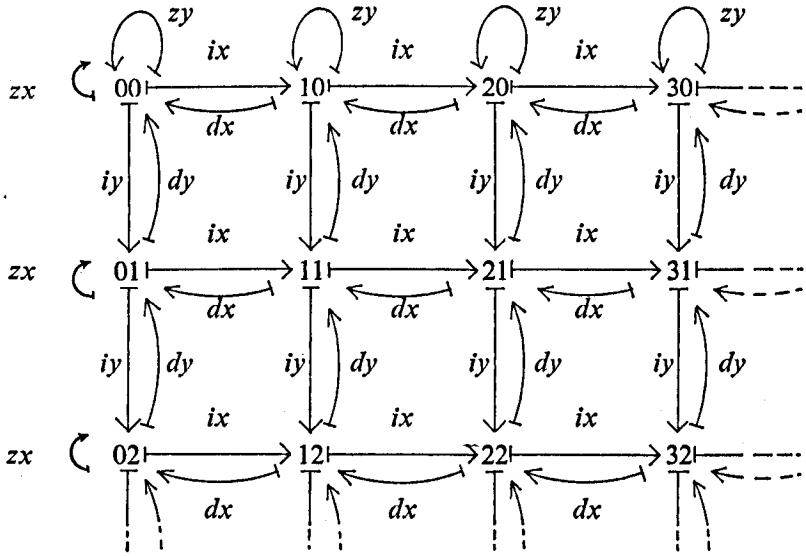
The set of pairs that touch is a subset of the direct product of the two graphs. It is also a relation (a 'nondeterminate partial function') that associates with each instruction zero or more states. Vacillation between these two aspects of the same set of pairings will underlie our treatment.

The asymmetry suggested by sphere and plane is of course spurious. The set-up is so far quite symmetrical. For example the above set of pairs can also be viewed as a nondeterminate partial function that associates with each machine-node zero or more program-nodes. Nevertheless the asymmetric terminology will be retained for informal explanation.

There are four respects in which the rolling sphere is an inadequate picture. For the first three we can fill in the gaps.

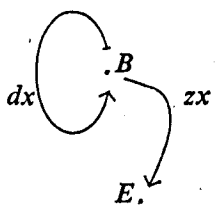
1. *Non-finiteness.* We do not exclude an infinity of instructions or of states. In the machine this infinity is familiar since it arises in non-finite interpretations e.g., in terms of numbers or of number-vectors. It sounds stranger for the program. However we shall want infinite programs for two reasons. We may wish to consider as a program the result of indefinitely many substitutions that 'eliminate' a self-referential definition. Also we may wish to consider as a program the set of all programs of a certain kind, e.g., the set of strings in a given alphabet (i.e., label-set).

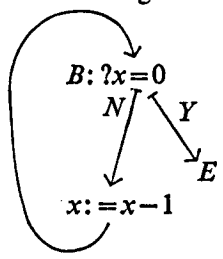
*Example* 3. Let the plane be an infinite quadrant with a grid of nodes and six colours of edges, *ix*, *iy*, *dx*, *dy*, *zx*, *zy*.




This can be thought of as the set of pairs of natural numbers $(x, y)$ with six partial operations: *i*ncrement $x$ by one; ditto $y$; *d*ecrement $x$ by one if non zero and otherwise undefined; ditto $y$; identity restricted to $x$ being zero; ditto $y$. It can also be thought of as a two-register machine. Now the following sphere begun with node $B$ touching any grid point will roll to the west edge and stop. It is a program for clearing the $x$ register.

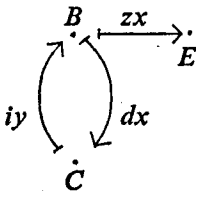
Alongside the graph is a flow diagram and an ALGOL 60 program, each of which can be thought of as an alternative representation of the graph according to rules that can be stated generally.

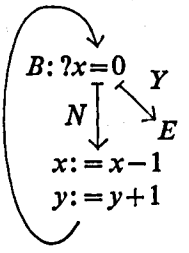



```
B: if x≠0 then
   begin x:=x−1;
      goto B
   end
else if x=0 then
   goto E
```

Again the following program will zigzag south-westerly and add its coordinates.



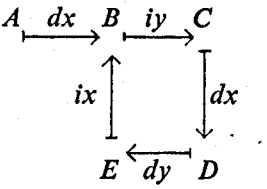




```
B: if x≠0 then
   begin x:=x−1;
         y:=y+1;
         goto B
   end
else if x=0 then
   goto E
```

If begun at the instruction-state pair $(B, 21)$ this ball-and-plane will follow an 'orbit' that can be pictured as the following graph:

$$(B,21) \xrightarrow{dx} (C,11) \xrightarrow{iy} (B,12) \xrightarrow{dx} (C,02) \xrightarrow{iy} (B,03) \downarrow zx \; (E,03)$$

Looking ahead we shall give definitions by which this orbit is the 'sub-algebra generated by $(B, 21)$ in the direct product of' the program and the machine. In the present example the orbit has a highly special shape, consisting of a single chain. For a third example with the same machine consider the figure below.




Here are two of its orbits, determined by starting at $(A, 11)$ and $(A, 21)$ respectively:

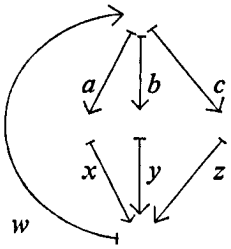
$$(A,11) \xrightarrow{dx} (B,01) \xrightarrow{iy} (C,02)$$

$$(A,21) \xrightarrow{dx} (B,11) \xrightarrow{iy} (C,12) \downarrow dx \; (D,02) \xrightarrow{dy} (E,01) \xrightarrow{ix} (B,11)$$

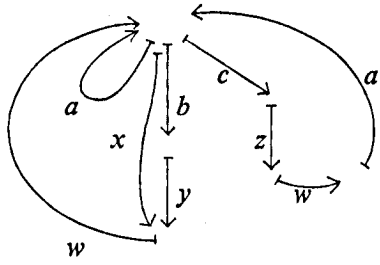
Cycling is not the only cause of non-chains. Points (2) and (4) below give two other sources of elaborately shaped 'orbits' that we wish to take account of.

2. *Nondeterminateness.* If choices arise for the rolling sphere we are interested in considering all possible behaviours, not just one. This is sometimes 'parallel computation' as in the evaluation of a term. Sometimes it is 'back-up' (or 'look-ahead' depending on your point of view) as when two branches must be pursued to discover which of them is eventually not barren. Sometimes it is 'genuinely nondeterminate' in the sense of yielding two answers, and is interesting only as an auxiliary to another system that is constrained to imitate it in some way. This situation is familiar in system specification, e.g., the formal description of a programming language.

*Example* 4.

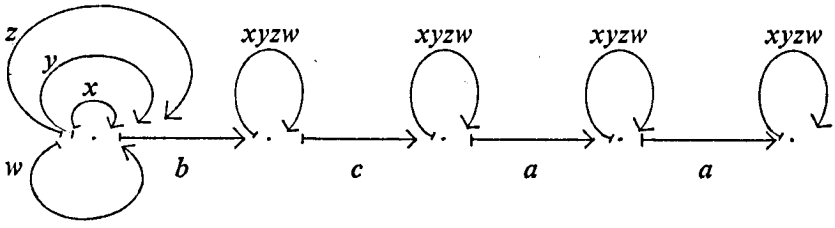



Plane



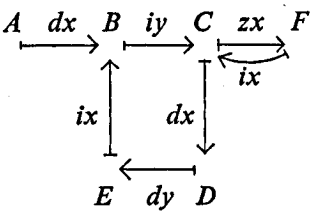

Sphere

The sphere can roll around the loop on the plane, choosing which branch, except that after choosing *c* it is forced into *a* the following time round. But now let us complicate the picture by adding a *third* graph. Physically this might be on another plane that can slide quite freely against the original one, except for the same condition as before: the coincident point must traverse each graph with matching labels. Suppose this third plane is a strip with

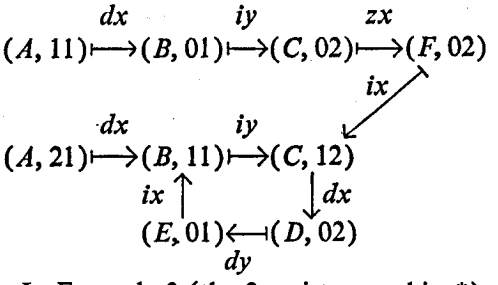



(where we abbreviate four 1-cycles to one with a composite label). Then this strip determines the behaviour. So does any similar strip. If the two original graphs are thought of as machine and program, this third one can be thought of as data, a tape of *a*, *b* and *c* that is traversed until it comes to a *c* not followed by an *a*.

*Example* 5. We do not exclude a nondeterminate start. Consider the following program for the machine of Example 3:




Started at the set $\{(A, 11), (A, 21)\}$ we get




3. *Start and finish.* In Example 3 (the 2-register machine*) we were able to say that a certain program computes the sum of its initial number-pairs. In the example we let the ball roll to a stand-still. Other devices for termination are a designated subset of end-states or a designated subset of end-instructions.

In general, given a plane, any sphere with marked begin-nodes and end-nodes 'computes' a 'partial nondeterminate unary function' over the set of nodes on the plane.

The marks on the sphere have disturbed the symmetry. In the more general definitions that come below, the symmetry is restored. A definition is given of the operation computed by one graph (sans marks) on another. Also a definition is given of the operation computed by one 'marked graph' on another.

4. *Polyadicity.* A label (i.e., a colour) in a directed graph determines a partial nondeterminate operation over the set of vertices. This operation is necessarily unary. We want to relax the latter condition and allow polyadic (including unary and nullary) operations. Moreover we want to restore source/target symmetry by allowing operations that may have several arguments *and* several results. We need this to encompass, for example, multi-entry multi-exit flow-diagrams. When the appropriate definition comes

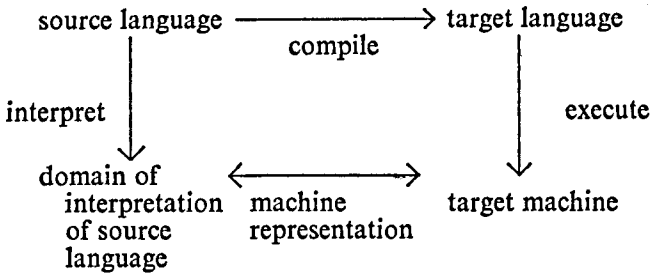
* This machine is Minsky's (1967) 2-register machine with Karp's (1959) device for using partial functions as a way of doing branching. The example is remarkable in that for any partial recursive function of one number there is a finite sphere-graph that essentially computes it. Thus the machine is in an important sense as powerful as any computing device can be.

below we shall mention in passing a corresponding generalization of the graphs, called 'polygraphs'. This serious generalization means replacing the rolling sphere by a prodigiously athletic deformable surface that keeps all its contact-points moving correctly by occasional resort to the fourth dimension.

The rolling sphere metaphor can just about be stretched to encompass infinity, nondeterminacy, and end-conditions. But polygraphs beat it. So the ball-and-plane terminology will not be mentioned again. However, most of the properties mentioned below can helpfully be illustrated in terms of the unary special case that we have been discussing in this introduction.

## MOTIVATION

The present work is another step in the direction of Burstall and Landin (1969), although it does not require familiarity with that paper. Briefly the step made there was to express the compilation of a very simple programming language as an algebraic transformation. The fact that the execution of a compiled program was a 'correct' way of proceeding, was expressed by a commutative diagram whose significance is informally indicated below.




The algebraic formulation of this situation was a little less than straightforward since not all the arrows are homomorphisms of algebras. Nevertheless they depended on homomorphic maps and thus on the algebraic structure of the objects indicated at the nodes of the diagram. In the rather trivial case treated there the required algebraic structure was immediately forthcoming since the source language was a familiar algebra of trees [i.e., 'terms' in logic or 'Ω-words' as used by Cohn (1965)] and the target language had semigroup structure familiar from automata theory. In so far as the treatment was straightforward, this was due to the following:

(i) the source language was pure expressions. It had no difficult structure such as assignments, conditions, or jumps.
(ii) the target language was pure assignments. It had no difficult structure like jumps, or computed addresses.

If the treatment is to work for more practical situations, these restrictions must be relaxed. We need to be able to consider ALGOL-like source programs as having an algebraic structure that can be exploited by homomorphic maps,

both compiling-maps and interpreting-maps. The need can be indicated by the gap in the following diagram:

| | Restricted to unary operations | Allow polyadic operations |
|---|---|---|
| Restricted to programs whose computation is the same 'shape' as the program | strings interpreted in a machine (sequential machines) | trees interpreted in an arbitrary algebra ('generalized automata theory') |
| Allow branches, joins and loops | program schemes interpreted in an arbitrary function/predicate structure | ? |

Automata theory was first concerned with a machine reacting to successive symbols drawn from some input alphabet, changing its state, and possibly producing an output. If we disregard the output this is a special case of two divergent generalizations as shown in the above $2 \times 2$ table:

1. A 'generalized automaton' in Thatcher and Wright's (1968) sense receives a labelled tree (i.e., what in logic is called a 'term') and in a nonsequential way it 'evaluates' the tree considered as an expression interpreted in general algebra. This specializes to the earlier case if the operators consist of one nullary and the rest unary.
2. A single-cell program schema is interpreted in an algebraic structure that has both functions and predicates. Predicates are encompassed by the device of Karp (1959), namely replacing them by pairs of functions whose intended interpretation is as complementary restricted identities; e.g., instead of the numerical predicate $\geqslant 0$ we use the restricted identities $I_{\geqslant 0}$ and $I_{<0}$.

Is there something that comprehends them all? The present paper fills this gap. That is to say, it provides something that provides both the left-side and the right-side of the desired commutative diagram. Compiling can now be formulated as the activity of representing one such system by means of another.

## NONDETERMINATE OPERATIONS AND ALGEBRAS

What follows takes for granted a set $U$ whose members play somewhat the role of 'machine-states', 'instructions', and other things of which the most important is instruction–state pairs.

*Definition.* A **string** is a function whose domain is the set $\{0, 1, 2, \ldots, n-1\}$ for $n \geqslant 0$.

*Definition.* A **nondeterminate operation** is a correspondence between strings.

*Definition.* For sets $A$ and $B$, $A \multimap B$ is the set of correspondences between $A$ and $B$, i.e., the set of 'partial nondeterminate functions' from $A$ to $B$.

*Definition.* For $f \in A \multimap B$, $x \subseteq A$, $Im_f x$ is the image of $x$ by $f$.

*Definition.* The **dot product** $g \cdot f$ of two correspondences is defined by

$$(x, z) \in g \cdot f \text{ iff } \exists y \,.\, (x, y) \in f \text{ and } (y, z) \in g.$$

In particular if $f$ and $g$ are functions then $g \cdot f$ is the function such that

$$(g \cdot f)(x) = g(fx).$$

*Definition.* The **sum** $f+g$ of two functions that produce sets is defined by

$$(f+g)x = fx \cup gx.$$

More generally for a set $F$ of functions $(\sum F)x = \bigcup_{f \in F} fx$.

So $f+g = f \cup g$ iff $f$ and $g$ have disjoint domains or agree on the overlap; otherwise both are defined but $f+g$ is a function and $f \cup g$ is not.

*Definition.* The **dagger** $f^\dagger$ of a set-function is defined by:

$$f^\dagger(x) = \bigcup_{i \geq 0} f^i(x).$$

Note that $(f+g) \cdot h = f \cdot h + g \cdot h$, but not generally $f \cdot (g+h) = f \cdot g + f \cdot h$.

*Definition.* A function $f$ on sets is **increasing** if $f(x) \supseteq x$ for all $x$.

*Definition.* $I_S$ is the identity correspondence over the set $S$. If $S$ is the entire domain, or the entire set of subsets of it then, loosely, $I_U$ is used.

*Definition.* For given $x$ (usually a set) $Kx$ is the constant set function whose result is $x$ regardless of argument.

*Definition.* A function $f$ on sets is **additive** if

$$f(x \cup y) = fx \cup fy.$$

So if $f$ is additive then $f \cdot (g+h) = f \cdot g + f \cdot h$.

*Definition.* A set $x$ is $f$-**reduced** if $fx \subseteq x$.

*Definition.* A set of subsets is a **closure system** if it is closed under arbitrary intersections.

*Definition.* $f$ is **monotonic** if $x \subseteq y$ implies $fx \subseteq fy$.

*Definition.* $f$ is a **closure operation** if it is increasing, monotonic, and if $f^2(x) \subseteq f(x)$ for all $x$.

If $f$ is monotonic the set of $f$-reduced subsets is a closure system. So

*Definition.* For monotonic $f$ let $J_f$ be the closure-operation associated with the closure-system consisting of the $f$-reduced sets.

*Definition.* $f$ is **continuous** if for any ascending chain $x_0 \subseteq x_1 \subseteq \ldots$,

$$f(\bigcup_i x_i) = \bigcup_i (fx_i).$$

*Theorem*

For an increasing, monotonic and continuous function $f$, $f^\dagger = J_f$.

*Proof.* (i) $f(f^\dagger x) = \bigcup_i f^{i+1}x \subseteq f^\dagger x$

So $f^\dagger x$ is $f$-reduced.

So $J_f x \subseteq f^\dagger x$

(ii) $f(J_f x) \subseteq J_f x$

So $f^{i+1}(J_f x) \subseteq f^i(J_f x)$ by induction

So $\bigcup_i f^i(J_f x) \subseteq J_f x$

So $f^\dagger x \subseteq f(J_f x) \subseteq J_f x.$

*Theorem*

If $f$, $g$ are monotonic and continuous

$$J_{f+g} = (f^\dagger \cdot g^\dagger)^\dagger.$$

*Proof.* (Similar technique to preceding theorem.)

(i) Since $f$, $g$ are continuous

$$(f+g)((f^\dagger \cdot g^\dagger)^\dagger x) \subseteq \bigcup_i (f^\dagger \cdot g^\dagger)^i x$$

So any $(f^\dagger \cdot g^\dagger)^\dagger x$ is $(f+g)$-reduced. So $J_{f+g} x \subseteq (f^\dagger \cdot g^\dagger)^\dagger x$.

(ii) Conversely any $(f+g)$-reduced set is also $(f^\dagger \cdot g^\dagger)^\dagger$-reduced.

In particular $J_{f+g} x$ is.

So $(f^\dagger \cdot g^\dagger)^\dagger x \subseteq (f^\dagger \cdot g^\dagger)^\dagger (J_{f+g} x) \subseteq J_{f+g} x$.

*Corollary.* $(f+g)^\dagger = (f^\dagger \cdot g^\dagger)^\dagger$.

*Definition.* An **algebra** is an indexed set of nondeterminate operations, i.e., a function whose range is nondeterminate operations. The members of its domain are the **operators** of the algebra. The objects in the strings that are operated on are the **elements** of the algebra.

It is usually convenient to think of the operators as the symbols that are written to indicate them. But notice that changing just the operators of an algebra does change the algebra. This is a bit like saying that you get a different group if you indicate its binary operator by $+$ instead of $\times$. This fastidious regard for the operators is necessary when we are speaking of two algebras and want to pair off corresponding operations.

*Definition.* An algebra is **monotectonic** if each element occurs at most once throughout the results of all its operation–instances. It is **monogenic** if the analogous statement for arguments holds.

*Definition.* The **direct product** $A \times B$ of two algebras is defined by:

$$(((a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)), ((a'_1, b'_1), \ldots, (a'_n, b'_n))) \in (A \times B)_\omega$$

iff both $((a_1, \ldots, a_m), (a'_1, \ldots, a'_n)) \in A_\omega$

and $((b_1, \ldots, b_m), (b'_1, \ldots, b'_n)) \in B_\omega$.

### Pictorial presentation of an algebra – polygraphs

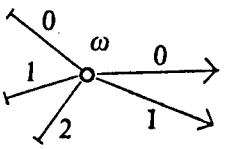
Two edges of a graph are **duplicates** if they agree on source, target, and label. Disregarding duplicates, labelled graphs have been absorbed into our treatment as unary algebras. So it is natural to seek a graphical representation of arbitrary algebras that specializes suitably in the unary case. 'Polygraphs' fill this bill. They are of course not necessary to a precise account but they and their terminology may aid intuition in the subsequent definitions.

*Definition.* A **polygraph** $\mathbb{A}$ is a set $A$ (its **vertices**), with a set $E$ (its **polyedges**), a set $\Omega$ (its **labels**), and a function $G$ associating with each polyedge $e$ the following three things:

1. A label, (the label or colour of $e$).
2. A vector of nodes, (the source vector of $e$).
3. A vector of nodes, (the target vector of $e$).

So $\mathbb{A}$ is without duplicates iff $G$ is one-one.

We draw a polyedge (e.g. one with in-arity 3 and out-arity 2) thus:




The indices for the sources and targets can often be omitted by assuming a left-to-right or top-to-bottom convention. We usually abbreviate 'polyedge' to 'edge'. A polygraph can be drawn as a network of edges with some cross-over convention if needed. By an '$\omega$-polyedge' we mean a polyedge bearing $\omega$ as its label. There is thus a (1, 1) correspondence between algebras and the polygraphs that have no duplicate polyedges.

The terminology corresponds as follows:

| *Algebras* | *Polygraphs* |
|---|---|
| element | vertex |
| operator | label |
| operation-instance | polyedge |
| operation | set of edges bearing the same label |
| argument-vector | source-vector |
| result-vector | target-vector |

If we draw the direct product of two polygraphs $P$ and $Q$, we shall get a vertex for *every* pair of vertices taken from $P$ and $Q$ respectively. In the 'orbits' diagrammed in the introduction this did not happen. The next section bridges this gap.

## ACCESSIBILITY AND SUBALGEBRAS*

For some observations about an algebra $A$ it is not necessary to distinguish the operators from one another. I.e., we are concerned with just one operation, namely $\cup A$ (forgetting the colours). When discussing 'accessibility' within an algebra we can also disregard serial ordering of the elements (very easy to forget in the polygraph).

* *Notation.* We indicate application either by juxtaposition or by subscription. e.g., $Ac_Ax=(AcA)x$ and $J(Ac_A)=Jac_A$.

*Definition.* The **immediate accessibility function** $Immac_A$ of an algebra $A$ is defined by

$$Immac_A\, x = \{b \mid \exists\omega \,.\, \exists a_1, a_2, \ldots, a_m \text{ all in } x \,.\, \exists b_1, \ldots, b_n \text{ such that } ((a_1, \ldots, a_m), (b_1, \ldots, b_n)) \in A_\omega \text{ and } b \in \{b_1, \ldots, b_n\}\}.$$

$Immac_A$ operates on a set $x$ of elements and produces the set of elements that are accessible by precisely one application of an operation of $A$, starting within $x$. $Immac_A$ is not necessarily an increasing function, but $I_U + Immac_A$ is, since it cumulates the argument set onto the result set.

*Definition.* A set $x$ is a **subalgebra** of $A$ if it is $Immac_A$-reduced. The intersection of two, or indeed any set of subalgebras is a subalgebra, hence we may define:

*Definition.* $Ac_A = J(Immac_A)$.

In particular $Ac_A\phi$ is the minimal subalgebra of $A$. It is also possible to consider $Ac_A x$ as the minimal subalgebra of an algebra obtained from $A$ by adding nullaries.

*Definition.* For a correspondence $B$ whose range is elements and whose domain is disjoint from the operators of $A$, $Nulls_B\, A$ is the algebra obtained from $A$ by adding nullary operations, one for each member $b$ of the domain of $B$, and having as result(s) the unit-strings comprising the elements indexed by $b$.

So $\quad Ac_A x = Ac_{Nulls_{I(x)}A}\phi$

The minimal subalgebra $Ac_A\,\phi$ is not merely $Immac_A$-reduced; it is a fixed point of $Immac_A$. If $Immac_A$ is increasing, then every subalgebra is a fixed point. But not in general otherwise. However $Ac_A x$ does satisfy

$$Immac_{Nulls_{I(x)}A}\, x = x$$

Since all the operations of $A$ are finitary, $Immac_A$ is continuous.

If $Immac_A$ is additive, in particular if $A$ is unary or if $Immac_A$ is increasing, we have

$$Immac_A^{\dagger} = Ac_A.$$

Moreover for any $A$

$$Immac_A^{\dagger}\phi = Ac_A\phi.$$

But in general the sequence

$$x, Immac_A x, Immac_A^2 x, \ldots$$

is not increasing and does not exhaust the elements accessible from $x$, since it imposes precise contemporaneity on the intermediate results used at each stage. If, for example, the identity correspondence is among the operations of $A$, then this condition is powerless, since $I_U + Immac_A$ is increasing. But in general we may find there are members of $Immac_A(x \cup Immac_A x)$ that belong neither to $Immac_A x$ nor to $Immac_A^2 x$. However:

*Theorem*
For any $A$, $(I_U+Immac_A)^\dagger = Ac_A$. Since $Immac_A^\dagger\phi = Ac_A\phi$,

$$Ac_A x = Ac_{Nulls I_{(x)A}}\phi = Immac^\dagger_{Nulls_{I(x)A}}\phi = (Kx+Immac_A)^\dagger\phi$$

## TEXTUAL PRESENTATION OF AN ALGEBRA – EQUATION SETS

Consider two algebras $A$ and $B$. $A_\omega$ is the operation indexed by $\omega$ in the algebra $A$; similarly $B_\omega$. If $S \in A \longrightarrow\!\!\!\!\times\ B$ we loosely use $Sa$ for $Im_s\, a$.

*Definition.* An $S \in A \longrightarrow\!\!\!\!\times\ B$ is **consistent** if the following condition holds: Given any $\omega$-polyedge

$$((a_1, \ldots, a_q), (a'_1, \ldots, a'_r)) \in A^q \times A^r$$

of $A$, and any $b_1 \in Sa_1$, $b_2 \in Sa_2, \ldots, b_q \in Sa_q$, then

$$B_\omega(b_1, \ldots, b_q) \subseteq Sa'_1 \times \ldots \times Sa'_r \tag{1}$$

I.e. $$Sa'_1 \times Sa'_2 \times \ldots \times Sa'_r \supseteq Im_{B\omega}(S_{a_1} \times \ldots \times S_{a_q}) \tag{2}$$

I.e. any assignment to $a$ gets transformed into an assignment to $a.'$

There is one such condition for each operation-instance in $A$, and each condition can be split up into $r$ conditions

$$S_{a'_1} \supseteq Im_{1st\ B\omega}(S_{a_1} \times \ldots \times S_{a_q})$$
$$S_{a'_2} \supseteq Im_{2nd\ B\omega}(S_{a_1} \times \ldots \times S_{a_q})$$
$$\ldots \text{etc.} \tag{3}$$

These can be collected in accordance with the $A$-element on the l.h.s. to give one inequation for each $A$-element:

$$S_{a'_1} \supseteq Im_{1st\ B\omega}(S_{a_1} \times \ldots \times S_{a_q}) \cup \text{other similar terms.} \tag{4}$$

The r.h.s. has one contribution for each arrow pointing to the l.h.s. variable $a_1$ in the polygraph of $A$.

*Theorem*
The consistent assignments are just the subalgebras of $A \times B$.

*Proof.* $S$ is a subalgebra iff

$$(a_1, b_1) \in S \wedge (a_2, b_2) \in S \wedge \ldots \wedge (a_q, b_q) \in S \wedge$$
$$(((a_1, b_1), (a_2, b_2), \ldots, (a_q, b_q)), ((a'_1, b'_1), \ldots, (a'_r, b'_r))) \in (A \times B)_\omega$$

implies $(a'_1, b'_1) \in S \ldots (a'_r, b'_r) \in S$,

i.e., iff

$$b \in \underset{i}{\times} Sa_i \wedge (a, a') \in A_\omega \wedge (b, b') \in B_\omega \Rightarrow b' \in \underset{j}{\times} Sa'_j$$

i.e., iff

$$(a, a') \in A_\omega \Rightarrow b \in \underset{i}{\times} Sa_i \Rightarrow B_\omega b \subseteq \underset{j}{\times} Sa'_j$$

which is the condition that $S$ is consistent.

*Corollary* 1. An assignment $S$ is consistent iff it satisfies

$$S \supseteq Ac_{A \times B} S$$

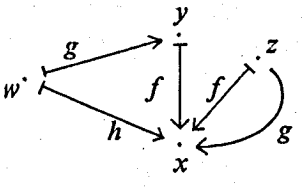
*Corollary* 2. The execution of $A$ on $B$ reaches the instruction $a \in A$ iff

$$Im(Ac_{A \times B}\phi)\{a\} \neq \phi.$$

The execution of $A$ never reaches $a$, for any $B$, iff for all algebras $B$

$$Im(Ac_{A \times B}\phi)\{a\} = \phi.$$

*Example* 1. Consider a transition diagram $A$ and an interpretation $B$ of its labels as state-transformations. We seek to assign to each node of $A$ a set of states, and to do so in a way that is 'consistent' with the intuitive idea of 'executing' $A$ relative to $B$. What are these consistency conditions? E.g., given a fragment including 4 nodes $x, y, z, w$, as shown in the diagram, the sets




$S_x$, $S_y$, $S_z$, $S_w$ assigned to them must satisfy the 5 conditions

$$S_x \supseteq B_f S_y, \quad S_x \supseteq B_f S_z, \quad S_x \supseteq B_g S_z, \quad S_x \supseteq B_h S_w, \quad S_y \supseteq B_g S_w.$$

I.e., collecting by target and within target by label:

$$S_x \supseteq B_f(S_y \cup S_z) \cup B_g S_z \cup B_h S_w$$

$$S_y \supseteq B_g S_w$$

Note that $y, z \in A_f^{-1}x$, $z \in A_g^{-1}x$, $w \in A_h^{-1}x$ and $w \in A_g^{-1}y$.

So, generalizing to an arbitrary element $x$ of an arbitrary unary algebra $A$, we must write for each node $x$ an inequation whose r.h.s. has a contribution for every labelled edge that points to $x$:

$$S_x \supseteq \bigcup_{\text{all } f} B_f(S(A_f^{-1}x)).$$

I.e., (remembering that $S$ is a nondeterminate function from the elements of $A$ to the elements of $B$), we have

$$S \supseteq B_f \cdot S \cdot A_f^{-1} \text{ (for all } f)$$

which (as suggested by the above theorem) is only a small rearrangement of the necessary and sufficient condition that $S$ is a *subalgebra* of $A \times B$.

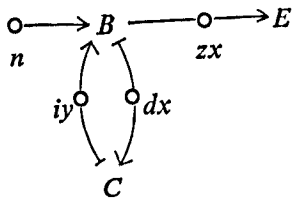
*Definition.* The **equation set** of an algebra $A$ is a set of equations in variables $S_k$, one for each element of $A$, with the form

$$S_a = Im_f(S_{a'} \times S_{a''} \times \ldots \times S_{a(q)}) \cup \ldots$$

other similar terms, one for each polyedge pointing to $a$, with $f$ being the 'untupled' operator that produces $a$, and $a', a'', \ldots$ being the source-vector of the polyedge.

*Example* 2. Consider the following algebra (taken from Example 3 of the

introduction) where $n$ is a nullary operator (e.g., its interpretation might be to produce the pair 2, 1).




The equation set is

$$S_B = Im_n(\ ) \cup Im_{iy}(S_C)$$
$$S_C = Im_{dx}(S_B)$$
$$S_E = Im_{zx}(S_B).$$

Note that an algebra element that is not in the range of any operation gives rise to an equation with a void r.h.s. Dually one that is not in the domain of any operation gives rise to a variable that has no r.h.s. occurrences.

*Lemma*

In an interpretation $B$ an assignment $S \in A \longrightarrow\!\!\!\!\!\times\ B$ satisfies the equation-set of $A$ if it satisfies $S = Ac_{A \times B} S$.

*Theorem*

There is a unique minimal consistent assignment and it is

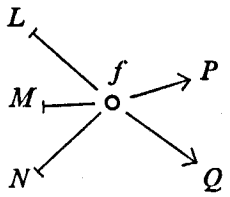
(a) the intersection of all consistent assignments.

(b) the unique minimal solution of the equation set of $A$, when interpreted in $B$.

*Example* 3. Further specializing Example 1, let the interpretation $B$ be the 'free' algebra for the algebra-type; i.e., let it be the algebra of operator-strings with the unary operations for adjoining single letters. Then the minimal consistent assignment gives the finite paths that reach each $a \in A$. This is the interpretation that 'remembers' the entire derivation of an element.

## TEXTUAL PRESENTATION OF AN ALGEBRA – POLYPROGRAMMING*

We describe here how to *write* an algebra. The rules specialize in the unary case to something not unlike conventional programming using labels and jumps. Suppose we are given a polygraph *and* some distinct labelling of its nodes. Then corresponding to each polyedge




* For some years I have aspired to 'language-free programming'. The present section brings this goal appreciably nearer. Abstract syntax is polygraphs not trees.

the following piece of text is called the **polystatement for** the edge:

$$(L, M, N): f; \textbf{goto}(P, Q) \tag{1}$$

An alternative way of writing it is

$$\textbf{goto}(P, Q)(f(L, M, N)) \tag{2}$$

which is called the **expression for** the edge.

Lastly, there is

$$(P, Q) \supseteq f(L, M, N) \tag{3}$$

which is the **inequation for** the edge.

A programmer's label $L$ can be considered either as the definiendum of a certain operation, to be applied wherever '**goto** $L$' is encountered, or as the definiendum of a certain intermediate result to be operated on wherever '$L$:' is encountered. The polystatement (1) emphasizes the operation aspect of both sources and targets. The inequation (3) emphasizes their intermediate results. The expression (2) refers to its sources as intermediate results and its targets as operations. These emphases are substantiated when we attempt to elide a label by substitutive elimination, as discussed below.

A whole polygraph can be textually presented as an *unordered* collection of polystatements, expressions, and inequations, arbitrarily mixed. Such a collection will be called a **polyprogram** for the algebra. The rest of this section consists of a number of informal observations of correspondences between algebras and some familiar and less familiar programming notations. They suggest various ways in which the above definition of 'polyprogram' might be 'sugared' to include things that come more naturally from a programmer's pen, as well as some generalizations of them.

In the unary case we have:

| *polygraph* | *statement* | *expression* | *inequation* |
|---|---|---|---|
| $L \xrightarrow{f} M$ | $L: f$; **goto** $M$ | **goto** $M(f(L))$ | $M \supseteq fL$ |

Now consider the elision of an intermediate label $M$ in

$$L \xrightarrow{f} M \qquad L: f; \textbf{goto}\ M \qquad \textbf{goto}\ M(f(L)) \qquad M = fL \tag{4}$$

$$M \xrightarrow{g} N \qquad M: g; \textbf{goto}\ N \qquad \textbf{goto}\ N(g(M)) \qquad N = gM \tag{5}$$

From the statements we get

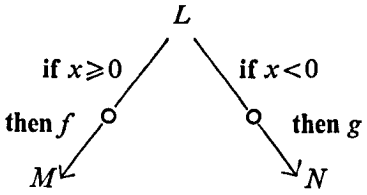
$$L: f; g; \textbf{goto}\ N$$

by substituting from (5) into (4) *provided* that (5) is the only statement having $M$ as its label. From the inequations we get $N = g(fL)$ by substituting from (4) into (5) *provided* that (4) is the only inequation having $M$ as its definiendum, i.e., it is a definition. From the expressions we get

$$\textbf{goto}\ N(g(f(L)))$$

provided that *both* conditions are met, and it is thus not obvious which of (4) and (5) is the host in the substitution.

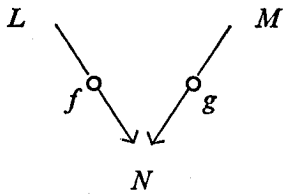
The possibilities for substitutive elimination are increased if the edges having a common source or target can be collected together. It is natural to extend statements so that edges can be collected by source, e.g., as shown in the diagram.




**L: if $x \geqslant 0$ then $f$; goto $M$**
**and if $x<0$ then $g$; goto $N$**

There is no obvious generalization of this to the case of poly-source operations.

Dually the definition notation leads naturally to collecting the edges having a common target, e.g.,

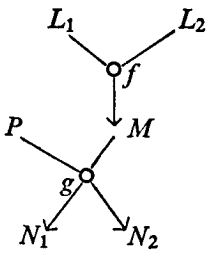



$N=f(L)$ **or** $g(M)$

Again there is no obvious extension here to polytarget operations.

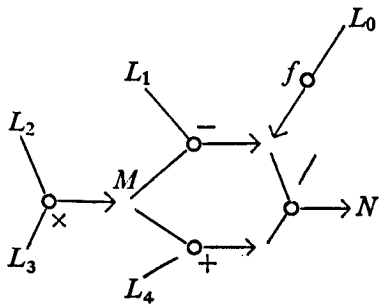
The extent to which labels can be eliminated is of course limited to including a 'cycle-breaking' subset. This is true in both definitions and statements.

In the polyadic case of substitutive elimination we have, e.g.:




$(N_1, N_2) =$

$g(P, f(L_1, L_2))$



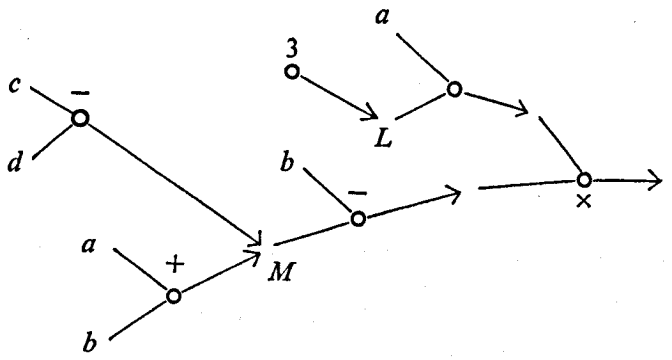

$N=(f(L_0)$ **or** $(L_1-M))/(M+L_4)$

$M=L_2 \times L_3$

So far there has been no mixing of the three formats, definitions, expressions and statements. But it seems natural to mix them; e.g., a fragment such as

$$(a+:L)\times(b-:M(c/d))\ldots \textbf{goto}\ L(3)\ldots \textbf{goto}\ M(a+b)\ldots$$

can obviously be attributed to the following fragment of polygraph:




In conventional programming a labelled statement can be approached either by jumping or by natural sequencing (ignoring its label). We have here a polyadic generalization of this phenomenon, e.g., at label $M$ above. It is perhaps worth noting that this generalization of jumping is precisely in accord with the author's $J$ operator (Landin, 1965, 1965*a*) in those cases where a comparison is possible. ($J$ generalizes ordinary ALGOL 60 jumping in two directions: the first is relaxing scope restrictions, and the second permitting arguments and/or results to be communicated. It is the latter that is relevant here.)

This section has given some precise ways of writing an algebra, together with some imprecise suggestions for making it more palatable.

## EXECUTING ONE ALGEBRA ON ANOTHER

This section defines the 'operation $F_BA$ computed by' a program-algebra $A$ on a machine-algebra $B$. The main result of this paper is the following: There is a structure over the set of algebras such that (i) any finite algebra can be expressed in terms of 'elementary' algebras. (ii) The function $F_B$ respects this structure; i.e., the operation computed by a program can be expressed systematically in terms of the operations computed by its 'parts'.

The proof of this result is in the following section.

*Recall.* $A \rightarrow\!\!\!\!\times\ B$ is the set of partial nondeterminate functions from $A$ to $B$. We use this notation loosely with $A$ and $B$ denoting algebras. For example, $f \in A \rightarrow\!\!\!\!\times\ B$ is a concise way of remarking that the correspondence $f$ will be used in 'post-composition', $f \cdot A_\omega$, with the operations of $A$, and in 'pre-composition', $B_\omega \cdot f$, with the operations of $B$.

*Definition.* $A \tilde{\rightarrow} B$ is the set of partial determinate functions from $A$ to $B$.

*Definition.* The **computation** $Comp_x(A, B)$ of $A$ on $B$ **generated by** $x \in A \longrightarrow\!\!\!\!\times B$ is the subalgebra $Ac_{A\times B}(x)$ of $A \times B$ generated by $x$.

For relating an arbitrary subalgebra of a direct product to a minimal subalgebra we use the following

*Lemma*

$$Ac(Nulls_x A \times Nulls_y B)\phi = Ac_{A\times B}(y \cdot x^{-1}).$$

*Corollary* 1. Any subalgebra of $A \times B$ is a minimal subalgebra of the product of nullary extensions of $A$ and $B$ (indeed in many different ways). In particular

$$Ac_{A\times B}(x) = Ac(Nulls_{I_A} A \times Nulls_x B).$$

*Corollary* 2. A computation generated by a set $x$ is also a 'spontaneous' computation (i.e., one generated by the void set). In fact

$$Comp_x(A, B) = Comp_\phi(Nulls_{I_A} A, Nulls_x B).$$

The next corollary will be needed when we are interested in what can be done by algebras that are determinate.

*Corollary* 3. The nullary extensions for the above lemma can be done with *determinate* nullaries.

*Proof.* Any correspondence $x$ can be factored as the product of a function and an inverse function.

*Terminology.* If $(a, b) \in Comp_\phi(A, B)$ we say that the instruction $a$ **attains** the state $b$; also that the state $b$ **accepts** the instruction $a$. This conforms with familiar usage of 'accept' since one possibility for the 'program' $A$ is the algebra of all strings under the unary affixing operations.

*Definition.* The **set generated** at the instruction $a \in A$ is the set of states attained at $a$, i.e., $Im_{(Ac(A\times B)\phi}\{a\}$. The **set recognized** at the state $b \in B$ is the set of instructions accepted by $b$, i.e., $Im_{Ac(A\times B)^{-1}\phi}\{b\}$. We generalize these to allow for unions of recognized sets:

*Definition.* The **result** $Gen(R, C)$ **generated** at the 'exit map' $R \in r \longrightarrow\!\!\!\!\times A$ by a computation $C \subseteq A \times B$ is $C \cdot R$, i.e., a set of union-sets of states indexed by $r$. For example, (i) if $r$ is a singleton this corresponds to an (unordered) set of terminal instructions. (ii) if $R$ is a singleton it reduces to the 'set generated by' as defined above.

The **argument** $Recog(R, C)$ **recognized at** the 'output map' $R \in r \longrightarrow\!\!\!\!\times B$ by a computation $C \subseteq A \times B$ is $C^{-1} \cdot R$, i.e., a set of union-sets of instructions indexed by $r$.

*Definition.* The **operation** $F_B A$ **computed by** algebra $A$ on $B$ is $\lambda x \,.\, Gen(I_A, Comp_x(A, B))$. So $F_B\, A$ transforms a nondeterminate assignment $x$ that assigns zero or more states to each instruction into another such nondeterminate assignment.

*Lemma*

$$F_B A x = Ac_{A\times B}(x); \text{ i.e., } F_B A = Ac(A \times B).$$

*Example* 1. Let $A$ be the sphere and $B$ the plane in our ball-and-plane machine. Then if no choices arise the set of vertex-pairings that ensue from the initial

pair $(a, b)$ is $Comp_{\{(a,b)\}}(A, B)$. The case of a Karp schema is essentially the same. Note that the 'no choice' condition is precisely that the computation is a *monogenic* algebra.

*Example* 2. Let $A$ be a labelled tree (i.e., what logicians call a term) and $B$ be a (determinate) interpretation for the operators. Then $Comp_\phi(A, B)$ associates the denotation with each vertex. In the particular case of unary operators this tree is a string of symbols taken from some input alphabet and $B$ is a transition diagram. Then $Comp_\phi(A, B)$ associates with each symbol the resulting machine state. The essential property here is that $A$ is monotectonic and $B$ total determinate. Under these conditions the computation is the same 'shape' as the program.

*Example* 3. Let $A$ be a 'free' algebra of labelled trees, and $B$ an interpretation as in Example 2. Then $Comp_\phi(A, B)$ again associates the denotation with each tree. So the set recognized by a state $b$ is the set of terms that denote $b$. In the unary case this algebra of trees is the algebra of strings under the unary operations of adjoining single letters. Then the set recognized by a state $b$ is the set of operator-sequences that reach $b$. In the case that $A$ and $B$ are finite this set is regular (Kleene's theorem).

*Example* 4. Corresponding to a context-free grammar there is an algebra $A$ over the non-terminal symbols and one $B$ over the terminal strings, such that $Gen(I_A, Comp_\phi(A, B))$ is the languages generated by the non-terminals.

*Example* 5. The associativity of direct-product can be used to relate a machine with hardware stack to a machine in which the stack is programmed. Suppose $A$ is an algebra in which each operation $A_\omega$ has some fixed number, $arity(\omega)$, of arguments, and one result. Define three related algebras as follows:

$$\text{Stack}(\omega, m) = \lambda n \,.\, \textbf{if } n = m \textbf{ then } n - arity(\omega) + 1$$
(otherwise undefined)

$$\text{IBM704}(\omega, m) = \lambda q \,.\, \textbf{let args} = q_{m-1}, q_{m-2}, \ldots, q_{m-arity(\omega)}$$
$$assign(A_\omega(\text{args}), m - arity(\omega), q)$$
(where *assign* is as in Example 6 below)

$$\text{B5000} = \text{IBM704} \times \text{Stack}.$$

Then a program-algebra $P$ does 'the same thing' on B5000 that $P \times$ Stack does on IBM704.

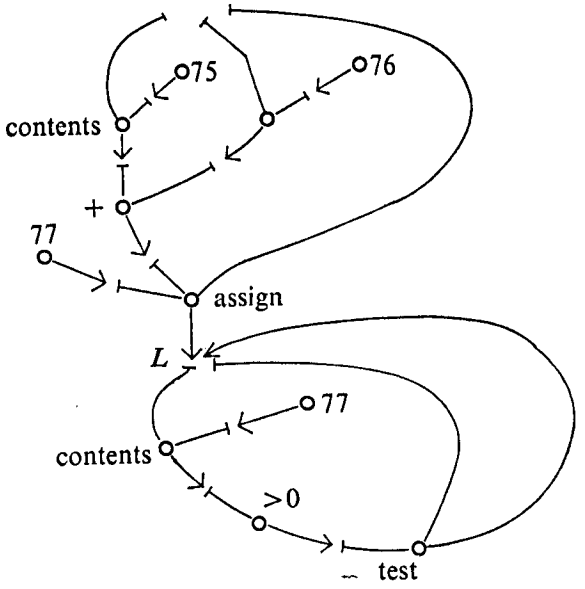
*Example* 6. Let $B$ be an algebra over $A \cup \{\text{true}, \text{false}\} \cup I \cup C^I$ where $I$ can be thought of as a set of addresses of cells in memory and $C$ can be thought of as the words that can occupy them. Let the operations of $B$ be some functions and relations over $C$, together with the following three:

$$contents = (I \times A^I \to A)\text{: } \lambda(i, q) \,.\, qi$$
$$assign = (A \times I \times A^I \to A^I)\text{:}$$
$$\lambda(a, i, q)j \,.\, \textbf{if } j = i \textbf{ then } a \textbf{ else } qj$$
$$test = (\{\text{true}, \text{false}\} \times A^I \to A^I)\text{:}$$
$$\lambda(p, q) \,.\, \textbf{if } p \textbf{ then } q \text{ (and otherwise undefined)}.$$

Then a flow diagram of assignments, tests and jumps can be viewed as an algebra $A$ of the same type. E.g.,

77: = *contents*(75) + *contents*(76); *L*: **if** *contents*(75) > 0 **then goto** *L*;

The corresponding algebra can best be observed as a polygraph, as shown in the diagram.




*Example* 7. Let the carrier of $B$ be $I \cup C \cup (C^I \times C^N)$ where $C$ is words as in Example 6, and $N$ is the positive integers. Then $Q = C^I \times C^N$ is the state-set of a memory + stack machine. $B$ has the obvious stack operations ($Q \rightleftarrows Q$), and also three special operations

$$\begin{aligned} \mathit{contents} &= (I \times Q \to Q)\colon \lambda(i,(q,s)) \,.\, (q, \mathrm{cons}(qi, s)) \\ \mathit{assign} &= (I \times Q \to Q)\colon \\ &\quad \lambda(i,(q,s)) \,.\, (\lambda j \,.\, \textbf{if } j = i \textbf{ then } \mathrm{head}\ s \textbf{ else } qj,\ \mathrm{tail}\ s) \\ \mathit{test} &= (Q \rightleftarrows Q)\colon \lambda(q,s) \,.\, \textbf{if } \mathrm{head}\ s \textbf{ then } (q, \mathrm{tail}\ s). \end{aligned}$$

Then a program $A$ is a flow-diagram of elementary operations.

### Superposition of algebras

*Definition.* The **superposition** of two algebras $A$, $B$ is their sum $A+B$. In terms of polygraphs this means drawing them on transparencies with vertices identified by $(x, y)$-coordinates, and then superposing the transparencies. Thus, for example, any finite algebra has a 'patchwise' construction as the superposition of a set of 'elementary' algebras, i.e., algebras each having just one polyedge.

Immediate accessibility in $A+B$ is given by

$$\mathit{Immac}_{A+B} = \mathit{Immac}_A + \mathit{Immac}_B.$$

Hence the subalgebras of $A+B$ are precisely those common to $A$ and $B$ (using the inequation that characterizes subalgebras).

*Theorem*

$$Ac_{A+B}=(Ac_A \cdot Ac_B)^\dagger$$

(Intuitively, any derivation in $A+B$ consists of alternate bursts of invoking the operations of $A$ and the operations of $B$.)

*Proof.*
$$\begin{aligned} Ac_{A+B} &= (I_U + Immac_A + Immac_B)^\dagger \\ &= ((I_U + Immac_A)^\dagger \cdot (I_U + Immac_B)^\dagger)^\dagger \\ &= (Ac_A \cdot Ac_B)^\dagger. \end{aligned}$$

*Theorem*

$$A \times (B+C) = A \times B + A \times C.$$

Hence the main result:

*Theorem*

$$F_A(B+C) = (F_A B \cdot F_A C)^\dagger$$

*Proof.*
$$\begin{aligned} Ac(A \times (B+C)) &= Ac(A \times B + A \times C) \\ &= (Ac_{A \times B} \cdot Ac_{A \times B})^\dagger \\ &= (F_A B \cdot F_A C)^\dagger. \end{aligned}$$

So, if we take for structure over the operations the function $\lambda(f, g) \,.\, (f \cdot g)^\dagger$ the function $F_A$ maps homomorphically from algebras-under-superposition to operations.

**Acknowledgements**

I have been helped greatly by talking to Drs H. Bekič, M. Bird and R.M. Burstall.

## REFERENCES

Burstall, R.M. & Landin, P.J. (1969) Programs and their proofs: an algebraic approach. *Machine Intelligence 4*, pp. 17–43 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Cohn, P. M. (1965) *Universal Algebra*. New York & London: Harper Row.

Karp, R.M. (1959) Some Applications of Logical Syntax to Digital Computer Programming. Harvard University Thesis.

Landin, P.J. (1965) A Correspondence between ALGOL 60 and Church's Lambda-Notation. *Comm. Ass. comput. Mach.*, **3**, 89–101 and 158–165.

Landin, P.J. (1965a) Generalization of jumps and labels – Univac Systems Programming Research Report Aug. 1965.

Minsky, M. (1967) *Computation: finite and infinite machines*. New York: Prentice-Hall.

Thatcher, J.W. & Wright, J.B. (1968) Generalized finite automata theory with an application to a decision-problem of second-order logic. *Math. System Theory*, **2**, 1.

# MECHANIZED REASONING

# 6

# A Note on Mechanizing Higher Order Logic

J. A. Robinson
College of Liberal Arts
Syracuse University

§1. Any description of a function in the lambda-calculus notation can be translated automatically into a description of the same function in a purely applicative notation which makes no use whatever of abstraction or bound variables.

For example, the function $\sqrt{[x(x+1)]}$ might be described as

$$\lambda x(\text{SQRT}((\text{TIMES } x)((\text{PLUS } x)\text{ONE}))). \tag{1}$$

But by making use of 'Schönfinkel's functions' A, K, and I, defined by

$$(((\text{A}x)y)z) = ((xz)(yz)) \tag{2}$$

$$((\text{K}x)y) = x \tag{3}$$

$$(\text{I}x) = x \tag{4}$$

we can describe $\sqrt{[x(x+1)]}$ as

$$((\text{A}(\text{K SQRT}))((\text{A}((\text{A}(\text{K TIMES}))\text{I}))((\text{A}((\text{A}(\text{K PLUS}))\text{I}))(\text{K ONE})))). \tag{5}$$

The description (5) contains no bound variables or abstraction operations. It is a purely applicative combination of the objects A, K, I, SQRT, TIMES, PLUS, and ONE. Of course, it is not at all obvious that (5) and (1) describe the same function. In order to prove that they do, however, it is necessary only to check that the result of applying (5) to an arbitrary object $z$ is the same as the result of applying (1) to $z$. In fact, we have

$$(\lambda x(\text{SQRT}((\text{TIMES } x)((\text{PLUS } x)\text{ONE})))z)$$
$$= (\text{SQRT}((\text{TIMES } z)((\text{PLUS } z)\text{ONE})))$$

when (1) is applied to $z$. When (5) is applied to $z$, we have

$$(((\text{A}(\text{K SQRT}))((\text{A}((\text{A}(\text{K TIMES}))\text{I}))((\text{A}((\text{A}(\text{K PLUS}))\text{I}))(\text{K ONE}))))z)$$
$$= (((\text{K SQRT})z)(((\text{A}((\text{A}(\text{K TIMES}))\text{I}))((\text{A}((\text{A}(\text{K PLUS}))\text{I}))(\text{K ONE})))z))$$
by (2);

$=$(SQRT(((A((A(K TIMES))I))((A((A(K PLUS))I))(K ONE)))$z$))
by (3);
$=$(SQRT((((A(K TIMES))I)$z$)(((A((A(K PLUS))I))(K ONE))$z$)))
by (2);
$=$(SQRT((((K TIMES)$z$)(I$z$))(((A((A(K PLUS))I))(K ONE))$z$)))
by (2);
$=$(SQRT((TIMES(I$z$))(((A((A(K PLUS))I))(K ONE))$z$)))
by (3);
$=$(SQRT((TIMES $z$)(((A((A(K PLUS))I))(K ONE))$z$)))
by (4);
$=$(SQRT((TIMES $z$)((((A(K PLUS))I)$z$)((K ONE)$z$))))
by (2);
$=$(SQRT((TIMES $z$)((((K PLUS)$z$)(I$z$))((K ONE)$z$))))
by (2);
$=$(SQRT((TIMES $z$)((PLUS(I$z$))((K ONE)$z$))))
by (3);
$=$(SQRT((TIMES $z$)((PLUS $z$)((K ONE)$z$))))
by (4);
$=$(SQRT((TIMES $z$)((PLUS $z$)ONE)))
by (3). Therefore (5) describes the same function as (1).

It seems most unlikely that one could in general write purely applicative 'Schönfinkel descriptions', like (5), of functions already known to one in some other form. Fortunately there is a general procedure – the Schönfinkel procedure – which, when applied to any expression written in the more intuitive lambda-calculus notation, will produce a correct translation of it into the Schönfinkel notation. The procedure consists simply of repeatedly applying the three following rules until no further applications can be made:

replace $\lambda xx$ by I; (6)

replace $\lambda x$B by (KB), provided the expression B contains no occurrences of $x$; (7)

replace $\lambda x$(BC) by ((A$\lambda x$B)$\lambda x$C) if (BC) contains one or more occurrences of $x$. (8)

For example, this procedure translates (1) into (5). We have:

(1) $=\lambda x$(SQRT((TIMES $x$)((PLUS $x$)ONE)));
$=$((A $\lambda x$ SQRT)$\lambda x$((TIMES $x$)((PLUS $x$)ONE)))
by (8);
$=$((A(K SQRT))$\lambda x$((TIMES $x$)((PLUS $x$)ONE)))
by (7);
$=$((A(K SQRT))((A $\lambda x$(TIMES $x$))$\lambda x$((PLUS $x$)ONE)))
by (8);
$=$((A(K SQRT))((A((A $\lambda x$ TIMES)$\lambda xx$))$\lambda x$((PLUS $x$)ONE)))
by (8);
$=$((A(K SQRT))((A((A(K TIMES))$\lambda xx$))$\lambda x$((PLUS $x$)ONE)))
by (7);
$=$((A(K SQRT))((A((A(K TIMES))I))$\lambda x$((PLUS $x$)ONE)))
by (6);
$=$((A(K SQRT))((A((A(K TIMES))I))((A $\lambda x$(PLUS $x$))$\lambda x$ ONE)))
by (8);

$=((A(K\ SQRT))((A((A(K\ TIMES))I))((A((A\ \lambda x\ PLUS)\lambda xx))\lambda x\ ONE)))$
by (8);
$=((A(K\ SQRT))((A((A(K\ TIMES))I))((A((A(K\ PLUS))\lambda xx))\lambda x\ ONE)))$
by (7);
$=((A(K\ SQRT))((A((A(K\ TIMES))I))((A((A(K\ PLUS))I))\lambda x\ ONE)))$
by (6);
$=((A(K\ SQRT))((A((A(K\ TIMES))I))((A((A(K\ PLUS))I))(K\ ONE))))$
by (7);
$=(5)$.

We saw previously, by a special calculation, that (1) and (5) describe the same function. More generally, whenever the Schönfinkel procedure produces a translation F of an expression $\lambda x B$, we may prove that

$$(\lambda x B z) = (F z) \tag{9}$$

holds for an arbitrary $z$ by noting that:

(i) if B is $x$ then $(\lambda x B z) = z$
$= (I z)$ by (4)
$= (F z)$ by (6).

(ii) if B does not contain $x$ then $(\lambda x B z) = z$
$= ((K B) z)$ by (3)
$= (F z)$ by (7).

(iii) if $B = (C D)$ contains $x$ then $(\lambda x B z) = (\lambda x (C D) z)$
$= (C\{z/x\} D\{z/x\})$
$= ((\lambda x C z)(\lambda x D z))$
$= (((A \lambda x C)\lambda x D) z)$ by (2)
$= (F z)$ by (8).

§2. The phenomena described in the previous section were discovered fifty years ago by Schönfinkel (1924). They have since been studied in great detail by Curry and Feys (1958) under the heading *combinatory logic.* An excellent summary can be found in Rosenbloom (1950, Chapter 3, section 4).

For our present purposes the situation can be summed up by saying that whatever can be expressed in a language based on application *and* abstraction as fundamental notions can be expressed in a far simpler language *based on application alone.*

The purpose of this note is to provoke investigations into the use of such simple 'Schönfinkel languages' as the vehicles of meaning in the mechanization of higher order theorem-proving problems. The extreme syntactic and semantic simplicity of these languages makes it seem likely that it will be much easier to develop proof procedures and interactive deductive systems for them than for the lambda-calculi suggested in Robinson (1969). For one thing, they can be treated as first-order equation calculi, permitting the use of resolution and related inference principles (such as paramodulation) which are well understood.

These possibilities are explored in the next section.

§3. Such a first-order system might look like the following. The *terms* are either *variables* $x, y, z$, etc., or *constants* A, B, C, PLUS, SQRT, etc., or else are *applications* $(\alpha, \beta)$ in which $\alpha$ and $\beta$ are terms.

The only *literals* in the system are equations $\alpha = \beta$ and inequations $\alpha \neq \beta$, where $\alpha$ and $\beta$ are terms.

One makes assertions in the system by writing *clauses*, i.e., finite collections of literals considered as disjunctions of their members, universally quantified with respect to all variables.

In other words, this is a first-order language in which there is only one relation symbol, namely equality; only one function symbol, namely application; and a collection of individual constants.

Among the constants there will be: A, K, I, TRUE, FALSE, NOT, OR, AND, IMPLIES, EQUAL, ALL, EXISTS, CHOICE, IF and FAIL. The first three of these will intuitively denote the three Schönfinkel functions; the next seven will denote what their mnemonic properties suggest; the next two will denote the quantifiers; CHOICE will denote Hilbert's epsilon operator or selection function, which assigns to every nonempty set some member of it (and, to the empty set, some arbitrary object); IF denotes the operator which corresponds to the 'if ... then ... else ...' construction of conditional expressions in languages like ALGOL, LISP or POP-2; FAIL denotes the dual of CHOICE.

These informal and intuitive *semantic* rules for the constants are embodied in a set of clauses called SEM, which plays the role of a set of axioms. They are:

SEM 1. $(((\mathrm{A}x)y)z) = ((xz)(yz))$
SEM 2. $((\mathrm{K}x)y) = x$
SEM 3. $(\mathrm{I}x) = x$
SEM 4. $x \neq \mathrm{TRUE}\ x \neq \mathrm{FALSE}$
SEM 5. $x \neq \mathrm{TRUE}\ (\mathrm{NOT}\ x) = \mathrm{FALSE}$
SEM 6. $x = \mathrm{TRUE}\ (\mathrm{NOT}\ x) \neq \mathrm{FALSE}$
SEM 7. $x \neq \mathrm{FALSE}\ (\mathrm{NOT}\ x) = \mathrm{TRUE}$
SEM 8. $x = \mathrm{FALSE}\ (\mathrm{NOT}\ x) \neq \mathrm{TRUE}$
SEM 9. $x \neq \mathrm{TRUE}\ y \neq \mathrm{TRUE}\ ((\mathrm{AND}\ x)y) = \mathrm{TRUE}$
SEM 10. $x \neq \mathrm{TRUE}\ y \neq \mathrm{FALSE}\ ((\mathrm{AND}\ x)y) = \mathrm{FALSE}$
SEM 11. $x \neq \mathrm{FALSE}\ y \neq \mathrm{TRUE}\ ((\mathrm{AND}\ x)y) = \mathrm{FALSE}$
SEM 12. $x \neq \mathrm{FALSE}\ y \neq \mathrm{FALSE}\ ((\mathrm{AND}\ x)y) = \mathrm{FALSE}$
SEM 13. $((\mathrm{AND}\ x)y) \neq \mathrm{TRUE}\ x = \mathrm{TRUE}$
SEM 14. $((\mathrm{AND}\ x)y) \neq \mathrm{TRUE}\ y = \mathrm{TRUE}$
SEM 15. $((\mathrm{AND}\ x)y) \neq \mathrm{FALSE}\ x = \mathrm{FALSE}\ y = \mathrm{FALSE}$
SEM 16. $x \neq \mathrm{TRUE}\ y \neq \mathrm{TRUE}\ ((\mathrm{OR}\ x)y) = \mathrm{TRUE}$
SEM 17. $x \neq \mathrm{TRUE}\ y \neq \mathrm{FALSE}\ ((\mathrm{OR}\ x)y) = \mathrm{TRUE}$
SEM 18. $x \neq \mathrm{FALSE}\ y \neq \mathrm{TRUE}\ ((\mathrm{OR}\ x)y) = \mathrm{TRUE}$
SEM 19. $x \neq \mathrm{FALSE}\ y \neq \mathrm{FALSE}\ ((\mathrm{OR}\ x)y) = \mathrm{FALSE}$
SEM 20. $((\mathrm{OR}\ x)y) \neq \mathrm{FALSE}\ x = \mathrm{FALSE}$
SEM 21. $((\mathrm{OR}\ x)y) \neq \mathrm{FALSE}\ y = \mathrm{FALSE}$
SEM 22. $((\mathrm{OR}\ x)y) \neq \mathrm{TRUE}\ x = \mathrm{TRUE}\ y = \mathrm{TRUE}$
SEM 23. $x \neq \mathrm{TRUE}\ y \neq \mathrm{TRUE}\ ((\mathrm{IMPLIES}\ x)y) = \mathrm{TRUE}$
SEM 24. $x \neq \mathrm{TRUE}\ y \neq \mathrm{FALSE}\ ((\mathrm{IMPLIES}\ x)y) = \mathrm{FALSE}$

SEM 25. $x \neq \text{FALSE}\ y \neq \text{TRUE}\ ((\text{IMPLIES}\ x)y) = \text{TRUE}$
SEM 26. $x \neq \text{FALSE}\ y \neq \text{FALSE}\ ((\text{IMPLIES}\ x)y) = \text{TRUE}$
SEM 27. $((\text{IMPLIES}\ x)y) \neq \text{FALSE}\ x = \text{TRUE}$
SEM 28. $((\text{IMPLIES}\ x)y) \neq \text{FALSE}\ y = \text{FALSE}$
SEM 29. $((\text{IMPLIES}\ x)y) \neq \text{TRUE}\ x = \text{FALSE}\ y = \text{TRUE}$
SEM 30. $x \neq y\ ((\text{EQUAL}\ x)y) = \text{TRUE}$
SEM 31. $x = y\ ((\text{EQUAL}\ x)y) = \text{FALSE}$
SEM 32. $((\text{EQUAL}\ x)y) \neq \text{TRUE}\ x = y$
SEM 33. $((\text{EQUAL}\ x)y) \neq \text{FALSE}\ x \neq y$
SEM 34. $x \neq \text{TRUE}\ (((\text{IF}\ x)y)z) = y$
SEM 35. $x \neq \text{FALSE}\ (((\text{IF}\ x)y)z) = z$
SEM 36. $(\text{ALL}\ x) \neq \text{TRUE}\ (xy) = \text{TRUE}$
SEM 37. $(xy) \neq \text{TRUE}\ (\text{EXISTS}\ x) = \text{TRUE}$
SEM 38. $(x(\text{FAIL}\ x)) = \text{TRUE}\ (\text{ALL}\ x) = \text{TRUE}$
SEM 39. $(\text{EXISTS}\ x) \neq \text{TRUE}\ (x(\text{CHOICE}\ x)) = \text{TRUE}$
SEM 40. $(\text{EXISTS}\ x) \neq \text{FALSE}\ (xy) = \text{FALSE}$
SEM 41. $(x(\text{CHOICE}\ x)) \neq \text{FALSE}\ (\text{EXISTS}\ x) = \text{FALSE}$
SEM 42. $(xy) \neq \text{FALSE}\ (\text{ALL}\ x) = \text{FALSE}$
SEM 43. $(\text{ALL}\ x) \neq \text{FALSE}\ (x(\text{FAIL}\ x)) = \text{FALSE}$
SEM 44. $x = x$
SEM 45. $x \neq y\ y = x$
SEM 46. $x \neq y\ y \neq z\ x = z$
SEM 47. $x \neq y\ u \neq v\ (xu) = (yv)$.

Any particular theorem-proving problem can then be treated by writing a set PROB of clauses in this language and seeking to deduce $\square$ (the empty clause) from the set: SEM$\cup$PROB. The deduction can proceed according to any valid principles of inference which apply to equality clauses. In particular the resolution principle may be used as sole principle; or the resolution principle together with paramodulation (Robinson and Wos 1969); or Sibert's system (Sibert 1969); or the E-resolution system of Morris (1969). In any of these systems, if SEM$\cup$PROB is unsatisfiable (in the usual first-order sense) then a deduction of $\square$ is automatically obtainable from SEM$\cup$PROB as premises; and conversely.

The underlying assumption here is that the (first-order) unsatisfiability of SEM$\cup$PROB is equivalent to the (intuitive, higher order) unsatisfiability of PROB alone. Whether or not this is so depends on the 'correctness' of the set SEM as a specification of the fundamental semantics. For our present purposes we will simply postulate the correctness of SEM, and confine our attention to the problem of designing suitable deductive machinery.

The various deductive systems mentioned above, while theoretically adequate, are no use in practice. The difficulty lies in the fact that the deductions produced in any of these systems are too long. Their length is caused by the very small size of the individual inferences they contain; they correspond to a 'micro' level of analysis of the reasoning, in which each 'macro' inference is broken down into a sequence of extremely elementary steps.

The way to deal with this problem seems clear: identify the 'macros', and set up deductive machinery in which they are the basic inference principles.

§4. An example will illustrate the ideas set forth in the previous section.

We are to show that the statement:

for all $y$, if (R$y$) then ($y$M) (1)

follows from the statements:

for all $x$, if (Q$x$) then ($x$M); (2)

for all $p$, if ($p$Q) then ($p$R). (3)

We begin by constructing terms in our language corresponding to each statement. This is conveniently done in two stages: first, construct a term using lambda-notation; then apply Schönfinkel's procedure. For example, (1) first becomes:

(ALL $\lambda y$((IMPLIES (R$y$))($y$M))) (4)

and then, by Schönfinkel's procedure:

(ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM)))). (5)

In similar fashion (2) and (3) become respectively:

(ALL((A((A(K IMPLIES))((A(KQ))I)))((AI)(KM)))); (6)

(ALL((A((A(K IMPLIES))((AI)(KQ))))((AI)(KR)))). (7)

Now in order to prove that (1) follows from (2) and (3) we will *assert* (2) and (3) and *deny* (1), and seek to deduce a contradiction. In our language this is done by equating the corresponding terms to TRUE and to FALSE and asserting the equations. This yields three unit clauses, comprising PROB:

PROB 1. (ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM)))) = FALSE

PROB 2. (ALL((A((A(K IMPLIES))((A(KQ))I)))((AI)(KM)))) = TRUE

PROB 3. (ALL((A((A(K IMPLIES))((AI)(KQ))))((AI)(KR)))) = TRUE

We now deduce $\square$ from the set SEM$\cup$PROB, the deduction comprising a set of clauses labelled DED. From SEM 36 and PROB 3 we obtain, by *resolution*:

DED 1. (((A((A(K IMPLIES))((AI)(KQ))))((AI)(KR)))$y$) = TRUE

whence we immediately infer, by *normalization*:

DED 2. ((IMPLIES($y$Q))($y$R)) = TRUE.

What is *normalization*? It is a 'macro' inference principle, which consists of *repeatedly applying the equations* SEM 1, SEM 2 and SEM 3 *until no further applications are possible.* In the present case the 'micro' steps by which DED 2 is obtained from DED 1 are:

DED 1. (((A((A(K IMPLIES))((AI)(KQ))))((AI)(KR)))$y$) = TRUE

⇒DED 1.1. ((((A(K IMPLIES))((AI)(KQ)))$y$)(((AI)(KR))$y$)) = TRUE
using SEM 1;

⇒DED 1.2. (((((K IMPLIES)$y$)(((AI)(KQ))$y$))(((AI)(KR))$y$)) = TRUE
using SEM 1;

⇒DED 1.3. ((IMPLIES(((AI)(KQ))$y$))(((AI)(KR))$y$)) = TRUE
using SEM 2;

⇒DED 1.4. ((IMPLIES((I$y$)((KQ)$y$)))(((AI)(KR))$y$)) = TRUE
using SEM 1;

⇒DED 1.5. ((IMPLIES($y$((KQ)$y$)))(((AI)(KR))$y$)) = TRUE
using SEM 3;

⇒DED 1.6. ((IMPLIES($y$Q))(((AI)(KR))$y$)) = TRUE
using SEM 2;

⇒DED 1.7. ((IMPLIES($yQ$))((I$y$)((KR)$y$))) = TRUE
using SEM 1;
⇒DED 1.8. ((IMPLIES($yQ$))($y$((KR)$y$))) = TRUE
using SEM 3;
⇒DED 2. ((IMPLIES($yQ$))($yR$)) = TRUE
using SEM 2.

Normalization yields exactly one conclusion when applied to any clause, and corresponds, in the Schönfinkel notation, to *conversion to normal form* in the lambda calculus. It is exceedingly easy to carry out in the machine.

Continuing with the deduction, we obtain from DED 2 and SEM 29, by *resolution*:

DED 3. ($yQ$) = FALSE($yR$) = TRUE.

whence, *resolving* with SEM 4, we get:

DED 4. ($yQ$) ≠ TRUE($yR$) = TRUE

and *resolving* with SEM 4 again:

DED 5. ($yQ$) ≠ TRUE($yR$) ≠ FALSE.

Our next inference involves another 'macro' principle. From PROB 1 we obtain, by *abstraction*, the clause:

DED 6. (((A(K ALL))((A((A(KA))((A(K(A(K IMPLIES))))
((A((A(KA))((A(KK))I)))(KI)))))(K((AI)(KM)))))R) = FALSE

and by *abstraction*, likewise, from PROB 2, we get:

DED 7. (((A(K ALL))((A((A(KA))((A(K(A(K IMPLIES))))
((A((A(KA))((A(KK))I)))(KI)))))(K((AI)(KM)))))Q) = TRUE.

What is *abstraction*? One way to characterize it is to say that it is just *the reverse of normalization*: equations SEM 1, SEM 2, SEM 3 being applied repeatedly, in a chain of substitutions. However, it is the *right hand* sides of these equations, not the left hand sides, which get replaced; just the opposite way round from the normalization replacements. For example, the chain of 'micro' steps by which DED 6 is reached from PROB 1 is shown in figure 1.

In the sequence of equality inferences we have indicated which subterm is replaced in each line, and which term replaces it to form the next line; and in each replacement we have indicated by which of SEM 1, SEM 2, or SEM 3 these two terms are equal. It will be noted that this same chain, *when read from bottom to top*, is a normalization of DED 6; that is, PROB 1 *can be inferred from* DED 6 *by normalization.*

A theorem-proving program based on elementary equality inferences would have to 'discover' this chain of steps by isolating it from all other chains of elementary inferences. In fact there is something quite special about this particular chain, but what is special about it cannot be expressed at the 'micro' level.

The overall transaction in this inference is *the solution of the following problem*:

find $\mathscr{F}$ such that:

(ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM)))) = ($\mathscr{F}$R).

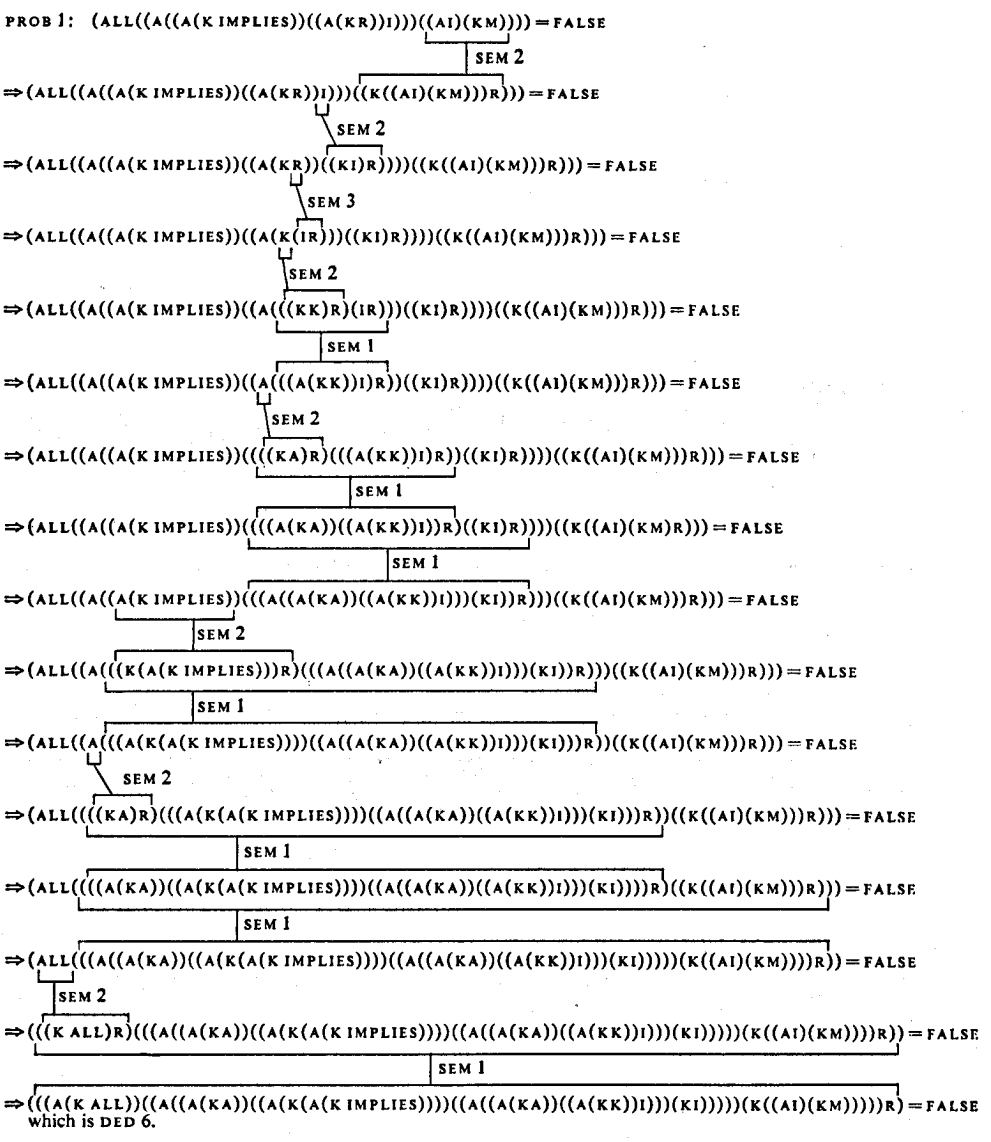



Figure 1

The left hand side of this equation is the left hand side of PROB 1. The problem intuitively is therefore: *express* PROB 1 *as*: $(\mathscr{F}\,\mathrm{R}) = \mathrm{FALSE}$. This amounts to asking 'what does the left hand side of PROB 1 say about R?' The left hand side of DED 6 has the form $(\mathscr{F}\,\mathrm{R})$, and its $\mathscr{F}$ in fact solves the problem.

Once the matter is seen in this light, however, a direct solution presents itself: first write

(ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM))))

as: (λx(ALL((A((A(K IMPLIES))((A(Kx))I)))((AI)(KM))))R)

and then *apply Schönfinkel's algorithm to eliminate the lambda notation.* The reader will easily verify that this produces the left hand side of DED 6.

The inference principle we are here calling *abstraction* can therefore be stated at the 'macro' level as follows: *from a clause* $C(t)$ (*i.e., a clause* $C$ *in which there is a particular occurrence of a term* $t$ *singled out for attention*) *infer the clause* $C(t')$ (*i.e., the result of replacing the singled-out occurrence of* $t$ *by an occurrence of the term* $t'$); *where* $t'$ *is the result of applying Schönfinkel's procedure to the expression* $(\lambda x t\{x/s\}s)$, *where* $s$ *is a constant occurring in* $t$.

(In the above, $t\{x/s\}$ is the result of substituting the variable $x$ for each occurrence in $t$ of the constant $s$.)

In the application of this rule which we have been examining, $C(t)$ is PROB 1; $t$ is its left hand side:

(ALL((A((A(K IMPLIES))((A(KR))I)))((AI)(KM))));

and $s$ is R. The reader will easily verify that $C(t')$ is then DED 6.

There are only finitely many ways in which abstraction can be applied to a given clause $C$, corresponding to the finitely many ways in which $t$ and $s$ can be chosen. For each choice of $t$ and $s$, the resulting clause is uniquely determined.

With this explanation of abstraction, the reader is now in a position to check that the next step in our deduction, DED 7, is indeed obtained by abstraction from PROB 2. That is, PROB 2 is $C$, its left hand side is $t$, and $Q$ is $s$. It follows that the result is as stated.

The deduction ends with

DED 8. □

which follows by *hyper-resolution* from DED 5, DED 6, and DED 7.

§5. The example of the previous section was also used in Robinson (1969). The main point of the proof is to spot that what (2) says about Q, (1) says about R; while (3) states that whatever can be said truly about Q can also be said truly about R. The abstraction rule is thus crucial in automatically generating the proof, and corresponds to a characteristically 'human' capability, namely, the ability to 'spot what a sentence says' about each thing which is mentioned in the sentence, and to see that some other sentence says the same about some other thing.

§6. The relation between Schönfinkel notation and the lambda-calculus notation is quite closely analogous to the relation between machine language and source language, in the context of programming. Schönfinkel's procedure, in this analogy, corresponds to compiling. Rules of inference such as normalization and abstraction are like routines written in machine code, whose function is viewed holistically at the source language level, yet analysed in detail at the machine level.

It would clearly be very useful to have, in terms of this analogy, a *decompilation* procedure – a procedure which translates a given piece of Schönfinkel notation into lambda notation by reversing the direction of application of the three replacement rules (6), (7) and (8) of section 1. There appears to be no difficulty about designing such a procedure. It would, for example, translate DED 6 into:

$$(\lambda x(\text{ALL } \lambda y((\text{IMPLIES } (xy))(y\text{M})))\text{R}) = \text{FALSE}.$$

§7. It is hoped that this brief discussion of the use of Schönfinkel notation to express higher order theorem-proving problems within first-order equation calculi will serve as a starting point for further investigations. There are a number of interesting questions that arise:

(1) What, besides normalization and abstraction, are appropriate 'macro' rules of inference? For example, one might well wish to broaden the notion of 'computing out' to cover not only the 'evaluation' of A, K and I – which is essentially what normalization is – but also the evaluation of other functions, such as the boolean functions, the standard arithmetic functions, or indeed any functions for which there exists information enough in the system to replace *applications* of them by the *results* of those applications.

(2) Is SEM adequate? There may be further constants which ought to be included and axiomatized as 'system constants' – e.g., CAR, CDR, CONS, and ATOM, of the LISP system. It may be that there are better ways of axiomatizing the present set of system constants than those incorporated in the present SEM.

(3) Ought we to reimpose the *type* classification (*see* Robinson 1969) or not? What purposes does this classification really serve?

(4) Might it be useful to allow relation symbols other than *equality*, and function symbols other than *application*, into the first-order calculi we use to express higher order problems? If so, what relations and functions should be represented? Usefulness apart, is it *theoretically necessary* to introduce further notions?

**Acknowledgement**

The work described was completed while the author held a Senior Visiting Fellowship in the Metamathematics Unit, Edinburgh University, financed by the Science Research Council.

## REFERENCES

Curry, H.B. & Feys, R. (1958) *Combinatory Logic, Volume I.* Amsterdam: North Holland Publishing Company.

Morris, J.B. (1969) E-Resolution: extension of resolution to include the equality relation. *Proceedings of the First International Joint Conference on Artificial Intelligence.* Washington D.C.

Robinson, G., & Wos, L. (1969) Paramodulation and theorem-proving in first-order theories with equality. *Machine Intelligence 4*, pp. 135–50 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Robinson, J.A. (1969) Mechanizing higher order logic. *Machine Intelligence 4*, pp. 151–70 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Rosenbloom, P.C. (1950) *The Elements of Mathematical Logic.* New York: Dover Publications.

Schönfinkel, M. (1924) Uber die Bausteine der mathematischen Logik. *Mathematische Annalen*, **92**, 305–16. English version entitled: On the building blocks of mathematical logic. In *From Frege to Godel: A Source Book in Mathematical Logic*, pp. 355–66 (ed. van Heijenoort, J.). Harvard University Press, 1967.

Sibert, E.E. (1969) A machine-oriented logic incorporating the equality relation. *Machine Intelligence 4*, pp. 103–34 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

# 7

# Transformational Systems and the Algebraic Structure of Atomic Formulas

John C. Reynolds
Argonne National Laboratory
Illinois

**Abstract**

If the set of atomic formulas is augmented by adding a 'universal formula' and a 'null formula', then the equivalence classes of this set under alphabetic variation form a complete non-modular lattice, with 'instance' as the partial ordering, 'greatest common instance' as the meet operation, and 'least common generalization' as the join operation. The greatest common instance of two formulas can be obtained from Robinson's Unification Algorithm. An algorithm is given for computing the least common generalization of two formulas, the covering relation of the lattice is determined, bounds are obtained on the length of chains from one formula to another, and it is shown that any formula is the least common generalization of its set of ground instances.

A transformational system is a finite set of clauses containing only units and transformations, which are clauses containing exactly one positive and one negative literal. It is shown that every unsatisfiable transformational system has a refutation where every resolution has at least one resolvend which is an initial clause. An algorithm is given for computing a common generalization of all atomic formulas which can be derived from a transformational system, and it is shown that there is no decision procedure for transformational systems.

## INTRODUCTION

This paper is a collection of theoretical properties of the entities and operations used in the resolution approach to mechanical theorem-proving (Robinson 1965). Hopefully, this material will eventually be helpful in the development of more efficient proof procedures, but we are presently unable to formulate a complete procedure which takes full advantage of our results.

These results fall in two separate areas. The first is the algebraic structure

of atomic formulas under instantiation. Robinson's Unification Algorithm allows the computation of the greatest common instance of any finite set of unifiable atomic formulas. This suggests the existence of a dual operation of 'least common generalization'. It turns out that such an operation does exist and can be computed by a simple algorithm.

As a result, if one adds a 'universal atomic formula' (whose ground instances are all ground formulas) and a 'null atomic formula' (with no ground instances), then the set of atomic formulas (more precisely, the equivalence classes of atomic formulas under alphabetic variation) forms a complete non-modular lattice, with 'instance' as the partial ordering, 'greatest common instance' as the meet operation, and 'least common generalization' as the join operation. The covering relation in this lattice determines the set of 'closest' instances of an atomic formula, and bounds can be obtained on the length of chains from one atomic formula to another.

The second area is the properties of transformational systems, which are sets of clauses containing only units and mixed two-clauses (clauses with one positive and one negative literal). The refutation of transformational systems seems to be the simplest non-trivial case of theorem-proving. Its simplicity is that a search for a refutation can be limited to resolutions in which one resolvend is a unit and the other is an initial clause; in effect one is doing path-searching rather than tree-searching. The non-triviality is that Church's theorem still holds: by an appropriate mapping of Post's correspondence problem it can be shown that no decision procedure exists for transformational systems.

A connection between these areas is provided by the least common generalization. It can be used to examine a subset $S$ of a transformational system and to compute a single unit which is a 'super-consequence' of $S$. This unit is not necessarily a valid consequence of $S$, but each of the (usually infinite number of) consequences must be an instance of the super-consequence. Thus when the super-consequence can be shown to be irrelevant, all of the consequences of $S$ must be irrelevant, and the generation of an infinite sequence of useless clauses can be avoided.

Throughout this paper, we will use the definitions and results given in Robinson's original paper on resolution (Robinson 1965). Also, a variety of well-known properties of lattices will be stated without proof; these properties are discussed in the opening chapters of a standard text such as Birkhoff (1967).

## THE LATTICE STRUCTURE OF ATOMIC FORMULAS

In this section we will show that the operation of instantiation induces a lattice-like structure on the set of atomic formulas, and we will examine various properties of this structure. First, we must generalize the notion of an atomic formula to include a *universal formula* $\mathcal{A}$ and a *null formula* $\Omega$; these formulas will be the greatest and least elements of the lattice.

*Definition.* A *generalized atomic formula* (GAF) is either a conventional atomic formula (CAF) as defined in Robinson (1965), or one of the special symbols $\mathscr{A}$, $\Omega$. A GAF is called a *ground GAF* iff it is a CAF and it contains no occurrences of variables.

Given GAFs $A$ and $B$, we write $A \geqslant B$ (read $A$ is a *generalization* of $B$ or $B$ is an *instance* of $A$) iff $A$ is $\mathscr{A}$, or $B$ is $\Omega$, or $A$ and $B$ are both CAFs and there exists a substitution $\theta$ such that $B = A\theta$. If $A \geqslant B$ and $B \geqslant A$, we write $A \simeq B$ (read $A$ and $B$ are *variants* or $A$ and $B$ are *equivalent*). If $A \geqslant B$ and not $B \geqslant A$, we write $A > B$ (read $A$ is a *proper generalization* of $B$ or $B$ is a *proper instance* of $A$).

(We assume that the set of conventional atomic formulas is generated from a fixed but unspecified vocabulary containing at least one constant, one unary function symbol, and one binary predicate symbol. A second, distinct predicate symbol will be required in Theorem 8; a binary function symbol will be required in Theorem 11.)

*Corollary* 1

The relation $\geqslant$ is a quasi-ordering in which $\mathscr{A}$ and $\Omega$ are unique greatest and least elements, i.e., for all GAFs $A$, $B$, and $C$:

$A \geqslant A$
$A \geqslant B$ and $B \geqslant C$ implies $A \geqslant C$
$\mathscr{A} \geqslant A$
$A \geqslant \Omega$
$A \geqslant \mathscr{A}$ iff $A = \mathscr{A}$
$\Omega \geqslant A$ iff $A = \Omega$.

*Lemma* 1

$A \simeq B$ iff one of the following cases occurs:

(1) $A = B = \mathscr{A}$.

(2) $A = B = \Omega$.

(3) $A$ and $B$ are CAFs and there is a substitution $\theta$ such that: $B = A\theta$, for all variables $X$ occurring in $A$, $X\theta$ is a variable, and for all pairs $X$, $Y$ of distinct variables occurring in $A$, $X\theta \neq Y\theta$.

*Proof.* If either $A$ or $B$ is not a CAF, the lemma is trivial. If $A$ and $B$ are CAFs and $A \simeq B$, then there are substitutions $\theta$ and $\psi$ such that $B = A\theta$ and $A = B\psi = A\theta\psi$. Then if $X$ is any variable occurring in $A$, $X\theta\psi = X$, so that $X\theta$ must be a variable. If $X$ and $Y$ are distinct variables occurring in $A$, then $X\theta\psi = X$ and $Y\theta\psi = Y \neq X\theta\psi$, so that $X\theta \neq Y\theta$.

On the other hand, if $B = A\theta$, where $\theta$ meets condition (3), let $X_1, \ldots, X_n$ be the variables occurring in $A$. Then $\psi = \{X_1/X_1\theta, \ldots, X_n/X_n\theta\}$ is a substitution, and $B\psi = A\{X_1\theta/X_1, \ldots, X_n\theta/X_n\}\psi = A$. Thus $A \simeq B$.

*Definition.* Let $S$ be a set of GAFs. If $I$ is a GAF such that, for all $A \in S$, $I \leqslant A$, then $I$ is a *common instance* of $S$. If $I$ is a common instance of $S$ and, for all common instances $I'$ of $S$, $I \geqslant I'$, then $I$ is a *greatest common instance* of $S$. If $G$ is a GAF such that, for all $A \in S$, $G \geqslant A$, then $G$ is a *common*

*generalization* of $S$. If $G$ is a common generalization of $S$ and, for all common generalizations $G'$ of $S$, $G \leqslant G'$, then $G$ is a *least common generalization* of $S$.

It is easily shown that:

*Corollary* 2

$\mathscr{A}$ and $\Omega$ are the only greatest common instance and least common generalization of the empty set. $\Omega$ and $\mathscr{A}$ are the only greatest common instance and least common generalization of the set of all GAFs.

If $A$ is a greatest common instance (least common generalization) of $S$, then $B$ is a greatest common instance (least common generalization) of $S$ iff $B \simeq A$.

We now define two total, computable binary functions $\sqcap$ and $\sqcup$, and show that these functions produce a greatest common instance and a least common generalization of any pair of GAFs.

*Definition.* Given GAFs $A$ and $B$, we define the GAF $A \sqcap B$ as follows:

(1) If $A = \mathscr{A}$, then $A \sqcap B = B$.

(2) If $B = \mathscr{A}$, then $A \sqcap B = A$.

(3) If $A = \Omega$ or $B = \Omega$, then $A \sqcap B = \Omega$.

(4) If $A$ and $B$ are both CAFs, then let $A'$ and $B'$ be variants of $A$ and $B$ respectively (chosen in some standard manner) such that no variable occurs in both $A'$ and $B'$. If $A'$ and $B'$ are not unifiable, then $A \sqcap B = \Omega$. Otherwise $A \sqcap B = A'\sigma = B'\sigma$, where $\sigma$ is the most general unifier of $A'$ and $B'$ [obtained from the Unification Algorithm given in Robinson (1965)].

Then the Unification Theorem in Robinson (1965) has the following direct consequence:

*Theorem* 1

For all GAFs $A$, $B$, and $C$, $A \geqslant A \sqcap B$, $B \geqslant A \sqcap B$, and if $A \geqslant C$ and $B \geqslant C$, then $A \sqcap B \geqslant C$. Thus $A \sqcap B$ is a greatest common instance of $\{A, B\}$.

*Definition.* Given GAFs $A$ and $B$, we define the GAF $A \sqcup B$ as follows:

(1) If $A = \Omega$, then $A \sqcup B = B$.

(2) If $B = \Omega$, then $A \sqcup B = A$.

(3) If $A = \mathscr{A}$, or $B = \mathscr{A}$, or $A$ and $B$ begin with distinct predicate symbols, then $A \sqcup B = \mathscr{A}$.

(4) If $A$ and $B$ are CAFs beginning with the same predicate symbol, then let $Z_1, Z_2, \ldots$ be a sequence of variables which do not occur in $A$ or $B$, and obtain $A \sqcup B$ by the following *Anti-unification Algorithm*:*

(a) Set the variables $\bar{A}$ to $A$, $\bar{B}$ to $B$, $\zeta$ and $\eta$ to the empty substitution, and $i$ to zero.

(b) If $\bar{A} = \bar{B}$, exit with $A \sqcup B = \bar{A} = \bar{B}$.

(c) Let $k$ be the index of the first symbol position at which $\bar{A}$ and $\bar{B}$ differ,

* This algorithm has been discovered independently by Mr Gordon Plotkin of the University of Edinburgh.

and let $S$ and $T$ be the terms which occur, beginning in the $k$th position, in $\bar{A}$ and $\bar{B}$ respectively.

(d) If, for some $j$ such that $1 \leqslant j \leqslant i$, $Z_j\zeta = S$ and $Z_j\eta = T$, then alter $\bar{A}$ by replacing the occurrence of $S$ beginning in the $k$th position by $Z_j$, alter $\bar{B}$ by replacing the occurrence of $T$ beginning in the $k$th position by $Z_j$, and go to step (b).

(e) Otherwise, increase $i$ by one, alter $\bar{A}$ by replacing the occurrence of $S$ beginning in the $k$th position by $Z_i$, alter $\bar{B}$ by replacing the occurrence of $T$ beginning in the $k$th position by $Z_i$, replace $\zeta$ by $\zeta \cup \{S/Z_i\}$, replace $\eta$ by $\eta \cup \{T/Z_i\}$, and go to step (b).

Each iteration of the Anti-unification Algorithm will either produce the termination condition $\bar{A} = \bar{B}$, or else it will increase $k$ without increasing the length of $\bar{A}$ or $\bar{B}$. Thus, since $k$ cannot exceed the length of $\bar{A}$ or $\bar{B}$, the algorithm must always terminate. The following lemma may be proved by induction on the number of iterations:

*Lemma* 2

After each iteration of the Anti-unification Algorithm:

(1) $\bar{A}\zeta = A$ and $\bar{B}\eta = B$.

(2) Each of the variables $Z_1, \ldots, Z_i$ occurs in both $\bar{A}$ and $\bar{B}$, but only to the left of the first symbol position at which $\bar{A}$ and $\bar{B}$ differ.

(3) There exist terms $S_1, \ldots, S_i, T_1, \ldots, T_i$ such that:

(a) $\zeta = \{S_1/Z_1, \ldots, S_i/Z_i\}$.

(b) $\eta = \{T_1/Z_1, \ldots, T_i/Z_i\}$.

(c) For $1 \leqslant j \leqslant i$, $S_j$ and $T_j$ differ in their first symbols.

(d) For $1 \leqslant j, k \leqslant i$, if $j \neq k$ then either $S_j \neq S_k$ or $T_j \neq T_k$.

*Theorem* 2

For all GAFs $A$, $B$, and $C$, $A \sqcup B \geqslant A$, $A \sqcup B \geqslant B$, and if $C \geqslant A$ and $C \geqslant B$, then $C \geqslant A \sqcup B$. Thus $A \sqcup B$ is a least common generalization of $\{A, B\}$.

*Proof.* The theorem is non-trivial only in the case where $A$ and $B$ are both CAFs beginning with the same predicate symbol. In this case, $A \sqcup B$ is defined by the Anti-unification Algorithm, and part 1 of Lemma 2 implies that $A \sqcup B \geqslant A$ and $A \sqcup B \geqslant B$.

Now suppose $C$ is a GAF such that $C \geqslant A$ and $C \geqslant B$, and let $D = C \sqcap (A \sqcup B)$. By Theorem 1, $C \geqslant D$, $(A \sqcup B) \geqslant D$, and, since $A$ and $B$ are both common instances of $C$ and $(A \sqcup B)$, $D \geqslant A$ and $D \geqslant B$. Moreover, since $D$ is both an instance of a CAF and a generalization of a CAF, $D$ must be a CAF. Thus there exist substitutions $\theta$, $\psi$, $\phi$ such that $D = (A \sqcup B)\theta$, $A = D\psi = (A \sqcup B)\theta\psi$, and $B = D\phi = (A \sqcup B)\theta\phi$.

Let $\zeta = \{S_1/Z_1, \ldots, S_i/Z_i\}$ and $\eta = \{T_1/Z_1, \ldots, T_i/Z_i\}$ be the final values of the variables $\zeta$ and $\eta$ in the execution of the Anti-unification Algorithm used to compute $A \sqcup B$. By part 1 of Lemma 2, $(A \sqcup B)\zeta = A = (A \sqcup B)\theta\psi$, and $(A \sqcup B)\eta = B = (A \sqcup B)\theta\phi$. Thus if $X$ is any variable occurring in $A \sqcup B$, then $X\zeta = X\theta\psi$ and $X\eta = X\theta\phi$.

symbol, and let $Z_1, \ldots, Z_k$ be distinct variables not occurring in $A_i$. Then set

$$A_{i+1} = A_i\{FZ_1 \ldots Z_k/X\}$$
$$\theta_{i+1} = (\theta_i - \{FT_1 \ldots T_k/X\}) \cup \{T_1/Z_1, \ldots, T_k/Z_k\}.$$

Increase $i$ by 1 and repeat step 2.

(3) If there is no pair $X$, $Y$ of distinct variables occurring in $A_i$ such that $X\theta_i = Y\theta_i$, then go to step 4. Otherwise, let $X$, $Y$ be such a pair and set

$$A_{i+1} = A_i\{Y/X\}$$
$$\theta_{i+1} = \theta_i - \{X\theta_i/X\}.$$

Increase $i$ by 1 and repeat step 3.

(4) Set $n = i$ and terminate.

Step 2 must terminate since each iteration decreases the total number of function symbol occurrences in the terms of $\theta_i$. Step 3 must terminate since each iteration decreases the number of distinct variables occurring in $A_i$. By induction on $i$, one can show that:

$$B = A_i\theta_i (0 \leqslant i \leqslant n).$$

All variables of $\theta_i$ occur in $A_i (0 \leqslant i \leqslant n)$.

$$A_i \rightarrow A_{i+1} \quad (0 \leqslant i \leqslant n-1).$$

When termination is reached, all of the terms of $\theta_n$ will be variables, and for any pair $X$, $Y$ of distinct variables occurring in $A_n$, $X\theta_n \neq Y\theta_n$. Thus $A_n \simeq B$.

We now consider the cases where $A$ or $B$ is not a CAF. If $A = \mathscr{A}$ and $B$ is a CAF, let $P$ be the predicate symbol beginning $B$, let $k$ be the degree of $P$, and let $A' = PZ_1 \ldots Z_k$, where $Z_1, \ldots, Z_k$ are distinct variables. Then $A \rightarrow A' \geqslant B$ and $A'$ and $B$ are CAFs, so that a chain from $A'$ to $B$ can be constructed as above and joined to the link $A \rightarrow A'$.

If $A$ is $\mathscr{A}$ or a CAF and $B$ is $\Omega$, let $B'$ be any ground instance of $A$. Then $A \geqslant B' \rightarrow B$ and $B'$ is a CAF, so that a chain from $A$ to $B'$ can be constructed as above and joined to the link $B' \rightarrow B$.

The remaining cases are $A = B = \mathscr{A}$ and $A = B = \Omega$, which are trivial. The assertion that $n \geqslant 1$ when $A > B$ is also trivial, since $n = 0$ implies $A \simeq B$.

*Definition.* The *size* of a GAF is defined as follows: size $(\mathscr{A}) = 0$. size $(\Omega) = \infty$. If $A$ is a CAF, then size $(A)$ is the number of symbol occurrences in $A$ minus the number of distinct variables occurring in $A$.

*Corollary* 5

If $A$ and $B$ are GAFs, then $A \simeq B$ implies size $(B) =$ size $(A)$, and $A \rightarrow B$ implies size $(B) >$ size $(A)$. More generally (by Theorem 4) $A > B$ implies size $(B) >$ size $(A)$.

We will now use Theorem 4 and Corollary 5 to obtain bounds on the length of chains, to determine the covering relation, and to show that the GAF lattice is complete.

*Theorem* 5

(1) If $B \neq \Omega$, then there is no chain from $A$ to $B$ whose length is greater than size $(B)$–size $(A)$.

(2) There are no infinite ascending chains from any GAF.

(3) If $A$ is a ground GAF, then the only chain from $A$ to $\Omega$ is $A \rightarrow \Omega$, and there are no infinite descending chains from $A$.

(4) If $A$ is not a ground GAF and is not $\Omega$, then there is no bound on the length of chains from $A$ to $\Omega$, and there is an infinite descending total chain from $A$.

*Proof.* (1) Let $A = A_0 > A_1 > \ldots > A_n \simeq B$ be any chain from $A$ to $B$. Then by Corollary 5, size $(A)$ = size $(A_0)$ < size $(A_1)$ < ... < size $(A_n)$ = size $(B)$. Thus $n <$ size $(B)$ − size $(A)$.

(2) An infinite ascending chain $A_0 < A_1 < A_2 < \ldots$ would imply that size $(A_0) >$ size $(A_1) >$ size $(A_2) > \ldots$. But this is impossible, since $A_1 \neq \Omega$ implies that $A_1$ has finite size.

(3) If $A_0$ is a ground GAF, then $A_0 > A_1$ implies $A_1 = \Omega$, and there is no $A_2$ such that $\Omega > A_2$.

(4) Suppose $A$ is a non-ground CAF. Let $X$ be a variable occurring in $A$, let $F$ be a unary function symbol, and let $A_i = A\{FX/X\}^i$. Then $A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow \ldots$ is an infinite descending total chain from $A$, and for any $n \geqslant 0$, $A_0 \rightarrow A_1 \ldots \rightarrow A_n \rightarrow \Omega$ is a chain of length $n+1$ from $A$ to $\Omega$.

If $A$ is $\mathscr{A}$, similar chains can be constructed by taking $A_1$ to be $PZ_1 \ldots Z_k$, where $P$ is a predicate symbol of degree $k \geqslant 1$ and $Z_1, \ldots, Z_k$ are distinct variables, and then continuing as above.

*Definition.* *A covers B* iff $A > B$ and there is no GAF $C$ such that $A > C > B$.

*Theorem* 6

$A$ covers $B$ iff $A \rightarrow B$.

*Proof.* Suppose $A$ covers $B$. By Theorem 4, there is a total chain $A = A_0 \rightarrow A_1 \rightarrow \ldots \rightarrow A_n \simeq B$ with some length $n \geqslant 1$. But if $n > 1$, then $A > A_1 > B$, which contradicts the definition of covering. Thus $n = 1$ and $A \rightarrow A_1 \simeq B$. By the definition of $\rightarrow$, this implies $A \rightarrow B$.

Conversely, suppose $A \rightarrow B$. Since this implies that $A > B$, we need only show that $A > C > B$ is impossible. We distinguish three cases:

(1) $A$ is a ground GAF and $B = \Omega$. This prevents $A > C > B$ since $\Omega$ is the only proper instance of $A$.

(2) Either (i) $A$ is $\mathscr{A}$ and $B$ is a CAF containing no function symbols and no repeated occurrences of a variable, or (ii) $A$ and $B$ are GAFs such that $B \simeq A\{Y/X\}$, where $X$ and $Y$ are distinct variables occurring in $A$. In either situation, size $(B)$ = size $(A) + 1$, while $A > C > B$ would imply size $(B) >$ size $(A) + 1$.

(3) $A$ and $B$ are CAFs such that for some function symbol $F$ of degree $k$, $B \simeq A\{FZ_1 \ldots Z_k/X_1\}$, where $X_1$ is a variable occurring in $A$ and $Z_1, \ldots, Z_k$ are distinct variables not occurring in $A$. Suppose there exists a GAF $C$ such that $A > C > B$. Then by Theorem 4, there exists a CAF $A_1$ such that $A \rightarrow A_1 \geqslant C > B$. Since $A \rightarrow A_1$ and $A$ is a CAF, we have $A_1 \simeq A\theta$, where $\theta$ is a substitution which meets the criteria of part 2 or 3 of the definition of the relation $\rightarrow$.

Since $A\theta \simeq A_1 \geqslant C > B \simeq A\{FZ_1 \ldots Z_k/X_1\}$, there is a substitution $\psi$ such that $A\theta\psi = A\{FZ_1 \ldots Z_k/X_1\}$.

Let $X_1, \ldots, X_n$ be the variables occurring in $A$. Then $X_1\theta\psi = FZ_1 \ldots Z_k$, and for $2 \leqslant i \leqslant n$, $X_i\theta\psi = X_i$. This implies that either (i) $X_1\theta$ begins with the symbol $F$ and $X_2\theta, \ldots, X_n\theta$ are distinct variables, or (ii) $X_1\theta, \ldots, X_n\theta$ are distinct variables. In order to satisfy both this condition and part 2 or 3 of the definition of $\rightarrow$, $\theta$ must have the form $\{FZ_1' \ldots Z_k'/X_1\}$, where $Z_1', \ldots, Z_k'$ are distinct variables not occurring in $A$. But then $A\theta = A\{FZ_1' \ldots Z_k'/X_1\}$ is a variant of $A\{FZ_1 \ldots Z_k/X_1\}$, which contradicts $A\theta \simeq A_1 \geqslant C > B \simeq A\{FZ_1 \ldots Z_k/X_1\}$.

*Theorem* 7

Any set $S$ of GAFs possesses a greatest common instance and a least common generalization, i.e., the GAF lattice is *complete.*

*Proof.* If $S = \{A_1, \ldots, A_n\}$ is finite, then the theorem is trivial. Let

$$I_0 = \mathscr{A}$$
$$I_{i+1} = I_i \sqcap A_{i+1} \quad (0 \leqslant i \leqslant n-1)$$
$$G_0 = \Omega$$
$$G_{i+1} = G_i \sqcup A_{i+1} \quad (0 \leqslant i \leqslant n-1).$$

Then it is easily shown that $I_n$ and $G_n$ are a greatest common instance and a least common generalization of $S$.

If $S$ is infinite, let $A_1, A_2, \ldots$ be an enumeration of the members of $S$, and let

$$G_0 = \Omega$$
$$G_{i+1} = G_i \sqcup A_{i+1} \quad (i \geqslant 0).$$

The $G_i$s satisfy $G_0 \leqslant G_1 \leqslant G_2 \leqslant \ldots$. But there cannot be an infinite sequence of $i$s such that $G_i < G_{i+1}$, for this would imply that some sub-sequence of the $G_i$s is an infinite ascending chain. Thus there exists an integer $i_0$ such that $G_i \simeq G_{i_0}$ for all $i \geqslant i_0$. It is easily shown that $G_{i_0}$ is a least common generalization of $S$.

A similar argument cannot be used for the greatest common instance, since the GAF lattice does contain infinite descending chains. Instead let

$$T = \{B \mid B \leqslant A \text{ for all } A \in S\},$$

and let $I$ be the least common generalization of $T$. Then if $A$ is any member of $S$, $A \geqslant B$ for all $B \in T$, and therefore $A \geqslant I$; thus $I$ is a common instance of $S$. On the other hand, if $B$ is a common instance of $S$, then $B \in T$, and therefore $B \leqslant I$; thus $I$ is a greatest common instance.

Since all greatest common instances of the same set are equivalent, we will refer to 'the' greatest common instance of a set, assuming that a particular member of the equivalence class can be selected in some well-defined manner. A similar situation holds for the least common generalization.

*Definition.* If $S$ is a set of GAFs, we write $\sqcap S$ for the greatest common instance of $S$ and $\sqcup S$ for the least common generalization of $S$.

It is easily shown that:

*Corollary* 6

If $S$ and $T$ are sets of GAFs such that $S \supseteq T$, then $\bigsqcup S \geqslant \bigsqcup T$ and $\bigsqcap S \leqslant \bigsqcap T$.

So far, our investigation of the GAF lattice has been limited to relations between the GAFs themselves. We now consider the relation between GAFs and their sets of ground instances:

*Definition.* For any GAF $A$, $\mathscr{G}(A)$ denotes the set of all ground GAFs which are instances of $A$.

*Theorem* 8

For all GAFs $A$ and $B$:

$\mathscr{G}(A)$ is empty iff $A=\Omega$.
$A \simeq \bigsqcup \mathscr{G}(A)$.
$A \geqslant B$ iff $\mathscr{G}(A) \supseteq \mathscr{G}(B)$.
$\mathscr{G}(A \sqcap B) = \mathscr{G}(A) \cap \mathscr{G}(B)$.
$\mathscr{G}(A \sqcup B) \supseteq \mathscr{G}(A) \cup \mathscr{G}(B)$.

*Proof.* The crux of the theorem is $A \simeq \bigsqcup \mathscr{G}(A)$, which implies that the set of ground instances of $A$ is 'rich' enough to determine $A$ within an equivalence. Once this assertion has been established, the rest of the theorem is a straightforward application of elementary lattice theory.

Suppose $A$ is a non-ground CAF, and let $B = \bigsqcup \mathscr{G}(A)$. Since every ground instance is an instance, $A$ is a common generalization of $\mathscr{G}(A)$, so that $A \geqslant B$. Since $A$ has at least one ground instance, $B \neq \Omega$. Thus there is a substitution $\theta$ such that $B = A\theta$.

Let $X_1, \ldots, X_n$ be the variables occurring in $A$, let $C$ be a constant, and let $F$ be a unary function symbol. Then $\mathscr{G}(A)$ must contain the following ground instances:

$$G_C = A\{C/X_1, \ldots, C/X_n\}$$
$$G_F = A\{FC/X_1, \ldots, FC/X_n\}$$
$$G_N = A\{C/X_1, FC/X_2, \ldots, \underbrace{F \ldots FC}_{n \text{ times}}/X_n\}.$$

Since each of these ground GAFs are instances of $B$, there are substitutions $\phi_C$, $\phi_F$, and $\phi_N$ such that

$$G_C = B\phi_C = A\theta\phi_C$$
$$G_F = B\phi_F = A\theta\phi_F$$
$$G_N = B\phi_N = A\theta\phi_N.$$

From the equations for $G_C$ and $G_F$ we have, for $1 \leqslant i \leqslant n$, $X_i\theta\phi_C = C$ and $X_i\theta\phi_F = FC$; thus each $X_i\theta$ must be a variable. From the equations for $G_N$ we have, for $1 \leqslant i \leqslant n$, $X_i\theta\phi_N = \underbrace{F \ldots FC}_{i \text{ times}}$; thus each $X_i\theta$ must be distinct. Therefore $B = A\theta$ is a variant of $A$.

If $A = \Omega$, then $\bigsqcup \mathscr{G}(A) = \bigsqcup\{\} = \Omega$. If $A$ is a ground GAF, then $\bigsqcup \mathscr{G}(A) = \bigsqcup\{A\} \simeq A$. If $A = \mathscr{A}$, then $\mathscr{G}(\mathscr{A})$ is the set of all ground GAFs. Since this

set contains GAFs which begin with different predicate symbols, its only common generalization is $\mathscr{A}$.

It should be noted that the existence of different predicate symbols is a necessary condition for this theorem; if all CAFs begin with the same symbol $P$, then $\bigsqcup\mathscr{G}(\mathscr{A}) = PZ_1 \ldots Z_k$.

Finally, we consider the interaction of our lattice relations with the operation of resolution. Since the lattice relations are relations between atomic formulas rather than clauses, it is difficult to make any significant statements about resolution in general. But useful results can be obtained for the case where a unit clause is resolved against a mixed two-clause (which we will call a *transformation*) to yield another unit.

*Definition.* A *transformation* is a clause containing exactly one positive and one negative literal; i.e., it is a clause of the form $\{\neg P, Q\}$, where $P$ and $Q$ are CAFs. If $A$ is a GAF and $t=\{\neg P, Q\}$ is a transformation, then $r_t(A)$ is defined as follows: If $A=\mathscr{A}$ then $r_t(A)=Q$, else if $A=\Omega$ or $\{A\}$ and $t$ have no resolvent, then $r_t(A)=\Omega$, otherwise $r_t(A)$ is the CAF which is the only member of the unique resolvent of $\{A\}$ and $t$.

*Lemma* 3

Let $A$ be a GAF and $t=\{\neg P, Q\}$ be a transformation. If $A\sqcap P=\Omega$, then $r_t(A)=\Omega$. If $A\sqcap P\neq\Omega$, then there is a substitution $\sigma$ such that $r_t(A)\simeq Q\sigma$, $A\sqcap P\simeq P\sigma$, and for all variables $X$, if $X$ is a variable of $\sigma$ or occurs in any term of $\sigma$, then either $X$ occurs in $P$ or $X$ does not occur in $Q$.

*Proof.* If $A=\mathscr{A}$, then $A\sqcap P\neq\Omega$ and $\sigma$ is the empty substitution. If $A=\Omega$, then $A\sqcap P=\Omega$ and $r_t(A)=\Omega$. Otherwise, let $A'$ be a variant of $A$ which has no variables in common with $P$ or $Q$. If $A'$ and $P$ are not unifiable, then $A\sqcap P=\Omega$, $\{A\}$ and $t$ have no resolvent, and $r_t(A)=\Omega$. If $A'$ and $P$ are unifiable, then $\{A\}$ and $t$ have the single resolvent (whose only member is) $r_t(A)\simeq Q\sigma$, where $\sigma$ is the most general unifier of $A'$ and $P$. Then the definition of $\sqcap$ implies that $A\sqcap P\simeq P\sigma\neq\Omega$. Moreover, the nature of the Unification Algorithm insures that every variable of $\sigma$ and every variable occurring in the terms of $\sigma$ is a variable which occurs in $A'$ or $P$. Thus if such a variable does not occur in $P$, it does not occur in $Q$.

*Theorem* 9

If $A$ and $B$ are GAFs and $t=\{\neg P, Q\}$ is a transformation, then

$$\text{if } A\geqslant B\sqcap P \text{ then } r_t(A)\geqslant r_t(B)$$
$$r_t(A)\simeq r_t(A\sqcap P)$$
$$Q\geqslant r_t(A)$$
$$r_t(A\sqcap B)\leqslant r_t(A)\sqcap r_t(B)$$
$$r_t(A\sqcup B)\geqslant r_t(A)\sqcup r_t(B).$$

*Proof.* Suppose $A\geqslant B\sqcap P$. Then $B\sqcap P$ is a common instance of $A$ and $P$, so that $A\sqcap P\geqslant B\sqcap P$. By Lemma 3, if $B\sqcap P=\Omega$, then $r_t(A)\geqslant r_t(B)=\Omega$. Otherwise, $A\sqcap P\neq\Omega$ and $B\sqcap P\neq\Omega$, so that there are substitutions $\sigma_A$ and $\sigma_B$ such that $r_t(A)\simeq Q\sigma_A$, $r_t(B)\simeq Q\sigma_B$, $A\sqcap P\simeq P\sigma_A$, $B\sqcap P\simeq P\sigma_B$, and for all variables

$X$, if $X$ is a variable of $\sigma_A$ or $\sigma_B$ or occurs in any term of $\sigma_A$ or $\sigma_B$, then either $X$ occurs in $P$ or $X$ does not occur in $Q$.

Since $P\sigma_A \geqslant P\sigma_B$, there is a substitution $\phi$ such that $P\sigma_B = P\sigma_A\phi$ and every variable of $\phi$ either occurs in $P$ or in some term of $\sigma_A$. Let $X$ be any variable occurring in $Q$. If $X$ occurs in $P$ then $X\sigma_B\phi = X\sigma_A$, while if $X$ does not occur in $P$, then $X$ is not a variable of $\sigma_A$, $\phi$, or $\sigma_B$, so that $X\sigma_B = X\sigma_A\phi = X$. Thus $Q\sigma_B = Q\sigma_A\phi$, and $r_t(A) \simeq Q\sigma_A \geqslant Q\sigma_B \simeq r_t(B)$.

The remaining parts of the theorem are direct consequences of this result and the basic properties of $\sqcap$ and $\sqcup$.

## TRANSFORMATIONAL SYSTEMS

In Theorem 9, we have developed certain properties of the resolution operation which hold only when one resolvend is a unit clause and the other is a transformation. This leads naturally to the investigation of sets of clauses, called *transformational systems*, which can be refuted by using this limited form of resolution.

*Definition.* A *transformational system* is a finite set of clauses each member of which is either a unit clause or a transformation. If $S$ is a transformational system, then $\mathscr{P}(S)$ is the set of CAFs which occur in the positive units of $S$, $\mathscr{N}(S)$ is the set of CAFs which occur in the negative units of $S$, and $\mathscr{T}(S)$ is the set of transformations in $S$.

We will call a resolution a *cross-resolution* if one resolvend is a clause containing no negative literals and the other resolvend is a clause containing at least one negative literal. In Robinson (1965a), it is shown that a set of clauses $S$ is unsatisfiable iff the empty clause can be derived by repeated cross-resolutions.

But if $S$ is a transformational system, then any cross-resolution of members of $S$ will either resolve a positive unit against a negative unit, yielding the empty clause, or a positive unit against a transformation, yielding another positive unit. Thus any refutation of a transformational system by cross-resolution will have the form shown in figure1 where the *p*s are positive units, the *t*s are transformations, and $n$ is a negative unit.

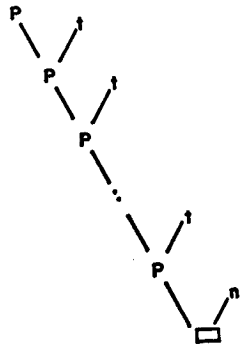



Figure 1

*Definition.* For any set $P$ of GAFs and any set $T$ of transformations, we define:

$$R_T(P) = \{r_t(A) \mid t \in T, A \in P\}$$
$$R_T^0(P) = P$$
$$R_T^{n+1}(P) = R_T(R_T^n(P)) \quad (n \geqslant 0).$$

If $S$ is a transformational system, it is evident that $\bigcup_{n=0}^{\infty} R_{\mathscr{T}(S)}^n(\mathscr{P}(S)) - \{\Omega\}$ is the set of (the members of) all positive units which can be derived from $S$ by cross-resolution. Moreover, a positive unit $\{A\}$ and a negative unit $\{\neg B\}$ will resolve to give the empty clause iff $A \sqcap B \neq \Omega$. Thus:

*Corollary* 7

A transformational system is unsatisfiable iff there exists an integer $n \geqslant 0$, and GAFs $A$ and $B$ such that $A \in R_{\mathscr{T}(S)}^n(\mathscr{P}(S))$, $B \in \mathscr{N}(S)$, and $A \sqcap B \neq \Omega$. Thus one can refute a transformational system by path-searching rather than tree-searching. In effect, the dyadic inference rule of resolution can be replaced by a finite set of monadic rules, one for each transformation.

The path-searching aspect is even more evident on the ground level. A transformational system $S$ is unsatisfiable iff the empty clause can be derived from some finite set of ground instances of $S$ by cross-resolution. Again, the refutation will have the form shown in figure 1, but now a positive unit $\{A\}$ will resolve against a transformation $\{\neg P, Q\}$ iff $A = P$. Thus:

*Corollary* 8

Let $S$ be a transformational system and $D$ be the directed graph whose set of nodes is the set of ground GAFs, and in which there is an arc from node $P$ to node $Q$ iff $\{\neg P, Q\}$ is a ground instance of some member of $\mathscr{T}(S)$. Then $S$ is unsatisfiable iff there is a path in $D$ from some ground instance of a member of $\mathscr{P}(S)$ to some ground instance of a member of $\mathscr{N}(S)$.

In the context of transformational systems, we can illustrate the notion of a 'super-consequence'. For a transformational system $S$, suppose that $P \subseteq \mathscr{P}(S)$ and $T \subseteq \mathscr{T}(S)$, Then a search procedure may eventually generate each member of the (usually infinite) set $S' = \bigcup_{n=0}^{\infty} R_T^n(P)$. We will call a clause $C$ a *super-consequence* of $P$ and $T$ if $C$ is a common generalization of $S'$. Usually $C$ itself will not be a valid consequence of $P$ and $T$, but if $C$ can be shown to be irrelevant (e.g., by subsumption or purity), then the generation of all members of $S'$ can be avoided.

The following theorem gives a method for computing super-consequences:

*Theorem* 10

Let $P$ be a finite set of GAFs, let $T$ be a finite set of transformations, and let:

$$C_0 = \sqcup P$$
$$C_{n+1} = C_n \sqcup \sqcup R_T(\{C_n\}) \quad (n \geqslant 0).$$

Then there is an integer $n$ such that $C_{n+1} \simeq C_n$. If $n_0$ is the least such integer, then $C_{n_0}$ is a common generalization of $\bigcup_{n=0}^{\infty} R_T^n(P)$.

*Proof.* The $C$s satisfy $C_0 \leqslant C_1 \leqslant C_2 \leqslant \ldots$. Thus, since there are no infinite ascending chains of GAFs, there is an integer $n$ such that $C_{n+1} \simeq C_n$. Let $n_0$ be the least such integer. Then by induction on $i$, $C_{n_0+1+i} \simeq C_{n_0}$ for all $i \geqslant 0$. Thus $C_n \leqslant C_{n_0}$ for all $n \geqslant 0$.

To complete the proof we will show, by induction on $n$, that $C_n$ is a common generalization of $R_T^n(P)$. The assertion is obvious for $n=0$. Assuming it is true for $n$, let $A$ be any member of $R_T^{n+1}(P)$. Then $A = r_t(B)$ for some $t \in T$ and $B \in R_T^n(P)$. Then $B \leqslant C_n$, so that $r_t(B) \leqslant r_t(C_n)$, by the first part of Theorem 9. Thus $A \leqslant r_t(C_n) \leqslant \sqcup R_T(\{C_n\}) \leqslant C_{n+1}$.

As an example, if $P = \{P(f(c)c)\}$ and $T = \{\{\neg P(x, y), P(f(f(y))x)\}\}$, then $n_0 = 1$, and $C_{n_0} \simeq P(f(x)x)$.

It is evident that the refutation of transformational systems is significantly simpler than the refutation of arbitrary sets of clauses. We conclude by showing that the problem is still non-trivial, in the precise sense that there is no decision procedure for transformational systems. This suggests that transformational systems may be a useful 'initial case' for the development of more efficient proof procedures.

Our proof will be accomplished by mapping a known unsolvable problem, the Post correspondence problem, into the decision problem for transformational systems.

*Definition.* A *correspondence problem* is a finite non-empty sequence of pairs, $\rho = (A_1, B_1), \ldots, (A_m, B_m)$, where each $A_i$ and each $B_i$ is a string over some finite set $V$ of characters. The problem $\rho$ is called *solvable* iff there exist integers $n \geqslant 0$, $i_0, \ldots, i_n$ such that $1 \leqslant i_j \leqslant m$ (for $0 \leqslant j \leqslant n$) and $A_{i_n} \ldots A_{i_0} = B_{i_n} \ldots B_{i_0}$, where the juxtaposition of string variables indicates string concatenation.

It is known (Floyd 1966, Post 1946) that there is no algorithm which will accept an arbitrary correspondence problem $\rho$ and determine whether $\rho$ is solvable.

In the following definition, corollary, and lemma, we assume that the vocabulary $V$ is fixed, that $\sigma$ is some one-to-one mapping from $V$ into a set of ground terms, and that $E$ is a constant, $F$ is a binary function symbol, $P$ is a binary predicate symbol, and $X$ and $Y$ are distinct variables.

*Definition.* Let $A = a_1 \ldots a_k$ be a string of length $k$ over $V$, and $T$ be a term. Then $\Sigma(A, T)$ denotes the term $F\sigma(a_1)F\sigma(a_2) \ldots F\sigma(a_k)T$.

*Corollary* 9

Let $A$ and $B$ be strings over $V$, $T$ be a term, and $\theta$ be a substitution. Then

$$\Sigma(A, T)\theta = \Sigma(A, T\theta)$$
$$\Sigma(A, \Sigma(B, E)) = \Sigma(AB, E)$$
$$\Sigma(A, E) = \Sigma(B, E) \text{ iff } A = B.$$

*Lemma* 4

Let $\rho = (A_1, B_1), \ldots, (A_m, B_m)$ be a correspondence problem, and let $S$ be the transformational system such that:

$$\mathscr{P}(S) = \{P\Sigma(A_i, E)\Sigma(B_i, E) \mid 1 \leqslant i \leqslant m\}$$
$$\mathscr{T}(S) = \{\{\neg PXY, P\Sigma(A_i, X)\Sigma(B_i, Y)\} \mid 1 \leqslant i \leqslant m\}$$
$$\mathscr{N}(S) = \{PXX\}.$$

Then $S$ is unsatisfiable iff $\rho$ is solvable.

An intuitive grasp of this lemma may be obtained by the following interpretation of $S$: $E$ denotes the empty string. Each ground term $\sigma(a)$ denotes the character $a$. The function $F(a, s)$ denotes the string obtained by adding the character $a$ to the beginning of the string $s$. The predicate $P(s, t)$ asserts that $s = A_{i_n} \ldots A_{i_0}$ and $t = B_{i_n} \ldots B_{i_0}$ for some $i_0, \ldots, i_n$. $\mathscr{P}(S)$ and $\mathscr{T}(S)$ are axioms which define $P$, and $\mathscr{N}(S)$ (whose negation occurs in $S$) is a theorem that $\rho$ is solvable.

*Proof.* We first show, by induction on $n$, that

$$R^n_{\mathscr{T}(S)}(\mathscr{P}(S)) = \bigcup_{i_0=1}^{m} \ldots \bigcup_{i_n=1}^{m} \{P\Sigma(A_{i_n} \ldots A_{i_0}, E)\,\Sigma(B_{i_n} \ldots B_{i_0}, E)\}.$$

The assertion is obvious for $n = 0$. Assuming it is true for $n$,

$$R^{n+1}_{\mathscr{T}(S)}(\mathscr{P}(S)) = \{r_{\{\neg PXY, P\Sigma(A_i, X)\Sigma(B_i, Y)\}}(x) \mid 1 \leqslant i \leqslant m \text{ and } x \in R^n_{\mathscr{T}(S)}(\mathscr{P}(S))\}$$
$$= \bigcup_{i_0=1}^{m} \ldots \bigcup_{i_{n+1}=1}^{m} \{r\}$$

where

$$r = r_{\{\neg PXY, P\Sigma(A_{i_{n+1}}, X)\Sigma(B_{i_{n+1}}, Y)\}}(P\Sigma(A_{i_n} \ldots A_{i_0}, E)\,\Sigma(B_{i_n} \ldots B_{i_0}, E))$$
$$= P\Sigma(A_{i_{n+1}}, X)\,\Sigma(B_{i_{n+1}}, Y)\{\Sigma(A_{i_n} \ldots A_{i_0}, E)/X,$$
$$\Sigma(B_{i_n} \ldots B_{i_0}, E)/Y\}.$$

Then by the first two parts of Corollary 9,

$$r = P\Sigma(A_{i_{n+1}}, \Sigma(A_{i_n} \ldots A_{i_0}, E))\,\Sigma(B_{i_{n+1}}, \Sigma(B_{i_n} \ldots B_{i_0}, E))$$
$$= P\Sigma(A_{i_{n+1}} \ldots A_{i_0}, E)\,\Sigma(B_{i_{n+1}} \ldots B_{i_0}, E)$$

which completes the induction.

By Corollary 7, $S$ is unsatisfiable iff there is some $n \geqslant 0$ and some $P\Sigma(A, E)\Sigma(B, E) \in R^n_{\mathscr{T}(S)}(\mathscr{P}(S))$, such that $P\Sigma(A, E)\Sigma(B, E) \sqcap PXX \neq \Omega$. But by the definition of $\sqcap$ and Corollary 9,

$$P\Sigma(A, E)\Sigma(B, E) \sqcap PXX \neq \Omega \text{ iff } \Sigma(A, E) = \Sigma(B, E) \text{ iff } A = B.$$

Thus $S$ is unsatisfiable iff $\rho$ is solvable.

Now suppose $D$ were a decision procedure for transformational systems. Then, given any correspondence problem $\rho$, we could use the mapping of Lemma 4 to convert $\rho$ into a transformational system $S$, and then determine whether $\rho$ is solvable by applying $D$ to $S$. Since it is known that this is impossible:

*Theorem* 11

There is no algorithm which will accept an arbitrary transformational system $S$ and determine whether $S$ is unsatisfiable.

In addition to establishing the non-triviality of transformational systems, this theorem is pertinent to Wos's unit preference strategy (Wos, Carson and Robinson 1964). Since cross-resolution is a special case of resolution, it is evident from figure 1 that a transformational system S is unsatisfiable iff the empty clause can be derived from $S$ by repeated resolutions in which at least one resolvend is a unit. Thus:

*Corollary* 10

There is no algorithm which will accept an arbitrary finite set of clauses $S$ and determine whether the empty clause can be derived from $S$ by repeated resolutions in which at least one resolvend is a unit.

In effect, there is no decision procedure for the unit section of the unit preference strategy.

**Acknowledgement**

This work was performed under the auspices of the United States Atomic Energy Commission.

## REFERENCES

Birkhoff, G. (1967) *Lattice Theory*. Amer. Math. Soc. Colloquium Publications, 25.

Floyd, R.W. (1966) *New Proofs of Old Theorems in Logic and Formal Linguistics*. Carnegie Institute of Technology.

Post, E.L. (1946) A Variant of a Recursively Unsolvable Problem. *Bull. Amer. Math. Soc.*, **52**, 264–8.

Reynolds, J.C. (1968) A Generalized Resolution Principle Based upon Context-Free Grammars, *Proc. IFIP Congress 1968*, *2*, 1405–11. Amsterdam: North Holland.

Robinson, J.A. (1965) A Machine-Oriented Logic Based on the Resolution Principle. *J. Ass. comput. Mach.*, **12**, 23–41.

Robinson, J.A. (1965a) Automatic Deduction with Hyper-Resolution. *Int. J. comput. Math.*, **1**, 227–34.

Wos, L., Carson, D. and Robinson, G. (1964) The Unit Preference Strategy in Theorem Proving. *Proc. AFIPS*, **26**, 615–21.

# 8

# A Note on Inductive Generalization

Gordon D. Plotkin
Department of Machine Intelligence and Perception
University of Edinburgh

In the course of the discussion on Reynolds' (1970) paper in this volume, it became apparent that some of our work was related to his, and we therefore present it here.

R.J.Popplestone originated the idea that generalizations and least generalizations of literals existed and would be useful when looking for methods of induction. We refer the reader to his paper in this volume for an account of some of his methods (Popplestone 1970).

Generalizations of clauses can also be of interest. Consider the following induction:

The result of heating this bit of iron to 419°C was that it melted.
The result of heating that bit of iron to 419°C was that it melted.
The result of heating any bit of iron to 419°C is that it melts.

We can formalize this as:

Bitofiron (bit 1) $\wedge$ Heated (bit 1,419) $\supset$ Melted (bit 1)
Bitofiron (bit 2) $\wedge$ Heated (bit 2,419) $\supset$ Melted (bit 2)
$(x)$ Bitofiron $(x)$ $\wedge$ Heated $(x, 419)$ $\supset$ Melted $(x)$

Note that both antecedents and conclusion can be expressed as clauses in the usual first-order language with function symbols. Our aim is to find a rule depending on the form of the antecedents which will generate the conclusion in this and similar cases. It will turn out that the conclusion is the least generalization of its antecedents.

We say that the literal $L_1$ is more general than the literal $L_2$ if $L_1\sigma = L_2$ for some substitution $\sigma$. For clauses we say that the clause $C_1$ is more general than the clause $C_2$ if $C_1$ subsumes $C_2$. Although the implication relationship might give interesting results, the weaker subsumption relationship allows a more manageable theory. For example, we do not even know whether implication between clauses is a decidable relationship or not.

A least generalization of some clauses or literals is a generalization which is less general than any other such generalization. For example, $P(g(x), x)$ is a least generalization of $\{P(g(a()), a()), P(g(b()), b())\}$ and $P(g(x), x)$

is a least generalization of $\{Q(x) \vee P(g(a()), a()), R(x) \vee P(g(b()), b())\}$. We give another, more complex example taken from a board game situation later in the paper. Our logical language is that of Robinson (1965). MacLane and Birkhoff (1967) is a good reference for our algebraic language.

## PRELIMINARIES

We will use the symbols $t, t_1, u, \ldots$ for terms, $L, L_1, M, \ldots$ for literals, $D, D_1, D, \ldots$ for clauses, $\varphi$ for a function symbol or a predicate symbol or the negation sign followed by a predicate symbol.

A *word* is a literal or a term. We will use the symbols $V, V_1, W, \ldots$ for words.

We denote sequences of integers, perhaps empty, by the symbols $I, J, \ldots$.

We say that $t$ is in the $I$th place in $W$ iff:

when $I = \langle\rangle$, $t = W$ or

when $I = \langle i_1, \ldots, i_n \rangle$, then $W$ has the form $\varphi(t_1, \ldots, t_m)$ and $i_1 \leqslant m$ and $t$ is in the $\langle i_2, \ldots, i_n \rangle$th place in $t_{i_1}$. For example, $x$ is in the $\langle\rangle$th place in $x$, the $\langle 2 \rangle$th place in $g(y, x)$ and in the $\langle 3, 2 \rangle$th place in $P(a, b, g(y, x))$.

Note that $t$ is never in the $\langle\rangle$th place in $L$. We say that $t$ is *in* $W$, if $t$ is in the $I$th place in $W$ for some $I$.

$W_1 \leqslant W_2$ (read '$W_1$ is more general than $W_2$') iff $W_1\sigma = W_2$ for some substitution $\sigma$. For example, $P(x, x, f(g(y))) \leqslant P(l(3), l(3), f(g(x)))$. We can take $\sigma = \{l(3)|x,\ x|y\}$.

$C_1 \leqslant C_2$ (read '$C_1$ is more general than $C_2$') iff $C_1\sigma \subseteq C_2$ for some substitution $\sigma$. $C_1 \leqslant C_2$ means that $C_1$ subsumes $C_2$ in the usual terminology. For example, $P(x) \vee P(f()) \leqslant P(f())$. We can take $\sigma = \{f()|x\}$.

In both cases, the relation $\leqslant$ is a quasi-ordering. We have chosen to write $L_1 \leqslant L_2$ rather than $L_1 \geqslant L_2$ as Reynolds (1970) does, because in the case of clauses, $\leqslant$ is 'almost' $\subseteq$. Further, if $L^1$ is the universal closure of $L$ and $L^2$ is the element of the Lindenbaum algebra corresponding to $L^1$, then we have $L_1 \leqslant L_2$ iff $L_1^1 \rightarrow L_2^1$ iff $L_1^2 \leqslant L_2^2$.

## WORDS

Suppose that we can show that a property holds for variables and constants, and that whenever it holds for $t_1, \ldots, t_n$ then it holds for $\varphi(t_1, \ldots, t_n)$. Then the property holds for all words. This method of proof is called induction on words.

We write $W_1 \sim W_2$ when $W_1 \leqslant W_2$ and $W_2 \leqslant W_1$. As $\leqslant$ is a quasi-ordering, this defines an equivalence relation. It is known that $W_1 \sim W_2$ iff $W_1$ and $W_2$ are alphabetic variants.

Two words are *compatible* iff they are both terms or have the same predicate letter and sign.

If $K$ is a set of words, then $W$ is a *least generalization* of $K$ iff:

1. For every $V$ in $K$, $W \leqslant V$.
2. If for every $V$ in $K$, $W_1 \leqslant V$, then $W_1 \leqslant W$.

It follows from 2 that if $W_1$, $W_2$ are any two least generalizations of $K$, then $W_1 \sim W_2$.

We can define also the least generalization as a product in the following category. The objects are the words and $\sigma$ is a morphism from $V$ to $W$ iff $V\sigma = W$ and $\sigma$ acts as the identity, $\varepsilon$, on variables not in $V$. $V$ is then a least generalization of $\{W_1, W_2\}$ iff it is a product of $W_1$ and $W_2$. We could also have defined the dual of the least generalization. This would just be the most general unification of Robinson (1965) and would be the coproduct in the above category. This approach was suggested in a personal communication by R. M. Burstall, but we have not followed it up to any degree.

*Theorem* 1

Every non-empty, finite set of words has a least generalization iff any two words in the set are compatible.

Let $W_1$, $W_2$ be any two compatible words. The following algorithm terminates at stage *3*, and the assertion made there is then correct.

*1.* Set $V_i$ to $W_i (i=1, 2)$. Set $\varepsilon_i$ to $\varepsilon (i=1, 2)$. $\varepsilon$ is the empty substitution.

*2.* Try to find terms $t_1, t_2$ which have the same place in $V_1$, $V_2$ respectively and such that $t_1 \neq t_2$ and either $t_1$ and $t_2$ begin with different function letters or else at least one of them is a variable.

*3.* If there are no such $t_1, t_2$ then halt. $V_1$ is a least generalization of $\{W_1, W_2\}$ and $V_1 = V_2$, $V_i\varepsilon_i = W_i (i=1, 2)$.

*4.* Choose a variable $x$ distinct from any in $V_1$ or $V_2$ and wherever $t_1$ and $t_2$ occur in the same place in $V_1$ and $V_2$, replace each by $x$.

*5.* Change $\varepsilon_i$ to $\{t_i|x\}\varepsilon_i (i=1, 2)$.

*6.* Go to *2.*

*Example.* We will use the algorithm to find a least generalization of

$$\{P(f(a(), g(y)), x, g(y)), P(h(a(), g(x)), x, g(x))\}.$$

Initially,

$$V_1 = P(f(a(), g(y)), x, g(y))$$
$$V_2 = P(h(a(), g(x)), x, g(x)).$$

We take $t_1 = y$, $t_2 = x$ and $z$ as the new variable. Then after *4*,

$$V_1 = P(f(a(), g(z)), x, g(z))$$
$$V_2 = P(h(a(), g(z)), x, g(z))$$

and after *5*,

$$\varepsilon_1 = \{y|z\}, \varepsilon_2 = \{x|z\}.$$

Next, we take $t_1 = f(a(), g(z))$, $t_2 = h(a(), g(z))$ and $y$ as the new variable. After *4* and *5*,

$$V_1 = P(y, x, g(z)) = V_2$$
$$\varepsilon_1 = \{f(a(), g(z))|y\}\{y|z\}$$
$$= \{f(a(), g(y))|y, y|z\}$$
$$\varepsilon_2 = \{h(a(), g(z))|y\}\{x|z\}$$
$$= \{h(a(), g(x))|y, x|z\}.$$

The algorithm then halts with $P(y, x, g(z))$ as the least generalization.

*Proof.* Evidently the compatibility condition is necessary. Let $\{W_1, \ldots, W_n\}$ be a finite compatible set of words. If $n=1$, then the theorem is trivial. Suppose that the algorithm works and that $\inf\{V, W\}$ is the result of applying it to $V$ and $W$. Then it is easy to see that

$$\inf\{W_1, \inf\{W_2, \ldots, \inf\{W_{n-1}, W_n\}\ldots\}\}$$

is a least generalization of the set. Hence we need only show that the algorithm works.

The rest of the proof proceeds as follows. In order to avoid a constant repetition of the conditions on $t_1, t_2$ given in 2, we say that $t_1$ and $t_2$ are *replaceable* in $V_1$ and $V_2$ iff they fulfil the conditions of *2.*

To show that the algorithm halts and that when it does $V_1 = V_2$, we define a *difference* function by *difference* $(V_1, V_2)$ = number of members of the set $\{I | \textit{if } t_1, t_2 \textit{ are both in the Ith place in } V_1, V_2 \textit{ respectively then they are replaceable in } V_1 \textit{ and } V_2\}$. We also denote by $V_1', V_2'$ the result of replacing $t_1$ and $t_2$ in $V_1, V_2$ by $x$ in the way described in *4.*

Lemma 1.2 then shows that every time a pair of replaceable terms is replaced the difference drops. Consequently by lemma 1.1 it will eventually become zero and when it does, lemma 1.1 shows that we must have $V_1 = V_2$ and the algorithm will then halt.

We still have to show that the replacements take us in the correct direction. First of all, $V_i' < V_i$ since by lemma 1.3, $V_i'\{t_i | x\} = V_i$. It is also immediate from this that when the algorithm halts, $V_i' \varepsilon_i = W_i$. Now suppose that $W$ is any lower bound of $\{W_1, W_2\}$. Then a lower bound $V$ is a product of $W_1, W_2$ if the diagram of figure 1 can always be filled in along the dotted line, so that it becomes commutative in a unique way.

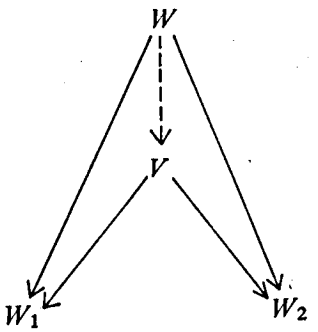



Figure 1

The category is the one defined above. In it there is either one or no morphisms between any two objects and hence it is not necessary in figure 1 to name the morphisms. Indeed, if a diagram can be filled in at all, it can be filled in commutatively and uniquely.

We show in lemma 1.4 that the diagram on figure 2 can be filled in commutatively.

Thus every time a replacement is made, the $V_i'$ are greater than any lower bound of $W_1$, $W_2$. Consequently when the algorithm halts, we have a product. We now give the statements and proofs of the lemmas.

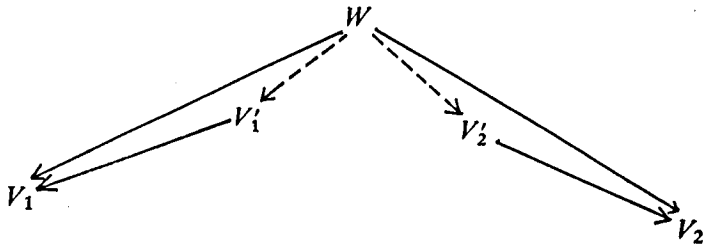



Figure 2

*Lemma* 1.1

If $V_1$ and $V_2$ are distinct compatible words, then there are $t_1$, $t_2$ which are replaceable in them.

*Proof.* By induction on words on $V_1$. If one of $V_1$, $V_2$ is a constant or a variable, or if they begin with different function symbols, then $V_1$, $V_2$ will do for $t_1$, $t_2$ respectively.

If $V_1$ is $\varphi(t_1^1, \ldots, t_n^1)$ and $V_2$ is $\varphi(t_1^2, \ldots, t_n^2)$, then for some $i$, $t_i^1 \neq t_i^2$ and by the induction hypothesis, applied to $t_i^1$, there are $u_1$, $u_2$ which are replaceable in $t_i^1$, $t_i^2$ and as $t_i^1$, $t_i^2$ have the same place in $V_1$, $V_2$ respectively, they are also replaceable in $V_1$, $V_2$.

*Lemma* 1.2

If $V_1$, $V_2$ are distinct compatible words, then *Difference* $(V_1', V_2') <$ *Difference* $(V_1, V_2)$.

*Proof.* By induction on words on $V_1$. If one of $V_1$ or $V_2$ is a variable or a constant then $t_1 = V_1$, $t_2 = V_2$ and $V_1' = V_2' = x$, so $0 =$ *Difference* $(V_1', V_2') < 1 =$ *Difference* $(V_1, V_2)$.

If $V_1$ is $f(v_1, \ldots, v_n)$ and $V_2$ is $g(u_1, \ldots, u_m)$ where $f \neq g$, then if $t_i = V_i (i = 1, 2)$, $0 =$ *Difference* $(V_1', V_2') <$ *Difference* $(V_1, V_2)$, by lemma 1.1; otherwise,

$$\textit{Difference}(V_1', V_2') = 1 + \sum_{i=1,\min(m,n)} \textit{Difference}(v_i', u_i')$$

$$< 1 + \sum_{i=1,\min(m,n)} \textit{Difference}(v_i, u_i)$$

(by induction hypothesis, since $m, n \neq 0$)

$$= \textit{Difference}\ (V_1, V_2).$$

In the remaining case where $V_1$ and $V_2$ both have the form $\varphi(t_1, \ldots, t_n)$, a similar but less complicated argument applies.

*Lemma* 1.3

$$V_i'\{t_i | x\} = V_i\ (i = 1, 2).$$

*Proof.* Since $V_i'$ is obtained from $V_i$ by replacing some occurrences of $t_i$ in $V_i$ by $x$, and since $x$ does not occur in $V_i$, substituting $t_i$ for $x$ in $V_i'$ will produce $V_i$. $(i = 1, 2)$.

*Lemma* 1.4

If $V_1$, $V_2$ are distinct compatible words and $V\sigma_i = V_i (i=1, 2)$, then there are $\sigma_1', \sigma_2'$ so that $V\sigma_i' = V_i' (i=1, 2)$.

*Proof.* It is convenient to denote by $f_i(u_1, u_2, t_1, t_2)$ the result of applying the operation of *4* to $u_1, u_2$ on $u_i (i=1, 2)$.

Let $\sigma_i = \{u_i^1 | y_1, \ldots, u_i^m | y_m\} (i=1, 2)$;

$v_i^j = f_i(u_1^j, u_2^j, t_1, t_2) (i=1, 2; j=1, m)$;

$\sigma_i' = \{v_i^1 | y_1, \ldots, v_i^m | y_m\} (i=1, 2)$.

By lemma 1.3, $\sigma_i = \sigma_i' \{t_i | x\} (i=1, 2)$. We show by induction on $V$, that: if $V$, $V_1$, $V_2$ are such that $V\sigma_i = V_i (i=1, 2)$, then $V\sigma_i' = V_i' (i=1, 2)$.

Suppose that $V$ is a constant, then $V = V_1 = V_2$ and the result is trivial. Suppose that $V$ is a variable, $y$. If $y \neq y_i$ for $i=1, m$ then $y = V = V_1 = V_2$ and the result is again trivial. If $y = y_i$ say, then $V\sigma_i' = f_i(V_1, V_2, t_1, t_2) = V_i'$. Suppose $V$ is $\varphi(u_1, \ldots, u_n)$ then if $V_i = \varphi(w_1^i, \ldots, w_n^i)$,

$$V\sigma_i' = \varphi(u_1\sigma_i', \ldots, u_n\sigma_i') = \varphi(w_1^i, \ldots, w_n^i) \text{ (by the induction hypothesis)} = V_i'.$$

This concludes the proof.

The next lemma is used in the proof of the existence of least generalizations of clauses.

*Lemma* 2

Let $K = \{W_i | i=1, n\}$ be a set of words with a least generalization $W$ and substitutions $\mu_i (i=1, n)$ so that $W\mu_i = W_i (i=1, n)$.

*1.* If $t$ is in $W$ then $t$ is a least generalization of $\{t\mu_i | i=1, n\}$.

*2.* If $x, y$ are variables in $W$ and $x\mu_i = y\mu_i (i=1, n)$ then $x = y$.

*Proof. 1.* Evidently, $t$ is a generalization of $\{t\mu_i | i=1, n\}$. Suppose $u$ is any other and that $u\lambda_i = t\mu_i (i=1, n)$. Let $u\tau$ be an alphabetic variant of $u$ such that $u\tau$, $W$ have no common variables. Let $W'$ be $W$, but with $t$ replaced by $u\tau$ wherever $t$ occurs in $W$. Then $\tau^{-1}\lambda_i \cup \mu_i$ is defined – this follows from the construction of $\tau$ – and $W'(\tau^{-1}\lambda_i \cup \mu_i) = W_i (i=1, n)$. Hence there is a $v$ so that $W'v = W$, as $W$ is a least generalization of $\{W_i | i=1, n\}$. Hence $u(\tau v) = (u\tau)v = t$. Hence, $t$ is a least generalization of $\{t\mu_i | i=1, n\}$.

*2.* Suppose that $y \neq x$. Let $W' = W\{y | x\}$. Then $W'$, $W$ are not alphabetic variants, but $W \leqslant W'$. Let $W = W[x, y, y_3, \ldots, y_m]$, where $x, y, y_3, \ldots, y_m$ are the distinct variables of $W$. We have,

$$\begin{aligned} W_i = W\mu_i &= W[x\mu_i, y\mu_i, y_3\mu_i, \ldots, y_m\mu_i] \\ &= W[y\mu_i, y\mu_i, y_3\mu_i, \ldots, y_m\mu_i] \text{ (by hypothesis)} \\ &= W[y, y, y_3, \ldots, y_m]\mu_i \\ &= W'\mu_i \text{(by construction)}. \quad (i=1, \text{n}). \end{aligned}$$

This contradicts the fact that $W$ is a least generalization of $\{W_i | i=1, n\}$. Hence $y = x$.

This completes the proof.

## CLAUSES

Just as we did with words, we write $C \sim D$, when $C \leqslant D$ and $D \leqslant C$. This defines an equivalence relation. We also say that $C$ is a least generalization of a set of clauses, $S$, when:

*1.* For every $E$ in $S$, $C \leqslant E$.

*2.* If for every $E$ in $S$, $D \leqslant E$, then $D \leqslant C$.

Any two least generalizations of $S$ are equivalent under $\sim$. However, when $C_1 \sim C_2$, $C_1$ and $C_2$ need not be alphabetic variants. For example, take

$$C_1 = \{P(x), P(f())\}; \quad C_2 = \{P(f())\}.$$

It turns out that there is a reduced member of the equivalence class, under $\sim$, of any clause. This member is unique to within an alphabetic variant. $C$ is *reduced* iff $D \subseteq C$, $D \sim C$ implies that $C = D$. In other words, $C$ is reduced iff it is equivalent to no proper subset of itself.

*Lemma* 3

If $C\mu = C$, then $C$ and $C\mu$ are alphabetic variants.

*Proof.* We regard $C$ as a set ordered by $\leqslant$, and suppose without loss of generality that $\mu$ acts as the identity on variables not in $C$. Let $L$ be in $C$. The sequence $L = L\mu^0, L\mu^1 = L\mu, L\mu^2, \ldots$ is increasing relative to $\leqslant$. As $C$ is finite and all members of the sequence are in $C$, it follows that for some $i$, $L\mu^i \sim L\mu^j (j \geqslant i)$. Hence for some $N$ and for all $L$ in $C$, $L\mu^N \sim L\mu^{N+1} (i \geqslant 0)$. As $C\mu = C$, there is an $M$ in $C$ so that $M\mu^N = L$, given $L$ in $C$. Hence, $L = M\mu^N \sim M\mu^{N+1} = L\mu$, and so $\mu$ maps variables into variables. But as $C\mu = C$, $C$ and $C\mu$ have the same number of variables. Hence $\mu$ maps distinct variables of $C$ to distinct variables of $C\mu$, and so $C$ and $C\mu$ are alphabetic variants, thus completing the proof.

*Theorem* 2

If $C \sim D$, and $C$ and $D$ are reduced, then they are alphabetic variants. The following algorithm gives a reduced subset, $E$, of $C$ such that $E \sim C$.

*1.* Set $E$ to $C$.

*2.* Find an $L$ in $C$ and a substitution $\sigma$ so that $E\sigma \subseteq E \setminus \{L\}$.

If this is impossible, stop.

*3.* Change $E$ to $E\sigma$ and go to *2.*

(It is necessary to be able to test for subsumption in order to carry out stage *2.* Robinson (1965) gives one way to do this.)

*Proof.* As $C \sim D$, there are $\mu, \nu$ so that $C\mu \subseteq D$, $D\nu \subseteq C$. Hence, $C\mu\nu \subseteq C$. But $C$ is reduced so $C\mu\nu = C$. Hence by lemma 3, $\mu\nu$ maps the variables of $C$ into the variables of $C$ in a 1–1 manner. Hence $C$ and $D$ are alphabetic variants.

The algorithm halts at stage *2*, since the number of literals in $E$ is reduced by at least one at stage *3* and so if it does not halt before then, it will halt when this number is 1. If $C = \emptyset$, then it will halt on first entering *2.*

There is always a $\mu$ so that $C\mu \subseteq E$. For at stage *1*, take $\mu = \varepsilon$. If one has such

a $\mu$ before stage *2*, then $\mu\sigma$ will be one after stage *2*. Hence, when the algorithm halts, $C\mu \subseteq E$ and $E \subseteq C$, and then $C \sim E$.

Suppose $E$ is not reduced at termination. Then there is a proper subset $E'$ of $E$ so that $E' \sim E$. So there is a $\sigma$ such that $E\sigma \subseteq E'$ .Pick $L$ in $E \setminus E'$. Then $E\sigma \subseteq E' \subseteq E \setminus \{L\}$. This contradicts the fact that the algorithm has terminated and completes the proof.

This theorem is useful as our method of producing least generalizations of clauses tends to give clauses with many literals which may often be substantially reduced by the above procedure.

Let $S = \{C_i | i = 1, n\}$ be a set of clauses.

A set of literals, $K = \{L_i | i = 1, n\}$ is a *selection* from $S$ iff $L_i \in C_i (i = 1, n)$.

We can now state the main theorem.

*Theorem* 3

Every finite set, $S$, of clauses has a least generalization which is not $\emptyset$ iff $S$ has a selection. If $C_1$ and $C_2$ are two clauses with at least one selection, then the following algorithm gives one of their least generalizations.

Let $S = \{C_1, C_2\}$, and let the selections from $S$ be $\{L_l^1, L_l^2\} (l = 1, n)$ where $L_l^j$ is in $C_j$. Suppose that $L_l^j = (\pm)P_l(t_{1l}^j, \ldots, t_{k_l l}^j)$, where $(\pm)P$ is either $P_l$ or $\bar{P}_l$. Let $f_l$ be a function letter with $k_l$ places $(l = 1, n)$, and let $P$ be a predicate letter with $n$ places. Let $M_j$ be the literal

$$P(f_1(t_{11}^j, \ldots, t_{k_1 1}^j), \ldots, f_n(t_{1n}^j, \ldots, t_{k_n n}^j)) \; (j = 1, 2).$$

Find the least generalization of $\{M_1, M_2\}$ by the method of theorem 1. Suppose that this is

$M = P(f_1(u_{11}, \ldots, u_{k_1 1}), \ldots, f_n(u_{1n}, \ldots, u_{kn}))$. Let $C$ be the clause

$$\{(\pm P)_i(u_{1i}, \ldots, u_{k_i i}) | i = 1, n\}.$$

Then $C$ is a least generalization of $S = \{C_1, C_2\}$.

Evidently, it is not necessary in any actual calculation to change any $P_l$ to the corresponding $f_l$.

*Proof.* We begin by showing that the algorithm works. By theorem 1, there are $v_i (i = 1, 2)$ so that $Mv_i = M_i$. From lemma 2, it follows that $f_i(u_{1i}, \ldots, u_{k_i i})$ is a least generalization of $\{f_i(t_{1i}^j, \ldots, t_{k_i i}^j) | j = 1, 2\}$.

Hence $L_i = (\pm)P_i(u_{1i}, \ldots, u_{k_i i})$ is a least generalization of $\{L_i^1, L_i^2\} (i = 1, n)$.

Hence $Cv_i = \bigcup_{l=1}^{n} \{L_l^i\} \subseteq C_i$ and so $C$ is a generalization of $\{C_1, C_2\}$.

Note also that by lemma 2, if $xv_i = yv_i (i = 1, 2)$, then $x = y$.

Now suppose that $E$ is any generalization of $\{C_1, C_2\}$. We show that $E \leqslant C$. Let $\alpha_i$ be chosen so that $E\alpha_i \subseteq C_i$. $(i = 1, 2)$, and let $E = \{M_1, \ldots, M_m\}$. $\{M_p\alpha_i | i = 1, 2\}$ will be a selection, say $\{L_{p'}^1, L_{p'}^2\}$. Consequently, $M_p \leqslant L_{p'}$, the corresponding least generalization of the selection, and there is a $\beta_p$ so that $M_p\beta_p = L_{p'}$.

Hence, if $\bigcup_{p=1}^{m}\beta_p$ exists, $E(\cup\beta_p)\subseteq C$ and we will have finished. Now $\cup\beta_p$ exists precisely if, whenever $x$ is an individual variable in $M_{p_1}$ and $M_{p_2}$ $(p_1\neq p_2)$ then $x\beta_{p_1}=x\beta_{p_2}$. Let the notation $A\xrightarrow{\alpha}B$ ($A$, $B$ literals; $\alpha$ a substitution) mean $A\alpha=B$. We have shown that the relationships described by figure 3 hold:

$(i,k=1,2)$

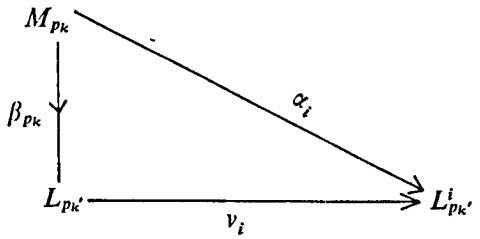



Figure 3

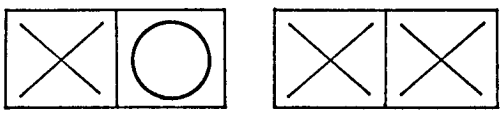
Now $x\beta_{p_k}$ is a term in $L_{p_k'}(k=1,2)$, which is a least generalization of $\{L^1_{p_k'}, L^2_{p_k'}\}$. Hence, by lemma 2, $x\beta_{p_k}$ is a least generalization of $\{x\beta_{p_k}v_1, x\beta_{p_k}v_2\}=\{x\alpha_1, x\alpha_2\}$ from the diagram, $(k=1,2)$.

Consequently, $x\beta_{p_1}, x\beta_{p_2}$ are alphabetic variants. Let $x_1, x_2$ be variables having the same place in $x\beta_{p_1}, x\beta_{p_2}$ respectively. Then as $x\beta_{p_1}v_i=x\beta_{p_2}v_i=x\alpha_i(i=1,2)$, it follows that $x_1v_i=x_2v_i(i=1,2)$. Hence by our note on the properties of the $v_i(i=1,2)$ at the beginning of the proof, $x_1=x_2$. Hence $x\beta_{p_1}=x\beta_{p_2}$, and $\cup\beta_p$ exists and $E\leqslant C$ and so $C$ is a least generalization of $\{C_1, C_2\}$.

Next, we note that $C$ is not empty and indeed if a particular combination of sign and predicate letter occurs in a selection from $\{C_1, C_2\}$ then it occurs in $C$. Suppose that $S=\{C_i\}(i=1,q)$ is a finite set of clauses. If $C\,(\neq\emptyset)$ is a generalization of $S$, there are $\alpha_i$ so that $C\alpha_i\subseteq C_i$, and if $L$ is in $C$, $\{L\alpha_i\,|\,i=1,q\}$ is a selection. On the other hand, $\emptyset$ is a generalization of $S$. Hence if $S$ has no selection its only, and hence its least, generalization is $\emptyset$. Otherwise, by nesting infs as in the proof of theorem 1, we can find a least generalization of $S$ which, by the note at the beginning of this paragraph, will not be $\emptyset$. This completes the proof.

## AN EXAMPLE

Suppose that some two-person game is being played on a board with two squares, 1() and 2() and that the positions in figure 4 are won positions for the first player:



Position $p_1$()  Position $p_2$()

Figure 4

1() is the name of the left hand side square, and 2() of the right hand square; $p_1()$ and $p_2()$ are the names of the positions and $O()$, $X()$ are the names of the marks $O$, $X$. We describe the fact that these positions are wins by means of the following two clauses:

1. $\overline{Occ}(1(), X(), p_1()) \vee \overline{Occ}(2(), O(), p_1()) \vee Win(p_1())$.
2. $\overline{Occ}(1(), X(), p_2()) \vee \overline{Occ}(2(), X(), p_2()) \vee Win(p_2())$.

The course of the calculation is indicated as follows:

$\overline{Occ}(1(), X(), p_1())\ \overline{Occ}(1(), X(), p)$
$\overline{Occ}(1(), X(), p_2())\ \overline{Occ}(1(), X(), p)$

$\overline{Occ}(1(), X(), p_1())\ \overline{Occ}(1(), X(), p)\ \overline{Occ}(n_1, X(), p)$
$\overline{Occ}(2(), X(), p_2())\ \overline{Occ}(2(), X(), p)\ \overline{Occ}(n_1, X(), p)$

$\overline{Occ}(2(), O(), p_1())\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(n_2, O(), p)\ \overline{Occ}(n_2, x, p)$
$\overline{Occ}(1(), X(), p_2())\ \overline{Occ}(1(), X(), p)\ \overline{Occ}(1(), X(), p)\ \overline{Occ}(n_2, X(), p)\ \overline{Occ}(n_2, x, p)$

$\overline{Occ}(2(), O(), p_1())\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(2(), O(), p)\ \overline{Occ}(2(), x, p)$
$\overline{Occ}(2(), X(), p_2())\ \overline{Occ}(2(), X(), p)\ \overline{Occ}(2(), X(), p)\ \overline{Occ}(2(), X(), p)\ \overline{Occ}(2(), x, p)$

$Win(p_1())$ $\quad$ $Win(p)$
$Win(p_2())$ $\quad$ $Win(p)$

As indicated above, we have not replaced predicate symbols by function symbols, and we have left $P$ implicit. Each vertical column displays $M_1$ and $M_2$ at an instance of stage *2* of the algorithm of theorem 1. In a given column, the pairs of literals are corresponding arguments in $M_1$ and $M_2$. We find $t_1$ and $t_2$ by searching through $M_1$ and $M_2$ from left to right. As soon as two literals have become the same in a column, we do not mention them in subsequent columns.

Thus the least generalization is:

$\overline{Occ}(1(), X(), p) \vee \overline{Occ}(n_1, X(), p) \vee \overline{Occ}(n_2, x, p) \vee \overline{Occ}(2(), x, p) \vee Win(p)$.

We use the algorithm of theorem 2. We can take $L = \overline{Occ}(n_1, X(), p)$, $\sigma = \{1() | n_1\}$. This gives

$C\sigma = \overline{Occ}(1(), X(), p) \vee \overline{Occ}(n_2, x, p) \vee \overline{Occ}(2(), x, p) \vee Win(p)$.

Next, we can take $L = \overline{Occ}(n_2, x, p)$ and $\sigma = \{2() | n_2\}$ and obtain

$C\sigma = \overline{Occ}(1(), X(), p) \vee \overline{Occ}(2(), x, p) \vee Win(p)$.

The algorithm stops at this point. The final clause says that if a position has an $X$ in hole 1 and hole 2 has something in it, then the position is a win, which, given the evidence, is fairly reasonable.

We leave it to the reader to verify that the conclusion of the inductive argument given at the beginning of this note is indeed a least generalization of its antecedents. The main computational weakness in the method for finding a reduced least generalization lies in that part of the reducing algorithm which requires a test for subsumption. For suppose that we are looking for the inf of two clauses each with nine literals in a single predicate letter (this can arise in descriptions of tic–tac–toe, say); there will be at least eighty-one literals in the raw inf, and we will have to try to tell whether or not a clause of eighty-one literals subsumes one of eighty.

## FURTHER RESULTS

We give without proof some further results obtained on the algebraic nature of the $\leqslant$ relation between clauses.

Let $[C]$ denote the equivalence class under $\sim$ of $C$. We say that $[C] \leqslant [D]$ iff $C \leqslant D$. It is easily seen that this is a proper definition.

The set of equivalence classes forms a lattice with the lattice operations given by:

$$[C] \sqcap [D] = [\inf \{C, D\}]$$
$$[C] \sqcup [D] = [C\xi \cup D],$$

where $\xi$ is a substitution which standardizes the variables of $C$ and $D$ apart.

This lattice is not modular. It has an infinite strictly ascending chain $[C_i]$ $(i \geqslant 1)$ where

$$C_1 = \{P(x_0, x_1)\}; \quad C_i = C_{i-1} \cup \{P(x_{i-1}, x_i)\}.$$

This chain is bounded above by $\{P(x, x)\}$.

There is also a rather complicated infinite strictly descending chain, $[C_i]$ $(i \geqslant 1)$ with the following properties:

1. No literal in any $C_i$ contains any function symbols.
2. The clauses are all formed from a single binary predicate letter and a single unary one.

Any infinite descending chain is bounded below by $\emptyset$. We hope to publish the proofs elsewhere.

### Acknowledgements

I should like to thank my supervisors R.J. Popplestone and R.M. Burstall for all the different kinds of help they gave me. Particular thanks are due to Dr Bernard Meltzer, without whose encouragement this paper would not have been written. The work was supported by an SRC research studentship.

## REFERENCES

MacLane, S. & Birkhoff, G. (1967) *Algebra*. New York: Macmillan.

Popplestone, R.J. (1970) An experiment in automatic induction. *Machine Intelligence 5* pp. 203–15 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Reynolds, J.C. (1970) Transformational systems and the algebraic structure of atomic formulas. *Machine Intelligence 5* pp. 135–52 (eds Meltzer, B. & 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.

# 9

# Power Amplification for Theorem-Provers

B. Meltzer
Metamathematics Unit
University of Edinburgh

## INTRODUCTION

The development of automatic theorem-provers has reached an interesting stage, where on the one hand new, more powerful methods are required and at the same time current methods are finding new important and unexpected applications. This article deals with both these matters.

The first part gives a criticism of present methods, as well as suggestions of how, building on schemes of the resolution type, one might increase their power.

The second part gives an account of ideas and initial experiments on a new extension of theorem-provers, namely to programs which will do induction; it is interesting that theorem-provers which were originally developed for the purpose of proving mathematical theorems are now finding application in such different fields as information retrieval (Darlington 1969), robotics (Green 1969), automatic writing of programs (Waldinger and Lee 1969) and now – one of the most important functions of any comprehensive artificial intelligence – induction.

## PART I

### Present methods and perspectives

The central method employed today for theorem-proving, Resolution (Robinson 1965a), with all its rich development of strategies and ordering principles (Andrews 1969, Kowalski 1970, Kowalski and Hayes 1969, Loveland (to be published), Meltzer 1968, Robinson 1965b, Slagle 1967, Wos *et al.* 1964, Wos *et al.* 1965), as well as new inference systems such as paramodulation (Robinson and Wos 1969), are designed within the framework of the lower predicate calculus, and depend on and have been motivated by the completeness theorem of that calculus. But the completeness theorem is not the whole of mathematics. Not only is this indicated by the incompleteness

results of Gödel and others for formal number theory and other parts of mathematics, but also by the more practical consideration that the methods developed in programs so far have used almost exclusively only basic *atomic* predicates and very short steps of inference. It is therefore not in the least surprising that even with the greatly improved strategies and ordering principles which have been discovered and used, the discipline of proving mathematical theorems on computers appears to have reached a plateau: theorems much more difficult or complex than ones already proved are not likely to come within their compass. A human mathematician would certainly not get very far in the study of group theory or number theory or analysis, if he confined himself to thinking and talking entirely in terms of simple atomic predicates like set-membership or the usual 3-place one of programmed elementary group theory. Neither would or does a theorem-proving program if it incorporates such constraints – *and*, in addition, is allowed to use for inference only binary resolution. It is as if one were asked to build the Forth Bridge, given only the constituent atoms of the metals and other materials required!

To increase their power, theorem-provers will need to employ 'higher-level' predicates and 'larger' inference steps. They will also need to make efficient use of lemmas, i.e., theorems already proved in the discipline concerned — the latter activity involving the crucial issue of criteria of *relevance*, already being actively studied in work on information retrieval (Darlington 1969, Green 1969).

### Macro predicates

One approach to the use of more powerful concepts is resort to second-order logic, as done in a piecemeal way by Darlington (1968), or omega-order logic, as proposed by Robinson (1969). However, within the framework of the lower predicate calculus and the resolution method, one can *define* new macro predicates in terms of atomic ones and attempt to carry out one's proofs by their use.

For instance, to take a simple example (cf. Hilbert and Ackermann 1950), suppose from the hypothesis, 'At most one straight line passes through two different points', the program is to prove 'Two different straight lines have not more than one point in common.' To make the formulae less clumsy, assume two *sorts* of individual variables in the formal system, one for points and one for lines; $L(x, g)$ is a 2-place predicate to be interpreted as '*point* $x$ on *line* $g$', and $E$ is the 2-place predicate of identity. The hypothesis then takes the form:

$$(x)(y)\{\bar{E}(x,y) \supset [(g)(h)\{L(x,g)\&L(y,g)\&L(x,h)\&L(y,h) \supset E(g,h)\}]\}$$

and the conclusion:

$$(g)(h)\{\bar{E}(g,h) \supset [(x)(y)\{L(x,g)\&L(y,g)\&L(x,h)\&L(y,h) \supset E(x,y)\}]\}.$$

The set of clauses to be proved unsatisfiable, if one introduces the Skolem constants, $x_0$, $y_0$, $g_0$ and $h_0$ is:

$$E(x, y)\, E(g, h)\, \bar{L}(x, g)\, \bar{L}(x, g)\, \bar{L}(y, g)\, \bar{L}(x, h)\, \bar{L}(y, h)$$
$$\bar{E}(g_0, h_0)$$
$$\bar{E}(x_0, y_0)$$
$$L(x_0, g_0)$$
$$L(y_0, g_0)$$
$$L(x_0, h_0)$$
$$L(y_0, g_0).$$

If however one had *defined* and used a 4-place 'macro' predicate $P(x, y, g, h) = L(x, g) \,\&\, L(y, g) \,\&\, L(x, h) \,\&\, L(y, h)$, one would have had only the four shorter clauses:

$$E(x, y)\, E(g, h)\, \bar{P}(x, y, g, h)$$
$$\bar{E}(g_0, h_0)$$
$$\bar{E}(x_0, y_0)$$
$$P(x_0, y_0, g_0, h_0),$$

which would require correspondingly less computing to generate the empty clause. The meaning of $P(x, y, g, h)$ is 'the points $x$ and $y$ are on the lines $g$ and $h$'.

This naïve example is a paradigm of what necessarily happens in any mathematical theory if it is to develop much power. So for example in the development of elementary group theory, besides the atomic 3-place predicate $P(x, y, z)$ meaning '$z$ is the product of $x$ and $y$', one needs quite soon to define and use the 2-place macro predicate

$$C(x, y) = (Ez)(u)(\bar{P}(z, x, u) \vee P(u, g(z), y))$$

meaning '$x$ is conjugate to $y$'. Here $g(z)$ is the group inverse function.

How is the program (or for that matter how are we) to decide what new predicates to define and use? The number that could be created are legion, and at this stage one can only suggest some of the criteria of choice that may be important: (1) The macro predicate is contained in the statement of the theorem to be proved. (2) It is contained in the statement of a lemma in store, which on some other criterion is relevant to the proof of the theorem. (3) It has been successfully used in the proof of another theorem similar to the theorem to be proved. (4) As in the simple geometrical example above, in some formal expression of the premises and conclusion, the more or less complex structure of atomic-type predicates concerned appears in a number of places.

Of course, once the data have been transformed by the use of macro predicates, even if – as in the above example – one or more atomic predicates have been completely eliminated thereby, there is no guarantee that even though the original set is unsatisfiable a method like resolution will now be able to prove this for the transformed set. What one therefore envisages is a

program which can carry out successive proof searches in a hierarchical way. If with a full load of macro predicates it fails to find a proof, it then 'unscrambles' these predicates (one at a time, say) i.e., replaces a predicate by its definiens, trying out the proof procedure each time. The completeness theorem ensures that it must, in principle, find a proof. The structure envisaged is hierarchical, because of course in the development of the corpus of proofs and lemmas, new macro predicates may be formed in terms not only of the basic atomic ones but also of already defined macro ones, present in the system.

**Larger inference steps**

The use of high-level predicates is associated with the question of the use of larger inference steps. Although most working theorem-provers use binary resolution, i.e., resolving on two clauses (or – more rarely – one), there have been proposals in the literature for using larger inference steps, which in effect process more than two clauses; for example, hyper-resolution (Robinson 1965b) and 'clash' resolution (Robinson 1967) are methods of this kind known to be refutation-complete. Since these particular methods do not appear to have 'caught on' in the design of theorem-provers, and since one would surely expect that more powerful programs will need to incorporate larger inference steps, it is worth considering this question in a more general way.

If one has a logical system with some basic inference machinery, the natural way to effect larger unitary inferences is to employ metatheorems (cf. Pitrat 1966). For instance, suppose in the resolution context we wished to use the metatheorem: from $\mathscr{A}$ and $\mathscr{A} \supset \mathscr{B}$ infer $\mathscr{B}$, where $\mathscr{A}$ and $\mathscr{B}$ are in general not unit clauses but may each be a conjunction of clauses of any length. Confining consideration, for present purposes, to ground clauses only, consider a few examples. Suppose $\mathscr{A}$ and $\mathscr{B}$ are each single clauses with respectively $n$ and $m$ literals, then the step is from the $n+1$ clauses

$$A_1\, A_2 \ldots A_n$$
$$\bar{A}_1\, B_1 \ldots B_m$$
$$\bar{A}_2\, B_1 \ldots B_m$$
$$\vdots$$
$$\bar{A}_n\, B_1 \ldots B_m$$

to infer the clause

$$B_1\, B_2 \ldots B_m.$$

This is a 'merged' clash resolvent. But this kind of inference quite soon breaks out of the framework of clash resolution. For example, if $\mathscr{A}$ consists of the two clauses $\{P, Q\}$ and $\{R, S\}$, and $\mathscr{B}$ is the unit clause $\{B\}$, then the step is from the six clauses

$P\ Q$
$R\ S$
$\bar{P}\ \bar{R}\ B$
$\bar{P}\ \bar{S}\ B$
$\bar{Q}\ \bar{R}\ B$
$\bar{Q}\ \bar{S}\ B$

to infer unit clause

$B$.

This is more general than a clash. In general this kind of inference will yield not a single clause but a conjunction of clauses, as may be seen for instance by interchanging the roles of $\mathscr{A}$ and $\mathscr{B}$ in the above example.

Similarly a metatheorem like the transitivity of implication, from $\mathscr{A} \supset \mathscr{B}$ and $\mathscr{B} \supset \mathscr{C}$ to infer $\mathscr{A} \supset \mathscr{C}$ – it is easily seen – includes as special cases binary resolution, multiple resolutions, clashes, multiple clashes as well as new and larger inferences.

How is one to implement such powers in theorem-provers? This is a difficult question to answer in general terms, because it cannot be known without a great deal of case study which are the most useful metatheorems to use in this way. But one may be guided by examining the practice of human mathematicians and noticing which metatheorems are most useful to them. The probability is that the choice will prove to be task-dependent, so that one might have stored in the machine an armoury of such macro-inference rules and – on the basis of experience – select for each problem type a subset of them.

It is of some interest to note that the use of metatheorems for inference is related to the previously discussed issue of the introduction of higher-level concepts. For instance, using the metatheorem, 'from $\mathscr{A}$ and $\mathscr{A} \supset \mathscr{B}$ infer $\mathscr{B}$' can be looked upon as being merely binary resolution with $\mathscr{A}$ and $\mathscr{B}$ *defined* as high-level propositional formulae, i.e., higher than atomic predicates.

#### General remarks

The proposal then is to add to theorem-provers the introduction by definition of macro predicates and propositions, the use of macro inference rules and the exploitation of a library of lemmas.

In a perhaps trivial sense the refutation-completeness of the proof system need not be impaired by these additions, if one uses the hierarchical unscrambling of the high-level predicates previously discussed and retains the basic rules, e.g., resolution, as part of the inference machinery. But other more interesting and useful completeness questions arise as to whether there are combinations of certain macro predicates and propositions with certain inference types which are refutation-complete, even without unscrambling. The major issue, however, for the successful use of these ideas is that of *relevance* in the choice not only of lemmas, but also of macro predicates and rules of inference.

## PART 2

### A hypothetico-deductive approach to programming induction

Perhaps the major task of intelligence, whether artificial or natural, is forming and checking out on theories. The checking is a deductive activity, consisting mainly of testing for consistency and compatibility with the known facts and secondarily of testing for redundancy of hypotheses; these functions can be carried out by theorem-provers now in existence provided the hypotheses and facts are couched in the appropriate logical language.

How are the theories to be formed? One of the basic operations involved is abstraction and generalization from the facts, and the ideas and experiments described below are concerned with this activity. Left on the side for the time being is the activity of generating new concepts for the explanation of the facts.

The lower predicate calculus, some form of which is used in most theorem-provers, is very well suited for carrying out such abstractions. For example from a *fact* expressed by a 3-place relation $P(a, b, c)$ one can immediately obtain various generalizations by abstracting on the argument constants and prefixing quantifiers; for example:

$$(x)P(x, b, c)$$
$$(y)(x)P(x, y, c).$$

What is therefore proposed is a system – let us call it Induc – which, given ground facts about some domain or domains of objects, first generates in the above manner generalizations of these facts. These constitute the axioms of a proposed theory. The theorem-prover is used to test the consistency of the theory and its compatibility with other available facts which have not been abstracted upon. If this test is passed the theory is *adequate*, though it may be 'cleaned up' if required by removing redundant axioms, the existence of which can also be tested for by means of the theorem-prover. If the theory is inconsistent or conflicts with the facts, the print-out of the proof by the theorem-prover will reveal which axioms may be responsible, and other combinations of axioms from the store of available generalizations are then tried out as a new proposed theory. When a theory has been obtained which passes the tests, it may be used for prediction and retained until such time as new facts force its modification.

Every theorem-prover has limited deductive power, due to there being an upper bound to the amount of data processing possible in a given time. In the case of refutation-complete methods like resolution, one can specify precisely, e.g., by an integer parameter, the level $l$ of deductive power of the programme. For example, in a saturation strategy, $l$ would be the depth of resolution allowed, a measure of the *length* of proof allowed. Such methods have the desirable property that if a valid theorem is not proved by a program of level $l$, there is a level $l' > l$ at which the program would find a proof;

furthermore $l$ is usually a parameter of the program which can be readily changed.

The notion is therefore introduced of a *theory of level l*, i.e., the theory is reliable as regards consistency, compatibility with the facts and redundance of axioms only to the degree that the restriction of the theorem-prover to working at level $l$ has not led to wrong conclusions. If one has doubts one may check out the theory at a higher level. Such a notion seems indispensable – and not only for the present stage of development of artificial intelligence. There is no reason to believe that, unlike humans, machines will ever have unlimited reasoning power. And our ability to specify precisely the deductive power of the machine and increase it by graded amounts indefinitely is of considerable epistemological as well as practical interest. As Alan Robinson has remarked, it would be a real advance in our understanding if we could specify for human beings the precise limits of deductive power within which the truth of Fermat's conjecture has proved undecidable.

### Experimental method

The feasibility of this approach is being investigated on the following problem. Particular facts about one or more particular groups are supplied, such as, e.g.,

$$e \,.\, e = e$$
$$a \,.\, e = a$$
$$(b \,.\, b).\ c \neq c$$

and a theory is developed and progressively refined as more and more facts are provided. Perhaps, in this way, one may be led to the axioms of group theory, but it is wrong to think this is the only criterion of success of the method. Any consistent theory arrived at which is compatible with all the facts, implies some of them, is not too long and makes predictions which are verified, is a successful theory; and such a theory may be different from group theory.

A number of constraints were deliberately imposed on the problem for the initial experiments, for the sake of simplicity. Every fact was presented in the form of a clause, so as to facilitate the use of existing theorem-provers. To avoid the use of equality axioms or paramodulation, only a single 3-place predicate letter $P$ was employed, where $P(x, y, z)$ means $x \,.\, y = z$.

Any application of the proposed method has right at the start to face up to a knotty question: what facts should be generalized? It will not do to generalize all the facts provided, unless they are rather few, as this would mean that the number of axioms of the theory is about as large as the number of facts, which is undesirable. So facts need to be sorted into two categories: those used for abstraction, and those used only for checking purposes. The former will usually be a good deal less in number than the latter. In the present experiments, a rather simple but probably sensible criterion was employed for this division: positive facts only, i.e., equalities, were used for

abstraction, and the negative ones, i.e., inequalities, used for checking out the theory.

Three further constraints were imposed. The facts supplied each involved no more than two individual constants, and abstraction was carried out on constants rather than separate *occurrences* of constants: that is to say, for example, the fact $P(a, e, a)$ might be generalized to $(x)(y)P(x, y, x)$ but *not* to $(x)(y)(z)P(x, y, z)$ which results from abstracting separately on *two* occurrences of $a$. Thirdly, existential quantifiers in the generalizations were eliminated by means of Skolem functors: this was done to make possible the direct use of existing theorem-provers.

### Choice of generalizations

Within the constraints imposed, the number of possible generalizations of a fact is finite, but some rank order must be placed on them to determine the sequence in which they are tried by Induc. It was decided to give more general clauses higher priority than less general ones implied by them, for two reasons: firstly, it is desirable to have as general a theory as possible; secondly, the opposite policy of giving the less general priority would mean less flexibility, since a theory which proved inconsistent could not be 'patched' by substituting for one of the offending axioms another immediately below it in the rank order.

However, this principle of giving a more general clause higher rank than a less general one is not by itself sufficient to determine a total order, it yields only a partial order (in fact, a lattice). For instance if $F(a, b)$ is a factual clause containing only two constants though any number of occurrences of them, the possible generalizations are the following three clauses:

1. $F(x, y)$
2. $F(x, b)$
3. $F(a, y)$.

For completeness we add the fact generalized on:

0. $F(a, b)$.

The generalization lattice for this case is shown in figure 1.

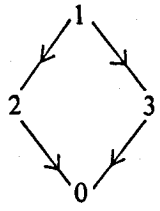



Figure 1. Generalization lattice for 2-constant abstraction used in experiments

An examination of this lattice reveals the formal similarity of clauses on the same level, and so immediately suggests, in accordance with the principle of higher priority of more generality, the choice order: 1, 2, 3.

The generalization lattice for 1-constant abstraction is the simple two-node one:

1. $F(x)$
0. $F(a)$.

Only these two lattices have been used in the experiments to date. The rank ordering of generalizations and the notion of 'fact' is discussed in the Appendix.

**Test of method**

The theorem-prover used is a complete binary resolution one based on Kowalski's optimal 'diagonal search' strategy (Kowalski 1970) and implemented by Isobel Smith in POP-2 for time-sharing on an ICL 4130 machine. It uses P1-deduction (Robinson 1965b), resolving on distinguished literals and factoring of positive clauses only (Kowalski and Hayes 1969). The program generates resolvents in successive strata, each characterized by the value of a parameter equal to the sum of the length and resolution depth of the clause; in the course of the program this parameter increases by steps of 1 to an upper bound (10 in the actual experiments). This upper bound is in effect the *level*, discussed above, of the theory generated by Induc.

The facts were extracted, in a more or less random fashion, from the composition tables of two very simple finite groups, the cyclic groups of order 2 and order 4, *viz.*:

| | $e$ | $a$ |
|---|---|---|
| $e$ | $e$ | $a$ |
| $a$ | $a$ | $e$ |

and

| | $e$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|
| $e$ | $e$ | $b$ | $c$ | $d$ |
| $b$ | $b$ | $d$ | $e$ | $c$ |
| $c$ | $c$ | $e$ | $d$ | $b$ |
| $d$ | $d$ | $c$ | $b$ | $e$ |

The facts used were:

$$e \,.\, e = e$$
$$a \,.\, e = a$$
$$(a \,.\, a) \,.\, e = a \,.\, (a \,.\, e)$$
$$(e \,.\, a) \,.\, a = e$$
$$e \,.\, a \neq e$$
$$a \,.\, a \neq a$$
$$b \,.\, c = c \,.\, b$$
$$b^3 = c$$
$$(b \,.\, b) \,.\, c \neq c$$
$$(b \,.\, c) \,.\, c \neq b.$$

Strictly speaking, in our formal system which does not contain equality, the expression of most of these facts would each require *two* clauses. For

example, the 7th one would require not only 7.0, given below, but also $\{\bar{P}(c, b, u)(P(b, c, u))\}$. The following list of clauses should therefore be looked upon as a *partial* presentation of these facts:

1.0 $P(e, e, e)$
2.0 $P(a, e, a)$
3.0 $\bar{P}(a, a, u)\,\bar{P}(u, e, w)\,\bar{P}(a, e, v)\,P(a, v, w)$
4.0 $\bar{P}(e, a, u)\,P(u, a, e)$
5.0 $\bar{P}(e, a, e)$
6.0 $\bar{P}(a, a, a)$
7.0 $\bar{P}(b, c, u)\,P(c, b, u)$
8.0 $\bar{P}(b, b, u)\,P(u, b, c)$
9.0 $\bar{P}(b, b, u)\,\bar{P}(u, c, c)$
10.0 $\bar{P}(b, c, u)\,\bar{P}(u, c, b)$.

The positive facts 1.0, 2.0, 3.0, 4.0, 7.0, 8.0 were chosen to abstract upon, the remaining four inequalities being used only for checking purposes. Each of the positive facts was provided with its rank order of generalizations algorithmically as described above. For example, the complete list for 1.0 was

1.1 $P(x, x, x)$
1.0 $P(e, e, e)$

and that for 8.0 was

8.1 $\bar{P}(x, x, u)\,P(u, x, y)$
8.2 $\bar{P}(b, b, u)\,P(u, b, y)$
8.3 $\bar{P}(x, x, u)\,P(u, x, c)$
8.0 $\bar{P}(b, b, u)\,P(u, b, c)$.

The Induc algorithm at the present stage of its evolution is essentially as follows:

(1) Fix the level $l$ of the theorem-prover (hereafter referred to as Deduc).
(2) Give as data to Deduc the highest order general clauses, as well as all the factual checking clauses.
(3) If no empty clause is generated, indicating consistency (at level $l$) of the input, go to (5).
If the empty clause is generated, indicating a contradiction, trace the proof and replace one of the input general clauses involved in the derivation by the next ranking one in its list. If it has no such successor replace it by its factual clause.
(4) Apply Deduc again and go to (3).
(5) Remove the factual clauses from the set of clauses. The remainder constitute an *adequate* theory of level $l$.
(6) (Optional test for redundancy.) In the standard way, for each clause in turn, use Deduc to test whether it is a logical implication of the other clauses and, if it is, discard it.

The theory of level 10 that resulted from this algorithm consisted of the following clauses:

2.3 $P(x, e, x)$
3.1 $\bar{P}(x, x, u)\ \bar{P}(u, y, w)\ \bar{P}(x, y, v)\ P(x, v, w)$
4.2 $\bar{P}(y, a, u)\ P(u, a, y)$
7.1 $\bar{P}(x, y, u)\ P(y, x, u)$
8.2 $\bar{P}(b, b, u)\ P(u, b, y)$

or, in the more familiar notation,

$x \cdot e = x$ ..... A1
$(x \cdot x) \cdot y = w$ implies $x \cdot (x \cdot y) = w$ ..... A2
$(y \cdot a) \cdot a = y$ ..... A3
$x \cdot y = y \cdot x$ ..... A4
$b^3 = y$ ..... A5.

It is noteworthy that Induc has delivered the right-identity axiom A1 in full generality, a special case of the associativity law in A2 and the general axiom of commutativity A4. The particular groups which provided the facts, being cyclic, *were* of course commutative; the remaining axioms A3 and A5 while not true of these groups are – to level 10 – consistent with the facts provided. Note also that Induc was given no information indicating that two distinct groups were being studied.

**General discussion**

The simplicity and success of Induc in obtaining such results on the basis of only ten facts may be surprising – but possibly only to those who take the view that the main business of induction is the accumulation and ordering of as many factual observations as possible. Chomsky (1968) refers to an interesting lecture by the great nineteenth-century American philosopher, Pierce, in which he maintained that the history of early science showed that approximations to correct theories were discovered with remarkable ease and rapidity, on the basis of highly inadequate data, as soon as certain problems were faced. He took the view that the instinctive structure of human intelligence imposed severe constraints upon the form of admissible hypotheses, thus reducing them to a manageable number, and only because of this could knowledge be acquired. Induc works in just this way, and its success – if one accepts Pierce's view – suggests that the formal structure of predicate logic, quite apart from its aptness for deduction, might in some sense model the mental operations we use in acquiring new knowledge.

The most important constraint imposed in Induc is the expression of all facts and axioms as (Skolemized) clauses, and the use of only abstraction and generalization on constants for creation of theories. The other constraints are inessential: for instance, the same generalization lattices could be used for abstracting on *occurrences* of constants; and 3-constant and higher degree abstractions can be made easily on the basis of their generalization lattices.

Emphatically the present system needs much work on its further development, particularly on such issues as criteria of choice of facts to abstract upon (but *see* Appendix), and economy in the operation of checking against facts when the latter become very large in number. Also, when as in the above example the facts come from more than one domain, the problem may arise of how the theory is to discriminate between the different domains: the provision of limited closure axioms or the use of sorts may be appropriate for this purpose. Other improvements one can envisage are the condensation of two or more axioms into one by generalization, a topic studied by Plotkin (1970), and the introduction of macro predicates as discussed in Part 1.

Also, of course, a system such as Induc need not be merely a digester of facts, but could easily be provided with the ability – at any stage of its theory construction – to deduce from its current axioms a new 'fact' as a *prediction*, which may then be checked against the outer world; if it proved wrong, an intimation in clausal form of this to Induc would set it off on another cycle of its algorithm to try to mend the theory. Such an ability is analogous to that of well-known programs like DENDRAL (Buchanan *et al.* 1969) and PROSE (Vigor *et al.* 1969).

Some may see in the notion above of *level* of an inductive theory something analogous to *degree of confirmation* in Carnap's (1963) well-known system of inductive logic. Both are measures of reliability of the theory, but the former refers only to the deductive power used in construction of the theory.

Induc seems to be the only application of theorem-provers to testing the consistency rather than the unsatisfiability of a set of formulae, reported in the literature. The strategies developed for unsatisfiability are not necessarily the best for consistency: for example, the strategy of set-of-support is unlikely to be very useful for the latter.

It is hoped that the approach here described, by its precise constraints on admissible hypotheses, may make induction amenable to analysis of the same rigorous kind as has proved so fruitful in the study of deduction. The extension of the method to higher-order logic should facilitate the generation of new concepts in theories.

**Acknowledgements**

I wish to thank P.Hayes, R.Kowalski, Isobel Smith and D.Kuehner for helpful discussions, particularly with regard to the clarification of the notion of 'fact' in the Appendix. I also gratefully acknowledge the support of the Science Research Council.

## REFERENCES

Andrews, P.B. (1969) Resolution with merging. *J. Assoc. comput Mach.*, **15**, 367–381.

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.

Carnap, R. (1963) *Logical foundations of probability*. Chicago: University of Chicago Press.

Chomsky, N. (1968) *Language and mind*. New York: Harcourt, Brace & World.

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.

Darlington, J.L. (1969) Theorem proving and information retrieval. *Machine Intelligence 4*, pp. 173–81 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Green, G. (1969) Application of theorem proving to problem solving. *Proceedings Int. Joint Conf. on Artificial Intelligence, Washington D.C.*, pp. 219–39 (eds Walker, D.E., & Norton, L.M.)

Hilbert, D. & Ackermann, W. (1950) *Principles of Mathematical Logic*. New York: Chelsea.

Kowalski, R. (1970) Search procedures in automatic theorem-proving. *Machine Intelligence 5*, pp. 181–201 (eds. Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

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.

Loveland, D. (to be published) A linear format for resolution. *I.R.I.A. symposium on automatic demonstration 1968*. Berlin: Springer-Verlag.

Meltzer, B. (1968) Some notes on resolution strategies. *Machine Intelligence 3*, pp. 71–5 (ed. Michie, D.) Edinburgh: Edinburgh University Press.

Pitrat, J. (1966) Réalisation de programmes de démonstration de théorèmes utilisant des méthodes heuristiques. *Thesis*. Paris: University of Paris.

Plotkin, G. (1970) Note on inductive generalization. *Machine Intelligence 5*, pp. 153–63 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Robinson, G. & Wos, L. (1969) Paramodulation and theorem-proving in first-order theories with equality. *Machine Intelligence 4*, pp. 135–50 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Robinson, J.A. (1965a) A machine-oriented logic based on the resolution principle. *J. Assoc. comput. Mach.*, **12**, pp. 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 in applied mathematics* XIX. Providence, Rhode Island: American Mathematical Society.

Robinson, J.A. (1969) Mechanizing higher-order logic. *Machine Intelligence 4*, pp. 151–70 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Slagle, J.R. (1967) Automatic theorem-proving with renamable and semantic resolution. *J. Assoc. comput. Mach.*, **14**, 687–97.

Vigor, D.B., Urquhart, D. & Wilkinson, A. (1969) PROSE – parsing recognizer outputting sentences in English. *Machine Intelligence 4*, pp. 271–84 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Waldinger, R.J. & Lee, R.C.T. (1969) PROW: a step toward automatic program writing. *Proceedings Int. Joint Conf. On Artificial Intelligence, Washington D.C.*, pp. 241–52 (eds Walker, D.E & Norton, L.N.).

Wos, L., Carson, D. & Robinson, G.A. (1964) The unit preference strategy in theorem proving. *A.F.I.P.S. Conference Proceedings 26*, pp. 615–21. Washington D.C.: Spartan Books.

Wos, L., Robinson, G.A. & Carson, D F. (1965) Efficiency and completeness of the set of support strategy in theorem proving. *J. Assoc. comput. Mach.*, **12**, 536–41.

## APPENDIX: GENERALIZATION LATTICES AND THE NOTION OF 'FACT'

The generalization lattices referred to in the text are really sub-lattices of what one might consider the 'full' lattice associated with a given fact. For example, taking again the case of a 2-constant fact $F(a, b)$, consider the following complete list of propositions which may be formed by abstraction and prefixing universal or existential quantifiers. (The clausal form of each is also given – $f$, $k$, $m$ being Skolem functors of degrees 1, 0, 0 respectively):

| | |
|---|---|
| 0. $F(a, b)$ | $F(a, b)$ |
| 1. $(x)(y)F(x, y)$ | $F(x, y)$ |
| 2. $(x)F(x, b)$ | $F(x, b)$ |
| 3. $(y)F(a, y)$ | $F(a, y)$ |
| 4. $(Ey)(x)F(x, y)$ | $F(x, k)$ |
| 5. $(Ex)(y)F(x, y)$ | $F(k, y)$ |
| 6. $(x)(Ey)F(x, y)$ | $F(x, f(x))$ |
| 7. $(y)(Ex)F(x, y)$ | $F(f(y), y)$ |
| 8. $(Ey)F(a, y)$ | $F(a, k)$ |
| 9. $(Ex)F(x, b)$ | $F(k, b)$ |
| 10. $(Ex)(Ey)F(x, y)$ | $F(k, m)$ |

If these propositions be partially ordered by the relation of logical implication the lattice of figure 2 results. The 'fact' labelled 0 is in the middle of this lattice, and the upper sub-lattice 1, 2, 3, 0 terminating on it is identical with the lattice of figure 1, which was used for theory construction in the experiments. These propositions are the only ones which actually *imply* 0, for 4, 5, 6, 7 have no such relation with it, while 8, 9, 10 are *implied by* it.

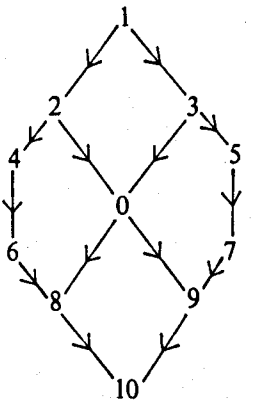



Figure 2. 'Full' generalization lattice for 2-constant abstraction referred to in the Appendix

The full lattice helps to clarify the notion of what we are to mean by a 'fact' in the present context. For the 'fact' 0 is not, as one might have expected, at the bottom of this lattice, which place is occupied by the mere ontological

commitment of proposition 10. Furthermore, it will be noticed in the examples given in the text that the expression of facts was allowed to contain universal quantifiers, which might run counter to an intuition that a 'fact' should contain only individual constants, not individual variables, in argument places of the predicates.

The view I take is that in the process of induction *any* proposition about the domain concerned may in principle be treated as a 'fact', on the basis of which one may try to abstract and generalize. Whether one actually chooses a given one for this purpose depends on whether one believes it correct and whether it does, or is likely to, prove useful for this purpose. It must be remembered that even 'facts' expressed with individual constants but no variables may have some theory implicit in them. For example, the eliciting of the fact $(a \,.\, a) \,.\, e = a \,.\, (a \,.\, e)$ from the composition table of the cyclic group of order 2, involves the consideration of *three* entries of the table, namely $a \,.\, a = e$, $e \,.\, e = e$, $a \,.\, e = a$, *as well as* general properties of equality such as transitivity.

Returning to the case considered above, we see that any one of the nodes might have been used as a fact to form the basis of a lattice. For example, if 10 were chosen, its lattice would include 1, 4, 5, 6, 7, 10; one could well imagine axiomatic set theory having been inductively built up partly from existential assertions of this kind. If 3 had been chosen, its lattice would have included only 1, 3. The choice of 0 in the experiments, we have seen, involved the lattice 1, 2, 3, 0.

À propos of the question raised in the text about criteria of choice of facts to generalize on, it is interesting to note that the proposed method of induction by its very nature provides one such: namely, if one has used some fact for this purpose and one finds in the course of development of the theory one has been forced down the lattice right down to its own node, one has thereby demonstrated that it is unsuitable and should belong only to the category of 'checking' facts. This in fact happened in the experiment described in the text – to the fact $P(e, e, e)$, there labelled 1.0.

# 10

# Search Strategies for Theorem-Proving

Robert Kowalski
Metamathematics Unit
University of Edinburgh

We will define the notions of abstract theorem-proving graph, abstract theorem-proving problem $\mathscr{P}$ and search strategy $\Sigma$ for $\mathscr{P}$. These concepts generalize the usual tree (or graph) searching problem and admit Hart, Nilsson and Raphael (1968) and Pohl (1969) theories of heuristic search. In particular the admissibility and optimality theorems of Hart, Nilsson and Raphael generalize for the classes $\mathscr{D}$ and $\mathscr{D}^u$ of diagonal search strategies for abstract theorem-proving problems. In addition the subclass $\mathscr{D}^u$ of $\mathscr{D}$ is shown to be optimal for $\mathscr{D}$. Implementation of diagonal search is treated in some detail for theorem-proving by resolution rules (Robinson 1965).

## SEARCH STRATEGIES, COMPLETENESS AND EFFICIENCY

Completeness and efficiency of proof procedures can be studied only in the context of search strategies. A system $T$ of inference rules and axioms can be complete or incomplete for a given class of intended interpretations. Similarly a search strategy $\Sigma$ for $T$ may or may not be complete for obtaining proofs constructible in $T$ – independently of the completeness of $T$. A proof procedure $(T, \Sigma)$ consists of a system of inference rules and axioms $T$ together with a search strategy $\Sigma$ for $T$. The procedure $(T, \Sigma)$ can be complete in several distinct senses depending upon the completeness of $T$ and $\Sigma$. These distinct notions of completeness are often confused and this confusion results in confused discussion regarding the value of complete versus heuristic methods in automatic theorem-proving.

The situation is no better with regard to discussions of efficiency. Proposals have been put forward both for increasing the strength of inference systems and for restricting the application of inference rules. Thus, for instance, $\omega$-order logic is a strong inference system, whereas set-of-support resolution (Wos, Carson and Robinson 1965) is a restricted inference rule. However it is only proof procedures $(T, \Sigma)$ which can be efficient for proving theorems.

A system of inference rules and axioms $T$ is potentially efficient only if it admits a search strategy $\Sigma$ which yields an efficient proof procedure $(T, \Sigma)$. On the other hand, a search strategy $\Sigma$ can be efficient for obtaining proofs constructible within $T$ regardless of the efficiency of the resulting proof procedure $(T, \Sigma)$, i.e., $\Sigma$ may do a best possible job for an impossible $T$.

It is interesting that certain inference-related rules can be defined only in the context of search strategies. Deletion of subsumed clauses is an important example. The completeness of a deletion strategy for a proof procedure $(T, \Sigma)$ is relative to the completeness of $(T, \Sigma)$ and might be better termed 'compatibility with $(T, \Sigma)$'. Our own proof for the compatibility of deleting subsumed clauses (Kowalski and Hayes 1969) fails because no regard is taken of this relativity to search strategies. The compatibility with given $(T, \Sigma)$ of deleting subsumed clauses has been proved for the case where $\Sigma$ is a level saturation search, $T$ is ordinary binary resolution (Robinson 1965) or AM-clash resolution (Sibert 1969) and deletion is done only after each level is saturated. Compatibility of the usual deletion rule for most complete $T$ and for arbitrary $\Sigma$ is proved in Kowalski (1970) where counterexamples are also exhibited for the compatibility and efficiency of alternative rules for deleting subsumed clauses. The compatibility with given $(T, \Sigma)$ of deleting tautologies is a much simpler matter – but first proved explicitly for arbitrary $\Sigma$ and most resolution systems $T$ in Kowalski (1970).

Despite the importance of search strategies, most research in automatic theorem-proving has concentrated on developing new inference systems which are either more powerful or more restricted than ones already existing. The Unit Preference strategy of Wos, Carson and Robinson (1964) seems to be the basic search strategy employed by most computer programs which implement resolution rules in proof procedures. Slagle's Fewest Components strategy (*see* Sandewall 1969), Green's (1969a) partitioning of clauses into active and passive clauses, and Burstall's (1968) indexing scheme seem to be the only other reported proposals for improving search strategies.

## THEOREM-PROVING GRAPHS

It is disconcerting that none of the research in tree searching techniques has yielded improved search strategies for theorem-proving. We wholly agree with Sandewall's (1967) assessment that searching for paths in trees is not general enough to represent the searches needed in automatic theorem-proving. A similar situation exists with respect to and/or trees where searching for subtrees cannot be represented helpfully by searching for paths in other trees. The notion of theorem-proving graph, defined below, is intended to extend the usual notion of tree (or graph) and to apply to the theorem-proving problem without encompassing the notion of and/or tree.

In the theorem-proving problem we begin with an initial non-empty set of sentences $S_0$ and with a set of inference rules $\Gamma$. If $\varphi \in \Gamma$ and $S$ is a set of sentences then $\varphi(S)$ is another set of sentences. $\varphi(S) = \varnothing$ if $\varphi$ is not applicable

to $S$. In particular $\varphi(S)=\varnothing$ if $S$ is not finite. In applications to resolution systems, $S_0$ is a set of clauses and $\Gamma$ consists of a single resolution rule or of a factoring rule and a separate rule for resolving factors. If $\varphi$ is binary resolution of factors then $\varphi(S)=S'\neq\varnothing$ if $S$ contains two factors which resolve or one factor which resolves with itself and each $C'\in S'$ is a resolvent of the clauses in $S$. If $\varphi$ is the operation of unifying literals in a single clause then $\varphi(S)=S'\neq\varnothing$ if $S$ is a singleton, $S=\{C\}$, and each $C'\in S'$ is a factor of $C$.

Given an initial set of sentences $S_0$ and a set of inference rules $\Gamma$ let $S^*$ be the set of all sentences which can be derived from $S_0$ by iterated application of the rules in $\Gamma$. Then each $\varphi\in\Gamma$ is a function $\varphi: 2^{S^*}\rightarrow 2^{S^*}$ defined on subsets of $S^*$ taking subsets of $S^*$ as values. Each sentence $C\in S^*$ can be assigned a level: if $C\in S_0$ then the level of $C$ is zero, otherwise $C\in\varphi(S)$ for some $\varphi\in\Gamma$ and for some $S\subseteq S^*$ and the level of $C$ is one greater than the maximum of the levels of the sentences $D\in S$. If $S_i$ is the set of all sentences of level $i$ then $S^*=\bigcup_{0\leqslant i} S_i$. Since a sentence $C\in S^*$ may have several distinct derivations, the level of $C$ need not be unique. Since $\varphi(S)\neq\varnothing$ only if $S$ is finite, the set of sentences which occur in a given derivation of a sentence $C\in S^*$ is always finite. The theorem-proving problem for a triple $(S_0, \Gamma, F)$, $F\subseteq S^*$, is that of generating by means of a search strategy $\Sigma$ some $C^*\in F$ by iterated application of the rules in $\Gamma$ beginning with the sentences in $S_0$. For certain applications it may be required to derive a sentence $C^*\in F$ having minimum level in $F$ or, more generally, having minimum cost in $F$, where cost is determined by some 'costing function' defined on the sentences in $S^*$. The tree (or graph) searching problem (Doran and Michie 1966, Sandewall 1969) can be interpreted as a theorem-proving problem $(S_0, \Gamma, F)$ where each operator $\varnothing\in\Gamma$ has the property that $\varphi(S)=\varnothing$ whenever $S$ is not a singleton.

A triple $(S_0, \Gamma, F)$ determines a directed graph whose nodes are single sentences $C\in S^*$. $C'$ is an immediate successor of $C$ (i.e., $C'$ is connected to $C$ by an arc directed from $C$ to $C'$) if for some $S\subseteq S^*$ and $\varphi\in\Gamma$, $C\in S$ and $C'\in\varphi(S)$. The situation is similar to that which exists for ordinary graph searching problems as distinguished from tree searching problems. Searching in a directed graph for a path from a node $a$ to a node $b$ can be interpreted as searching in a directed labelled tree for a path from a node $n_1$, with label $c(n_1)=a$, to a node $n_2$, with label $c(n_2)=b$. The tree search interpretation of graph searching has the property of representing a single node $c$ in a graph as distinct nodes $n_1, \ldots, n_k$ in a tree when the node $c$ can be generated in $k$ different ways as the end node of $k$ different paths from the initial node $a$. This property of the tree search representation is one which we find useful when extended to deal with the more general theorem-proving problem. In particular the extended tree search representation associates distinct nodes with distinct derivations. This $1-1$ correspondence between nodes and derivations allows the number of nodes generated by a search

strategy $\Sigma$ in the course of obtaining a terminal node to be treated as a measure of the efficiency of $\Sigma$ for the given problem.

We define the notion of an *abstract theorem-proving graph* ('abstract graph' or simply 'graph') $(G, s)$. The extended tree representation of an *interpreted theorem-proving graph* $(S_0, \Gamma)$ can be obtained from $(G, s)$ by labelling the nodes $n \in G$ by use of a labelling function $c: G \to S^*$, and by interpreting each application of the function $s$ to a subset $G' \subseteq G$ as an application of a function $\varphi \in \Gamma$ to the subset $\{c(n) | n \in G'\}$. An abstract theorem-proving graph is a pair $(G, s)$ where $G$ is a set of nodes, $s: 2^G \to 2^G$ is a successor function defined on subsets of $G$ taking subsets of $G$ as values. $G$ and $s$ satisfy the following conditions:

(1) $s(\emptyset) = \emptyset$.
(2) $s(G') \neq \emptyset$ implies that $G'$ is finite.
(3) $G' \neq G''$ implies that $s(G') \cap s(G'') = \emptyset$.
(4) Let $\mathscr{S}_0 = \{n \in G | n \notin s(G') \text{ for any } G' \subseteq G\}$,
let $\mathscr{S}_{k+1} = \{n \in G | n \in s(G') \text{ for some } G' \subseteq \bigcup_{i \leqslant k} \mathscr{S}_i,\ G' \cap \mathscr{S}_k \neq \emptyset\}$.

Then

(a) $\mathscr{S}_0 \neq \emptyset$,
(b) $G = \bigcup_{0 \leqslant i} \mathscr{S}_i$,
(c) $\mathscr{S}_i \cap \mathscr{S}_j = \emptyset$ for $i \neq j$.

The graph $(G, s)$ reduces to an ordinary tree if $s(G') \neq \emptyset$ implies that $G'$ is a singleton. For this case condition (3) states that distinct nodes have distinct sets of successors. More generally, (3) states that distinct sets of nodes have distinct sets of successors. It is precisely this condition which ensures that the graphs $(G, s)$ extend the ordinary tree representation of search spaces. Condition (4) states that $(G, s)$ is a levelled acyclic directed graph. In other words each $n \in G$ can be assigned a unique *level* $i$ where $n \in \mathscr{S}_i$ and $n \notin \mathscr{S}_j$ for all $j \neq i$. If $(S_0, \Gamma)$ is an interpretation of $(G, s)$ with labelling function $c$: $G \to S^*$ then $S_i = \{c(n) | n \in \mathscr{S}_i\}$ is just the set of labelled nodes of level $i$. Condition (3) guarantees that for each $C \in S^*$ and for each distinct derivation of $C$ from $S_0$ there is a distinct node $n \in G$ such that $C = c(n)$. There is no restriction that $\mathscr{S}_0$ or $S_0$ be finite. The case where $\mathscr{S}_0$ is infinite allows us to deal with axiom schemes in theorem-proving and more generally with potentially infinite sets of initial nodes $\mathscr{S}_0$.

The successor function $s$ of $(G, s)$ determines a partial ordering of the nodes in $G$: $n'$ is an *immediate successor* of $n$ (and $n$ an *immediate ancestor* of $n'$) if $n' \in s(G')$ and $n \in G'$ for some $G' \subseteq G$. A node $n'$ is a *successor* of $n$ (and $n$ an ancestor of $n'$), written $n' > n$, if $n'$ is an immediate successor of $n$ or if $n'$ is a successor of an immediate successor of $n$. We write $n \leqslant n'$ if $n < n'$ or $n = n'$. The definition of $(G, s)$ guarantees that for all $n \in G$ the set $\{n' | n' \leqslant n\}$ is finite, although the set $\{n' | n' \geqslant n\}$ may be infinite. Notice that in the theorem-proving interpretation of graphs $(G, s)$, a derivation of a sentence

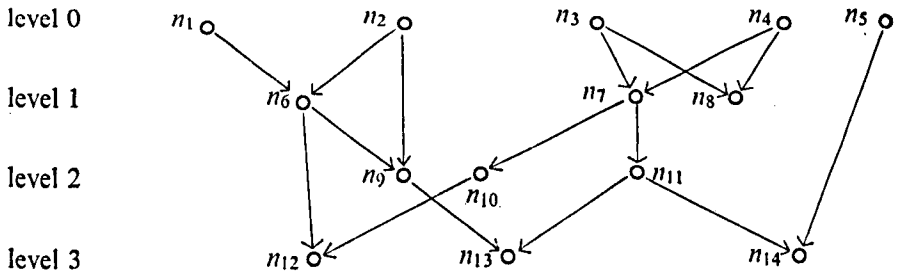



Figure 1

$c(n)$ consists of all the sentences $c(n')$ where $n' \leqslant n$. Each such derivation contains only finitely many sentences $c(n')$.

Figure 1 illustrates a graph $(G, s)$ where nodes are represented as points and where points $n$ and $n'$ are connected by a directed line from $n$ to $n'$ if $n'$ is an immediate successor of $n$. In general it is convenient to picture graphs as directed downward, so that $n$ lies above $n'$ if $n'$ is a successor of $n$. To determine in figure 1 if $n \in s(G')$ it suffices to verify that $G'$ is the set of all nodes connected to $n$ by an arc directed to $n$. Thus, for example,

$$s(n_1, n_2) = \{n_6\},$$
$$s(n_2, n_6) = \{n_9\},$$
$$s(n_3, n_4) = \{n_7, n_8\},$$
$$s(n_7) = \{n_{10}, n_{11}\},$$
$$s(n_2) = s(n_5) = s(n_8) = s(n_1, n_2, n_6) = \emptyset.$$

If the graph of figure 1 is interpreted as a resolution graph by a labelling function $c$: $G \to S^*$ then the two clauses $c(n_7)$ and $c(n_8)$ must be all the resolvents of the pair $c(n_3)$, $c(n_4)$. The clause $c(n_8)$ resolves with none of the clauses $c(n_i)$, $1 \leqslant i \leqslant 14$. The clauses $c(n_{10})$ and $c(n_{11})$ are either factors of $c(n_7)$ or are obtained from $c(n_7)$ by resolving $c(n_7)$ with itself. If $C = c(n_6) = c(n_7) = c(n_{14})$ then $C$ has three derivations, two of level one and one of level three. Derivations are not necessarily represented by derivation trees. For instance the derivation of $c(n_{13})$ consists of the clauses $c(n_1)$, $c(n_2)$, $c(n_3)$, $c(n_4)$, $c(n_6)$, $c(n_7)$, $c(n_9)$, $c(n_{11})$, $c(n_{13})$. The clause $c(n_2)$ is used twice in the derivation of $c(n_{13})$ but is represented by only one node in $G$.

An *abstract theorem-proving problem with non-negative costs* ('abstract problem with costs' or simply 'problem') is a tuple $\mathscr{P} = (G, s\ \mathscr{F}, g)$ where $\mathscr{F} \subseteq G$, the set of *terminal nodes* for $\mathscr{P}$ (or solution nodes), and $g$: $G \to \mathbb{R}$, the *costing function* of $\mathscr{P}$, ($\mathbb{R}$, the set of real numbers) are such that

(1) $n \in \mathscr{F}$ implies that $s(G') = \emptyset$ whenever $n \in G' \subseteq G$,

(2) (a) $g(n) \geqslant 0$ for all $n \in G$,

(b) if $n \in s(n_1, \ldots, n_k)$ (we write $s(n_1, \ldots, n_k)$ instead of $s(\{n_1, \ldots n_k\})$) then $g(n) \geqslant \max_{1 \leqslant i \leqslant k} g(n_i)$.

A solution to the problem $\mathscr{P}$ is obtained by constructing an algorithm $\Sigma$ which generates from $\mathscr{S}_0$ a node $n \in \mathscr{F}$. Each node $n \in \mathscr{F}$ is assigned a cost and it is often required to solve $\mathscr{P}$ by generating a node $n \in \mathscr{F}$ having minimal cost in $\mathscr{F}$. If $g(n) = 0$ for all $n \in G$ then in effect we have a problem without costs. Alternatively $g(n)$ may be taken to be the level of $n$, the number of nodes $n' \leqslant n$ or any other value which satisfies (3) above. In applications to resolution theory $g(n)$ is usually taken to be the level of the clause $c(n)$. For $n \in \mathscr{S}_0$ we do not require that $g(n) = 0$. This freedom allows us to assign different costs to distinct nodes in $\mathscr{S}_0$ and is especially useful when $\mathscr{S}_0$ is infinite. The set $\mathscr{F}$ may be empty in which case the problem has no solution. In resolution applications when $\mathscr{F} = \{n \in G | c(n) = \square\}$ then $\mathscr{F}$ is empty if the set $S_0 = \{c(n) | n \in \mathscr{S}_0\}$ is satisfiable. The general problem $\mathscr{P} = (G, s, \mathscr{F}, g)$ reduces to the ordinary tree searching problem when $(G, s)$ is a tree.

## SEARCH STRATEGIES FOR ABSTRACT THEOREM-PROVING PROBLEMS

A *search strategy* $\Sigma$ for $\mathscr{P}$ is a function $\Sigma: 2^G \to 2^G$ which generates subsets of $G$ from other subsets of $G$. Given such a function $\Sigma$ for $\mathscr{P}$ we define the sets $\Sigma_i$ of nodes already *generated by* $\Sigma$ *before the* $(i+1)$th *stage* and $\tilde{\Sigma}_i$ of nodes which are *candidates for generation by* $\Sigma$ *at the* $(i+1)$th *stage*:

$$\Sigma_0 = \varnothing, \ \tilde{\Sigma}_0 = \mathscr{S}_0, \tag{1}$$

$$\Sigma_{i+1} = \Sigma_i \cup \Sigma(\Sigma_i),$$

$$\tilde{\Sigma}_{i+1} = (\{n | n \in s(G'), \ G' \subseteq \Sigma_{i+1}\} \cup \tilde{\Sigma}_i) - \Sigma_{i+1}. \tag{2}$$

We require that $\Sigma$ satisfy

$$\Sigma(\Sigma_i) \subseteq \tilde{\Sigma}_i. \tag{3}$$

The set of nodes $\Sigma(\Sigma_i)$, chosen from the set of candidates $\tilde{\Sigma}_i$, is the set of nodes *newly generated by* $\Sigma$ *at the* $(i+1)$th *stage*. (It is easy to verify that $\Sigma_i \cap \tilde{\Sigma}_i = \varnothing$ for all $i \geqslant 0$.) The function $\Sigma$ should be interpreted as selecting subsets $G'$ of $\Sigma_i$ and generating nodes $n \in s(G')$ which have not been previously generated. The definitions above only partially formalize the intuitive notion of search strategy for $P$. In particular the search strategies $\Sigma$ are never allowed to display any redundancy, i.e., generate the same node twice. This restriction is not essential because given any concrete, possibly redundant, algorithm for generating nodes in $G$ there corresponds a unique search strategy $\Sigma$ which, except for redundancies, generates the same nodes in the same order.

Notice that $\Sigma(\Sigma_i)$ may contain more than one node – as is common with resolution strategies which simultaneously generate several resolvents of a single clash or several factors of a single clause. Notice too that nodes in $\mathscr{S}_0$ can be generated at any stage. We do not require that $\Sigma(\Sigma_i)$ contain a node $n \in \mathscr{F}$ when $\tilde{\Sigma}_i \cap \mathscr{F} \neq \varnothing$. If $\mathscr{P}$ is an ordinary tree search problem then the definition of search strategy for $\mathscr{P}$ provides a formal notion which applies to the usual strategies employed in searching for nodes in trees.

A search strategy $\Sigma$ for $\mathscr{P}=(G, s, \mathscr{F}, g)$ is *complete* for $\mathscr{P}$ if for all $n \in G$ there exists an $i>0$ such that $n \in \Sigma_i$. It is possible to define completeness in this way since we do not insist that $\Sigma$ generates no additional nodes after generating a first node $n^* \in \mathscr{F}$. We say that $\Sigma$ *terminates* at stage $i>0$ if $\mathscr{F} \cap \Sigma_{i-1}=\varnothing$ and either (1) $\mathscr{F} \cap \Sigma_i \neq \varnothing$ or (2) $\Sigma_i=\Sigma_{i-1}$. In the first case $\Sigma$ terminates with a solution and without a solution in the second case.

In the terminology of Hart, Nilsson and Raphael (1968), a search strategy $\Sigma$ is said to be *admissible* for $\mathscr{P}$ if $\Sigma$ is complete for $\mathscr{P}$ and terminates with a solution having least cost in $\mathscr{F}$ if $\mathscr{F} \neq \varnothing$, i.e., $n^* \in \mathscr{F} \cap \Sigma_i$, $\mathscr{F} \cap \Sigma_{i-1}=\varnothing$ implies that $g(n^*) \leqslant g(n)$ for all $n \in \mathscr{F}$. In resolution applications admissible search strategies are of special interest for robot control and automatic program writing (Green 1969b) where minimal cost solutions are related to simplest strategies and most efficient programs. More generally intuition suggests that, in the absence of special information about the location of non-minimal solutions, admissible search strategies will tend to be more efficient than non-admissible strategies for finding arbitrary solutions. An important step towards formalizing this intuition has already been made in the optimality theorems of Hart, Nilsson and Raphael (1968).

We define the notion of a search strategy $\Sigma$ for a problem $\mathscr{P}=(G, s, \mathscr{F}, g)$ being compatible with a *merit ordering* $\preccurlyeq$ defined on the nodes of $G$. For the moment we require only that $\preccurlyeq$ be reflexive and defined for all pairs of nodes in $G$. We write $n_1 \prec n_2$ ($n_1$ has *better merit* than $n_2$) when $n_1 \preccurlyeq n_2$ and not $n_2 \preccurlyeq n_1$. We write $n_1 \approx n_2$ ($n_1$ and $n_2$ have *equal merit*) if $n_1 \preccurlyeq n_2$ and $n_2 \preccurlyeq n_1$. A search strategy $\Sigma$ for $\mathscr{P}$ is compatible with a merit ordering $\preccurlyeq$ if for all $i \leqslant 0$,

(1) $\tilde{\Sigma}_i \neq \varnothing$ implies that $\Sigma(\Sigma_i) \neq \phi$,

(2) $n \in \Sigma(\Sigma_i)$ implies that $n \preccurlyeq n'$ for all $n' \in \tilde{\Sigma}_i$.

In other words $\Sigma$ always generates, from a non-empty set $\tilde{\Sigma}_i$, at least one node $n \in \tilde{\Sigma}_i$, and no node $n' \in \tilde{\Sigma}_i$ which is not generated by $\Sigma$ has better merit than any node $n \in \tilde{\Sigma}_i$ which is generated by $\Sigma$. Since a node $n$ may have better merit than an ancestor $n' < n$, $\Sigma$ need not generate nodes in order of merit. Distinct strategies $\Sigma$ and $\Sigma'$ for the same $\mathscr{P}$ compatible with the same merit ordering $\preccurlyeq$ differ only with regard to tie breaking rules for choosing which nodes to generate from a set of candidates having equal merit. If $\preccurlyeq$ is the trivial ordering, $n \preccurlyeq n'$ for all $n, n' \in G$, then $\preccurlyeq$ is a merit ordering for $G$ and any search strategy $\Sigma$ for $\mathscr{P}$ is compatible with $\preccurlyeq$. If $\preccurlyeq$ is the ordering by levels, $n \preccurlyeq n'$ if and only if $n \in \mathscr{S}_i$, $n' \in \mathscr{S}_{i'}$ and $i \leqslant i'$, then any search strategy for $\mathscr{P}$ compatible with $\preccurlyeq$ is a level saturation (or breadth first) strategy for $\mathscr{P}$. If $\preccurlyeq$ is the ordering by costs, $n \preccurlyeq n'$ if and only if $g(n) \leqslant g(n')$, then $\Sigma$ compatible with $\preccurlyeq$ is a cost saturation strategy for $\mathscr{P}$. If $\preccurlyeq$ is the inverse ordering by levels, $n \preccurlyeq n'$ if and only if $n \in \mathscr{S}_i$, $n' \in \mathscr{S}_{i'}$ and $i \geqslant i'$, then $\Sigma$ compatible with $\preccurlyeq$ is a depth first strategy for $\mathscr{P}$.

Lemma 1 states the fundamental properties of search strategies $\Sigma$ compatible with merit orderings: (a) any node $n_2 \in G$ is generated by $\Sigma$ before

any node $n_1$ which has worse merit than $n_2$ and than all the ancestors of $n_2$, (b) if $n_1$ is generated before $n_2$ then $n_2$ or some ancestor of $n_2$ has worse or equal merit to $n_1$.

*Lemma* 1

Let $\mathscr{P}=(G, s, \mathscr{F}, g)$ be a problem, $\preccurlyeq$ a merit ordering for $G$ and $\Sigma$ a search strategy for $\mathscr{P}$ compatible with $\preccurlyeq$.

(a) If $n_1 \in \Sigma_i$ and $n_2 \in G$ are such that $n \prec n_1$ for all $n \leqslant n_2$ then $n_2 \in \Sigma_{1-i}$.

(b) If $n_1 \in \Sigma_i$ and $n_2 \in \Sigma(\Sigma_i)$ then $n_1 \preccurlyeq n$ for some $n \leqslant n_2$.

*Proof.* (a) Let $n_1$ be generated at the $(j+1)$th stage, i.e., $n_1 \in \Sigma(\Sigma_j)$, $n_1 \notin \Sigma_j$ and $j<i$. If $n_2 \notin \Sigma_j$ then for some $n \leqslant n_2$, $n \notin \Sigma_j$ and $n \in \tilde{\Sigma}_j$. But $n \prec n_1$ and therefore $\Sigma$ is not compatible with $\preccurlyeq$ since it generates $n_1$ instead of $n$ at the $(j+1)$th stage. Therefore $n_2 \in \Sigma_j$ and $n_2 \in \Sigma_{i-1}$ since $j<i$.

(b) Suppose $n \prec n_1$ for all $n \leqslant n_2$. Then by (a), $n_2 \in \Sigma_{i-1}$ and therefore $n_2 \notin \Sigma(\Sigma_i)$.

A merit ordering $\preccurlyeq$ for $G$ is *$\delta$-finite* if for all $n \in G$ the set $\{n' \in G \mid n' \preccurlyeq n\}$ is finite (compare Hart, Nilsson and Raphael 1968). The importance of $\delta$-finite merit orderings is a consequence of Theorem 1: any search strategy compatible with a $\delta$-finite merit ordering is complete. Any merit ordering for a finite set $G$ is $\delta$-finite. Ordering by levels is $\delta$-finite if $\mathscr{S}_0$ is finite and $s(G')$ is finite for all $G' \subseteq G$, under the same conditions inverse ordering by levels is not $\delta$-finite if $G$ is infinite (by König's Lemma).

*Theorem* 1

If $\mathscr{P}=(G, s, \mathscr{F}, g)$ is a problem, $\preccurlyeq$ a $\delta$-finite merit ordering for $G$ and $\Sigma$ a search strategy for $\mathscr{P}$ compatible with $\preccurlyeq$, then $\Sigma$ is complete for $\mathscr{P}$.

*Proof.* Let $n^* \in G$ be given. We need to show that $n^* \in \Sigma_j$ for some $j>0$. If $G$ is finite then $G=\bigcup_{i>0} \Sigma_i$ since $\tilde{\Sigma}_i \neq \emptyset$ implies that $\Sigma(\Sigma_i) \neq \emptyset$ and since $\Sigma(\Sigma_i) \cap \Sigma_i = \emptyset$.

Otherwise if $G$ is infinite let $n' \leqslant n^*$ be a node such that $n \preccurlyeq n'$ for all $n \leqslant n^*$. Since $\preccurlyeq$ is $\delta$-finite, since $\tilde{\Sigma}_i \neq \emptyset$ implies that $\Sigma(\Sigma_i) \neq \emptyset$ and since $\Sigma(\Sigma_i) \subseteq \tilde{\Sigma}_i$ it follows that for some $j>0$ and for some $n_1 \in \Sigma_j$, $n' \prec n_1$, and therefore $n \prec n_1$ for all $n \leqslant n^*$ and by Lemma 1 (a), $n^* \in \Sigma_j$.

## HEURISTIC FUNCTIONS AND DIAGONAL SEARCH

There is special interest in merit orderings which can be expressed in terms of the cost function $g$ of $\mathscr{P}=(G, s, \mathscr{F}, g)$ and of an additional heuristic function $h$ (Hart, Nilsson and Raphael 1968, Nilsson 1968, Pohl 1969). A *heuristic function h* for $\mathscr{P}$ is a function $h: G \to \mathbb{R}$ such that $h(n) \geqslant 0$, for all $n \in G$. Let $f(n)=g(n)+h(n)$ for all $n \in G$. The intended interpretation of the heuristic function $h$ is that $f(n)=g(n)+h(n)$ is an estimate of the cost $g(n^*)$ of a terminal node $n^* \in \mathscr{F}$, such that $n \leqslant n^*$, i.e., $h(n)$ is an estimate of

$g(n^*)-g(n)$. If it is desired that $\Sigma$ be admissible then $h(n)$ is intended to estimate the minimum value of $g(n^*)-g(n)$ for $n^* \in \mathscr{F}$ such that $n \leqslant n^*$.

Suppose, for example, that we know of a given problem $\mathscr{P}_0 = (G_0, s_0, \mathscr{F}_0, g_0)$ that if it has a solution then its minimum cost is $k$. Suppose for simplicity that no $n \in G_0$ has cost $g_0(n)$ greater than $k$. Given only this information then an appropriate definition of a heuristic function $h_0$ for $\mathscr{P}_0$ is $h_0(n) = k - g_0(n)$ for all $n \in G_0$.

Suppose that a given problem $\mathscr{P}_1 = (G_1, s_1, \mathscr{F}_1, g_1)$ is interpreted as a resolution problem by a labelling function $c: G_1 \to S^*$. Suppose that the inference rules $\Gamma$ consist of a factoring operation for unifying two literals in a clause and of a separate resolution rule for resolving at most two factors. Let $g_1(n)$ be the level of $n$ and $\mathscr{F}_1 = \{n \mid c(n) = \square\}$. For $n \in G_1$ let $l(c(n))$ be the length of $c(n)$ (number of literals in $c(n)$). The heuristic function $h_1$ for $\mathscr{P}_1$ is defined by letting $h_1(n)$ be the expected length of $c(n)$:

(1) for $n \in \mathscr{S}_0$, $h_1(n) = l(c(n))$,
(2) for $c(n)$ a resolvent of $c(n_1)$ and $c(n_2)$, $h_1(n) = l(c(n_1)) + l(c(n_2)) - 2$,
(3) for $c(n)$ a factor of $c(n')$ (the result of unifying two literals in $c(n')$) $h(n) = l(c(n')) - 1$.

To the extent that merging does not occur (i.e., so long as $h_1(n) = l(c(n))$), $h_1(n)$ is a lower bound on the value of $g_1(n^*) - g_1(n)$ for $c(n^*) = \square$ when $c(n)$ occurs in a derivation of $\square$.

The costing function $g$ and heuristic function $h$ allow us to define two important classes of search strategies for $\mathscr{P}$. Given $\mathscr{P} = (G, s, \mathscr{F}, g)$ and $h$ a heuristic function for $\mathscr{P}$. Let the merit orderings $\preccurlyeq_d$ and $\preccurlyeq_{du}$ for $G$ be defined for all $n_1, n_2 \in G$, by

(1) $n_1 \preccurlyeq_d n_2$ if and only if $f(n_1) \leqslant f(n_2)$,
(2) $n_1 \preccurlyeq_{du} n_2$ if and only if $f(n_1) \leqslant f(n_2)$ and $h(n_1) \leqslant h(n_2)$ when $f(n_1) = f(n_2)$.

A search strategy $\Sigma$ for $\mathscr{P}$ is a *diagonal search strategy* for $\mathscr{P}$ and $h$ (written $\Sigma \in \mathscr{D}(\mathscr{P}, h)$) if and only if $\Sigma$ is compatible with the merit ordering $\preccurlyeq_d$. $\Sigma$ is an *upwards diagonal search strategy* for $\mathscr{P}$ and $h$ ($\Sigma \in \mathscr{D}^u(\mathscr{P}, h)$) if and only if $\Sigma$ is compatible with the merit ordering $\preccurlyeq_{du}$. Notice that $\mathscr{D}^u(\mathscr{P}, h) \subseteq \mathscr{D}(\mathscr{P}, h)$ and that $\mathscr{D}^u(\mathscr{P}, h) = \mathscr{D}(\mathscr{P}, h)$ if $h(n) = 0$ for all $n \in G$.

Except for minor differences, the search strategies $\Sigma \in \mathscr{D}(\mathscr{P}, h)$ coincide with those investigated in Hart, Nilsson and Raphael (1968) for the case of ordinary tree search. The search strategies $\Sigma \in \mathscr{D}^u(\mathscr{P}, h)$ differ from those in $\mathscr{D}(\mathscr{P}, h)$ by generating, from among candidate nodes which have equal merit for $\preccurlyeq_d$, those nodes whose cost is estimated to be closest to the cost of a solution node. In the case of the problem $\mathscr{P}_0$ and heuristic function $h_0$, defined above, $f_0(n) = g_0(n) + h_0(n) = k$ for all $n \in G_0$. All nodes in $G$ have equal merit for search strategies $\Sigma \in \mathscr{D}(\mathscr{P}_0, h_0)$. For $\Sigma \in \mathscr{D}^u(\mathscr{P}_0, h_0)$ nodes

which have cost closer to $k$ have better merit than nodes which have smaller cost. In case $g_0(n)$ is the level of $n$ for all $n \in G_0$ then $\Sigma \in \mathscr{D}^u(\mathscr{P}_0, h_0)$ is a depth-first search strategy, which intuitively seems the most efficient search strategy for $\mathscr{P}_0$, given only the information that a minimal solution of $\mathscr{P}_0$ must have level $k$. Concrete search algorithms for $\Sigma \in \mathscr{D}^u(\mathscr{P}_1, h_1)$ are discussed in the next section.

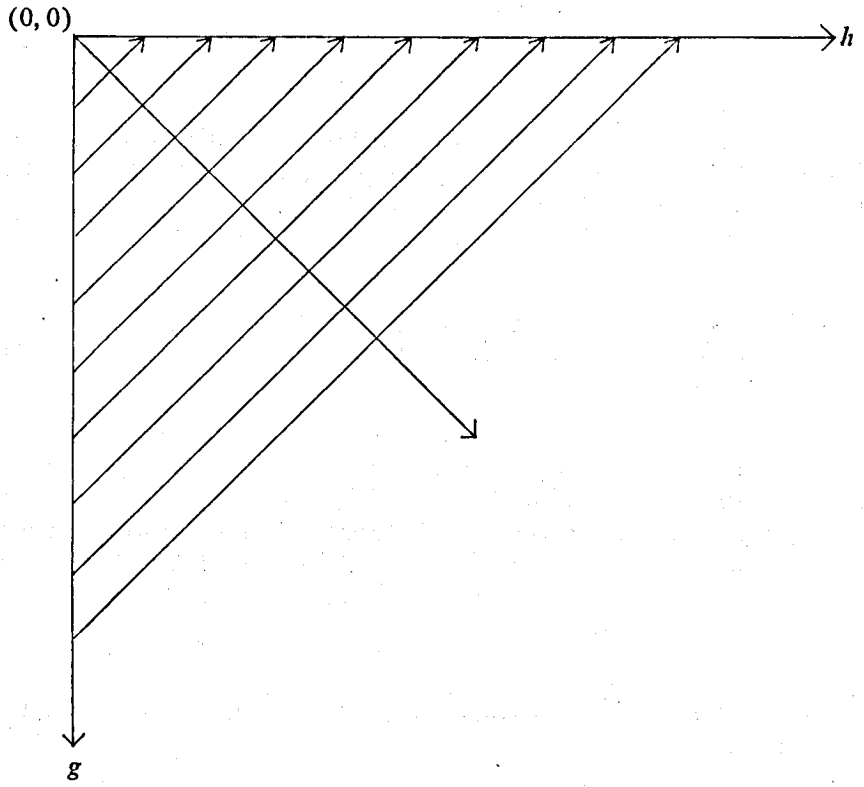



Figure 2

The terminology, diagonal and upwards diagonal search, is suggested by representing nodes $n \in G$ as occupying positions in the plane with coordinates $(h(n), g(n))$, where $h$ increases rightwards away from the origin and $g$ increases downwards away from the origin (see figure 2). $\Sigma \in \mathscr{D}(\mathscr{P}, h)$ attempts to generate nodes on consecutive diagonals in order of increasing distance from the origin (0, 0). In addition if $\Sigma \in \mathscr{D}^u(\mathscr{P}, h)$ then $\Sigma$ attempts to generate nodes, lying on a given diagonal $d$, upwards in order of increasing $h$. If $\preccurlyeq_d$ or $\preccurlyeq_{du}$ are $\delta$-finite then each diagonal contains only finitely many nodes $n \in G$ and for every diagonal $d$ there are only finitely many diagonals which contain nodes $n \in G$ and which are closer than $d$ to (0, 0).

Figure 3 illustrates Lemma 1 and Theorem 1 for a problem $\mathscr{P}$ and for a search strategy $\Sigma \in \mathscr{D}^u(\mathscr{P}, h)$ where $\preccurlyeq_{du}$ is assumed to be $\delta$-finite. The node $n^* \in \mathscr{T}$ has minimum cost in $\mathscr{T}$ and $n' \leqslant n^*$ is a node having worst merit in the set consisting of $n^*$ and all ancestors of $n^*$. The node $n \in G$ has better merit than $n^*$ and $n'' \leqslant n$ has worst merit in the set consisting of $n$ and all ancestors of $n$. Dots represent nodes, lying on diagonals, generated by $\Sigma$ before the generation of $n^*$. The small circles represent nodes generated by $\Sigma$ after the generation of $n^*$. The diagonal $d$ contains the node $n'$. By Lemma 1, $\Sigma$ generates $n^*$ before generating nodes having worse merit than $n'$, i.e., before generating nodes lying above $n'$ on $d$ and before generating nodes lying on diagonals to the right of $d$.

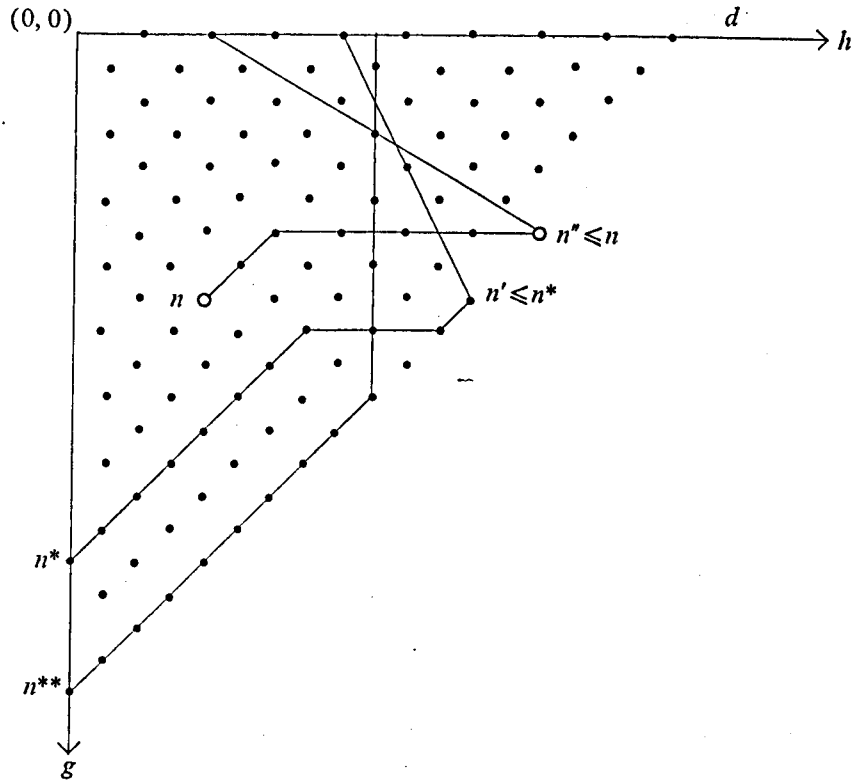



Figure 3

The heuristic function $h$ satisfies no conditions other than $h(n^*) = h(n^{**}) = 0$ and those imposed by the $\delta$-finiteness of $\preccurlyeq_{du}$. $\Sigma$ may fail to be admissible because some $n^{**} \in \mathscr{T}$ having worse merit than $n^*$ will be generated before $n^*$ if $n^{**}$ and all ancestors of $n^{**}$ have better merit than $n'$. The node $n \in G$ will not be generated before $n^*$ if $n''$ lies to the right of $d$ or above $n'$ on $d$.

## UPWARDS DIAGONAL SEARCH STRATEGIES FOR RESOLUTION

The algorithm $\Sigma^*$ defined below approximates an upwards diagonal search strategy for the resolution problem $\mathscr{P}_1$ and heuristic function $h_1$. The same algorithm $\Sigma^*$ when applied to the resolution problem $\mathscr{P}_2$ and heuristic function $h_2$ defined below is a pure upwards diagonal search strategy for $\mathscr{P}_2$ and $h_2$. The admissibility and optimality theorems of the next section apply to $\Sigma^*$ for $\mathscr{P}_2$ and $h_2$ and to $\Sigma^*$ for $\mathscr{P}_1$ and $h_1$, except when merging occurs in $\mathscr{P}_1$. A search strategy which differs inessentially from $\Sigma^*$ has been implemented in POP-2 by Miss Isobel Smith for a problem and heuristic function similar to $\mathscr{P}_1$ and $h_1$.

The definition and identification of the problem $\mathscr{P}_2$ was motivated by a suggestion of Mr Donald Kuehner. $\mathscr{P}_2=(G_2, s_2, \mathscr{F}_2, g_2)$ differs from $\mathscr{P}_1$ by interpreting clauses $c(n)$ as lists of literals and by explicitly exhibiting and assigning cost to the operation (treated as a special case of factoring) of identifying two copies of the same literal within a clause. The length $l(c(n))$ of $c(n)$ is defined as the number of literals in the clause $c(n)$, counting duplications. $g_2(n)$ and $h_2(n)$ are still defined respectively as the level of $n$ and expected length of $c(n)$. $h_2(n)=l(c(n))$ for all $n\in G_2$ and $h_2(n)$ is always a lower bound on the value of $g_2(n^*)-g_2(n)$ when $n\leqslant n^*$ and $n^*\in\mathscr{F}_2=\{n\in G_2 \mid c(n)=\square\}$.

Throughout the remainder of this section, $\mathscr{P}=(G, s, \mathscr{F}, g)$ and $h$ are either $\mathscr{P}_1$ and $h_1$ or $\mathscr{P}_2$ and $h_2$. The definition of $\Sigma^*$ for $\mathscr{P}$ and $h$ is the same for both of these cases except for the details remarked upon at the end of this section.

Clauses $c(n)$ are stored upon the generation of $n$ in cells $A(i,j)$ of a two-dimensional array $A$. $c(n)$ is stored in $A(i,j)$ if $l(c(n))=i$ and $g(n)=j$. Although it is natural to represent cells $A(i,j)$ as lists of clauses, we write $c(n)\in A(i,j)$ if $c(n)$ is stored in $A(i,j)$ when $n$ is generated. The *merit* of a node $n\in G$ is defined to be the cell $A(h(n), g(n))$. The cell $A(i,j)$ is said to be *better* than $A(i',j')$ (written $A(i,j)\prec A(i',j')$) if

(1) $i+j<i'+j'$ or
(2) $i+j=i'+j'$ and $i<i'$.

Thus *a node $n\in G$ has better upwards diagonal merit than a node $n'\in G$ if and only if the merit of $n$ is better than the merit of $n'$*, equivalently $n\leqslant_{d^u} n'$ if and only if $A(h(n), g(n))\prec A(h(n'), g(n'))$. Notice that for $\mathscr{P}_2$ and $h_2$, $n\in G_2$ has merit $A(i,j)$ if and only if $c(n)\in A(i,j)$. For $\mathscr{P}_1$ and $h_1$, if $n\in G_1$ has merit $A(i,j)$ then $c(n)\in A(i',j)$ where $i'=l(c(n))\leqslant h(n)=i$. $\Sigma^*$, on the whole, attempts to generate nodes of merit $A(i,j)$ before attempting to generate nodes of worse merit $A(i',j')\succ A(i,j)$. A node of merit $A(i,j)$ is generated either

(0) by inserting into $A(i,0)$, when $j=0$, a clause $c(n)$ where $l(c(n))=i$ and $g(n)=0$,

(1) by unifying two literals within a clause $c(n) \in A(i+1, j-1)$ or

(2) by resolving a factor $c(n_1) \in A(i_1, j_1)$ with a factor $c(n_2) \in A(i_2, j_2)$ where $n_1$ may be identical to $n_2$ and where

$i = i_1 + i_2 - 2$ and

$j = \max(j_1, j_2) + 1$.

$\Sigma^*$ employs two subalgorithms for generating nodes $n \in G$. The principal subalgorithm, *Fill*$(i, j)$, generates in all possible ways, from nodes already generated, nodes $n$ of merit $A(i, j)$ which have worse merit than all their ancestors. *Fill*$(i, j)$ terminates when all such nodes have been generated. *Fill*$(i', j')$, where $A(i', j')$ is the next cell after $A(i, j)$, begins when *Fill*$(i, j)$ terminates. $\Sigma^*$ begins by invoking *Fill*$(0, 0)$.

Whenever a node $n_0$ is generated by *Fill*$(i, j)$ the second subalgorithm *Recurse*$(c(n_0))$ interrupts *Fill*$(i, j)$ and generates in all possible ways, from nodes already generated, nodes $n$ which are immediate successors of $n_0$ and which are of merit $A(i_1, j_1)$ better than $A(i, j)$. In general whenever a node $n$ is generated, either directly by *Fill*$(i, j)$ or by some call of *Recurse*$(c(n'))$ which is local to *Fill*$(i, j)$, *Recurse*$(c(n))$ generates, from nodes already generated, immediate successors of $n$ which are of better merit than $A(i, j)$. Notice that if $n$ is generated by *Recurse*$(c(n'))$ during *Fill*$(i, j)$ then $n$ has better merit than some ancestor of merit $A(i, j)$. Notice too that the depth of recursion involved in *Recurse*$(c(n'))$ is bounded by the sum $i+j$.

## REMARKS

(1) If $\mathscr{P}$ and $h$ are $\mathscr{P}_2$ and $h_2$ and if $c(n_0)$ is generated directly by *Fill*$(i, j)$ then $c(n_0) \in A(i, j)$ and the only immediate successors of $n_0$ which are of better merit than $A(i, j)$ are nodes $n_1 \in A(i-1, j+1)$. Any such $n_1$ generated by *Recurse*$(c(n_0))$ is obtained either by factoring $c(n_0)$ or by resolving $c(n_0)$ with a unit clause $c(n)$ of level $g(n) \leqslant j$. More generally if $n_0$ is generated by *Recurse* during *Fill*$(i, j)$ and if $c(n_0) \in A(i_0, j_0)$ then it is easy to verify that $i_0 + j_0 = i + j$ and any immediate successor of $n_0$ of better merit than $A(i, j)$ is of merit $A(i_0 - 1, j_0 + 1)$.

(2) If $\mathscr{P}$ and $h$ are $\mathscr{P}_1$ and $h_1$ then $\Sigma^*$ may fail to do upwards diagonal search because of merging, i.e., nodes may be generated by *Recurse* which have worse merit than other candidates for generation. Suppose that $n_0$ is generated by *Fill*$(i, j)$ and that $c(n_0) \in A(i', j)$ where $i' < i$. Suppose that $n_1$ and $n_2$ of merit $A(i'-1, j+1)$ are generated by *Recurse*$(c(n_0))$, $n_1$ before $n_2$. Suppose that $n_3$ of merit $A(i'-1, j+2) \prec A(i, j)$ is generated by *Recurse*$(c(n_1))$. Then $n_2$ has better merit than $n_3$ but $n_3$ is generated before $n_2$ since *Recurse* $(c(n_1))$ must terminate before *Recurse* $c(n_0))$ generates $n_2$.

(3) For both $\mathscr{P}_1, h_1$ and $\mathscr{P}_2, h_2$, $\Sigma^*$ has the desirable property of attempting to resolve every unit clause $c(n_0)$ with all previously generated units $c(n_1)$ as soon as $c(n_0)$ is generated. If $n_0$ is generated during *Fill*$(i, j)$ and if $c(n_0) \in A(1, j_0)$ and $c(n_1) \in A(1, j_1)$ then $A(0, max(j_0, j_1) + 1) \prec A(i, j)$ and an attempt will be made to resolve $c(n_0)$ with $c(n_1)$ during *Recurse*$(c(n_0))$.

(4) Suppose that $Fill(i,j)$ has just begun, then $\Sigma^*$ has not yet generated any nodes of merit worse than $A(i,j)$. Thus if $n$ has merit $A(i,j)$ then either $j=0$ and $g(n)=0$ or $c(n)$ is a resolvent of factors $c(n_1)$ and $c(n_2)$ and both $n_1$ and $n_2$ are of merit better than $A(i,j)$. In order to generate all such nodes $n$ it suffices to attempt to resolve all clauses $c(n_1)$ with clauses $c(n_2)$ where

$$c_1 \in A(l,k),\ c_2 \in A(i-l+2, j-1)$$

for $0 \leqslant k \leqslant j-1$ and $1 \leqslant l \leqslant \frac{i}{2}$ if $i$ is even or $1 \leqslant l \leqslant \frac{i+1}{2}$ if $i$ is odd.

(5) The details for generating nodes during $Recurse(c(n))$ have already been discussed for $\mathscr{P}_2$ and $h_2$ in remark (1). For $\mathscr{P}_1$ and $h_1$ these details are more complicated. Suppose that $n$ has been generated during $Fill(i^*,j^*)$ and that $c(n) \in A(i,j)$. The following procedure will generate, without redundancy, from nodes generated before $n$, immediate successors of $n$ which are of better merit than $A(i^*,j^*)$:

(a) First resolve $c(n)$ with clauses in $A(i',j')$ where $j-1 \leqslant j' \leqslant i^*+j^*-i+2$, in order of decreasing $j'$, and for given $j'$, where $1 \leqslant i' \leqslant i^*+j^*-j'-i+1$ in arbitrary order but preferably in order of increasing $i'$.

(b) Next generate factors of $c(n)$ by attempting to unify, in all possible ways, two literals in $c(n)$.

(c) Finally resolve $c(n)$ with clauses in $A(i',j')$ where $1 \leqslant i' \leqslant i^*+j^*-i-j+1$; $0 \leqslant j' \leqslant j$ in arbitrary order but preferably in order of increasing $i'$.

## ADMISSIBILITY AND OPTIMALITY OF $\mathscr{D}$ AND $\mathscr{D}^u$

Let $\mathscr{P}=(G, s, \mathscr{F}, g)$ be an abstract theorem-proving problem. For $n \in G$ let

$$H(n)=\{g(n^*)-g(n) \mid n^* \in \mathscr{F},\ n \leqslant n^*\},$$
$$h^*(n)=inf\, H(n) \text{ when } H(n) \neq \varnothing,$$
$$h^*(n)=\infty \text{ when } H(n)=\varnothing.$$

Then when $n \leqslant n^*$, for some $n^* \in \mathscr{F}$, $h^*(n)$ is the greatest lower bound on the additional cost over $g(n)$ of $g(n^*)$. The heuristic function $h$ is intended to be an estimate of $h^*$. The only property of $\infty$ needed below is that $k<\infty$ for all real numbers $k$. Since we do not allow $h(n)=\infty$, it is often impossible to construct a heuristic function $h$ which gives a perfect estimate of $h^*$. In particular it is impossible to incorporate into a definition of $h$ any information that a node $n$ is not an ancestor of a node $n^* \in \mathscr{F}$. However such heuristic information can be applied to a problem $\mathscr{P}$ by defining a new problem $\mathscr{P}'$ which differs from $\mathscr{P}$ by containing no such nodes $n$. Alternatively it is possible to allow $h(n)=\infty$ in which case several complexities need to be introduced in preceding definitions (e.g., in the definition of $\delta$-finiteness).

A heuristic function $h$ for $\mathscr{P}$ satisfies the *lower bound condition* for $\mathscr{P}$ if

$$h(n) \leqslant h^*(n) \text{ for all } n \in G,$$

i.e., if $h(n) \leqslant g(n^*)-g(n)$ whenever $n^* \in \mathscr{F}$ and $n \leqslant n^*$. Thus the lower bound condition constrains in effect only the value of $h(n)$ when $n$ is an

ancestor of some solution node. Recall that $h_2$ satisfies the lower bound condition for $\mathscr{P}_2$ while $h_1$ does the same for $\mathscr{P}_1$ except for merging.

Lemma 2 states certain fundamental properties of heuristic functions $h$ satisfying the lower bound condition: (a) $h(n^*)=0$ for $n^* \in \mathscr{F}$, (b) no ancestor of a solution node $n^* \in \mathscr{F}$ has worse diagonal merit than $n^*$, (c) there exists a solution node $n^* \in \mathscr{F}$ having minimum cost in $\mathscr{F}$ if diagonal merit is $\delta$-finite.

*Lemma* 2

Let $\mathscr{P}=(G, s, \mathscr{F}, g)$ be an abstract theorem-proving problem and let the heuristic function $h$ for $\mathscr{P}$ satisfy the lower bound condition.

(a) If $n^* \in \mathscr{F}$ then $h(n^*)=h^*(n^*)=0$ and therefore $f(n^*)=g(n^*)$.
(b) If $n^* \in \mathscr{F}$ and $n \leqslant n^*$ then $f(n) \leqslant f(n^*)$.
(c) If $\preccurlyeq_d$ is $\delta$-finite then for some $n^* \in \mathscr{F}$ $g(n^*) \leqslant g(n)$ for all $n \in \mathscr{F}$, provided $\mathscr{F} \neq \varnothing$.

*Proof.* (a) is obvious, since $H(n^*)=\{0\}$ and $h^*(n^*)=0$.
(b) If $n^* \in \mathscr{F}$ and $n \leqslant n^*$ then $h(n) \leqslant g(n^*)-g(n)$.
But then $f(n)=g(n)+h(n) \leqslant g(n^*)=f(n^*)$.
(c) If $\preccurlyeq_d$ is $\delta$-finite then for all $n \in G$, the set $\{n' \mid f(n') \leqslant f(n), n' \in G\}$ is finite. In particular for $n \in \mathscr{F}$ the set $\{n' \mid g(n') \leqslant g(n), n' \in \mathscr{F}\}$ is finite and therefore contains an element $n^*$ such that $g(n^*)$ is minimal. But then $g(n^*) \leqslant g(n')$ for all $n' \in \mathscr{F}$.

*Theorem* 2

If $\preccurlyeq_d$ is $\delta$-finite for $\mathscr{P}=(G, s, \mathscr{F}, g)$ and if $h$ satisfies the lower bound condition for $\mathscr{P}$ then $\Sigma \in \mathscr{D}(\mathscr{P}, h)$ is admissible for $\mathscr{P}$.

*Proof.* Assume that $\mathscr{F} \neq \phi$. Let $n^* \in \mathscr{F}$ be such that $g(n^*) \leqslant g(n)$ for all $n \in \mathscr{F}$ (such an $n^* \in \mathscr{F}$ exists by Lemma 2(c)). By Theorem 1, $\Sigma$ is complete and therefore there is a stage $i$ such that for some $n$,

$$n \in \mathscr{F} \cap \Sigma_i \quad \text{and} \quad \mathscr{F} \cap \Sigma_{i-1} = \varnothing.$$

Suppose that $\Sigma$ is not admissible for $\mathscr{P}$. Then $g(n^*) < g(n)$. But, by Lemma 2, for all $n' \leqslant n^*$, $f(n') \leqslant f(n^*) = g(n^*) < f(n)$. So $f(n') < f(n)$ for all $n' \leqslant n^*$. But then $n' \prec n$ for all $n' \leqslant n^*$. By Lemma 1 (a), $n^* \in \Sigma_{i-1}$ and therefore $\mathscr{F} \cap \Sigma_{i-1} \neq \varnothing$, contrary to assumption.

Theorem 2 specializes to a generalization of Theorem 1 in Hart, Nilsson and Raphael (1968) when $s(G')=\varnothing$ for all $G' \subseteq G$ which are not singletons. In particular it is not necessary to require that $\mathscr{S}_0$ be finite or that $g(n)$ be strictly greater than $g(n')$ whenever $n' < n$. Since the specialization yields a tree representation of graph search, it is unnecessary to distinguish between the cost $g(n)$ and the total cost along some minimal path to $n$.

Figure 4 illustrates Lemma 2 and Theorem 2. $\mathscr{P}$, $\Sigma$, $n^*$, $n'$, $n$ and $n''$ are as in figure 3, but $h$ satisfies the lower bound condition. By Lemma 2, $n'$ lies on the same diagonal $d$ as does $n^*$. $\Sigma$ is admissible since any $n^{**} \in \mathscr{F}$ having

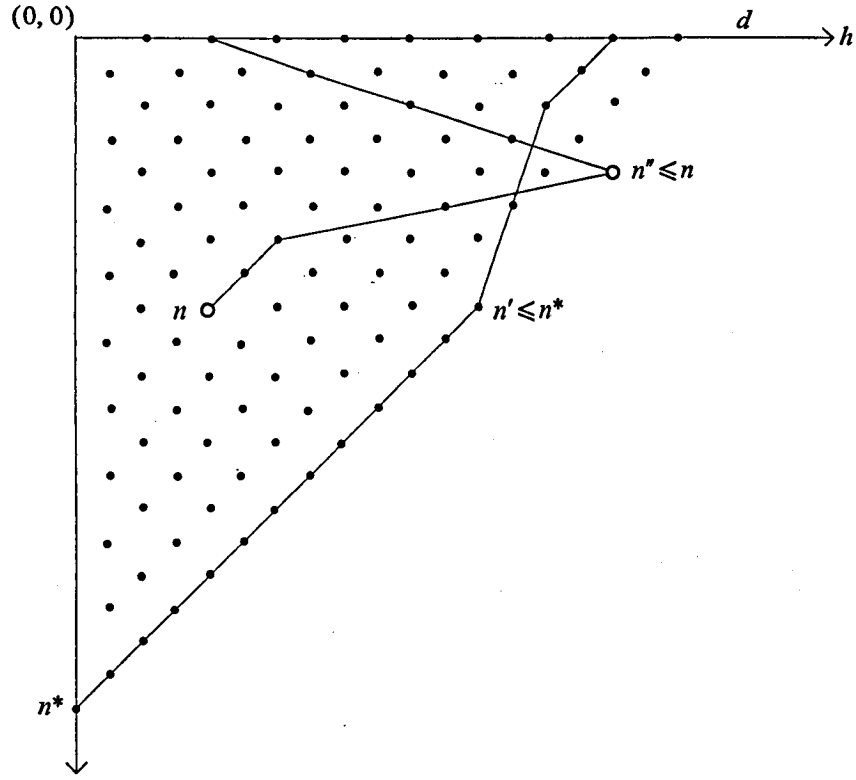



Figure 4

worse merit than $n^*$ lies on a diagonal to the right of $d$ and is not generated before $n^*$. It is still possible for a node $n \in G$ to have better merit than $n^*$ and not be generated before $n^*$ because $n''$ has worse merit than $n'$.

To prove the appropriate extension of the Hart–Nilsson–Raphael Theorem on the optimality of $\Sigma \in \mathscr{D}^u$, we need to formulate an assumption equivalent to their 'consistency assumption'. The reader familiar with Hart, Nilsson and Raphael (1968) will easily convince himself that the following condition is equivalent to the consistency assumption. We say that the evaluation function $f$ satisfies the *monotonicity condition* if

$$f(n') \leqslant f(n) \text{ for } n' \leqslant n \text{ and}$$
$$f(n^*) = g(n^*) \text{ for } n^* \in \mathscr{F}.$$

(The first condition is equivalent to

$$h(n) \geqslant h(n') - (g(n) - g(n')) \text{ for } n' \leqslant n.)$$

Notice that for $\mathscr{P}_2$ the evaluation function $f_2 = g_2 + h_2$ satisfies the monotonicity condition whereas for $\mathscr{P}_1$ the function $f_1 = g_1 + h_1$ is monotonic except for merging.

Figure 5 illustrates upwards diagonal search when the function $f$ satisfies the monotonicity condition. $\mathscr{P}$, $\Sigma$, $n^*$, $n'$, $n$ and $n''$ are as in figures 3 and 4. By Lemma 3, $h$ satisfies the lower bound condition and therefore $\Sigma$ is admissible and $n'$ lies on the same diagonal as $n^*$. The monotonicity condition implies that if $n$ has better diagonal merit than $n^*$ then all ancestors of $n$ have better merit than $n^*$ and therefore, by Lemma 1, $n$ is generated before $n^*$.

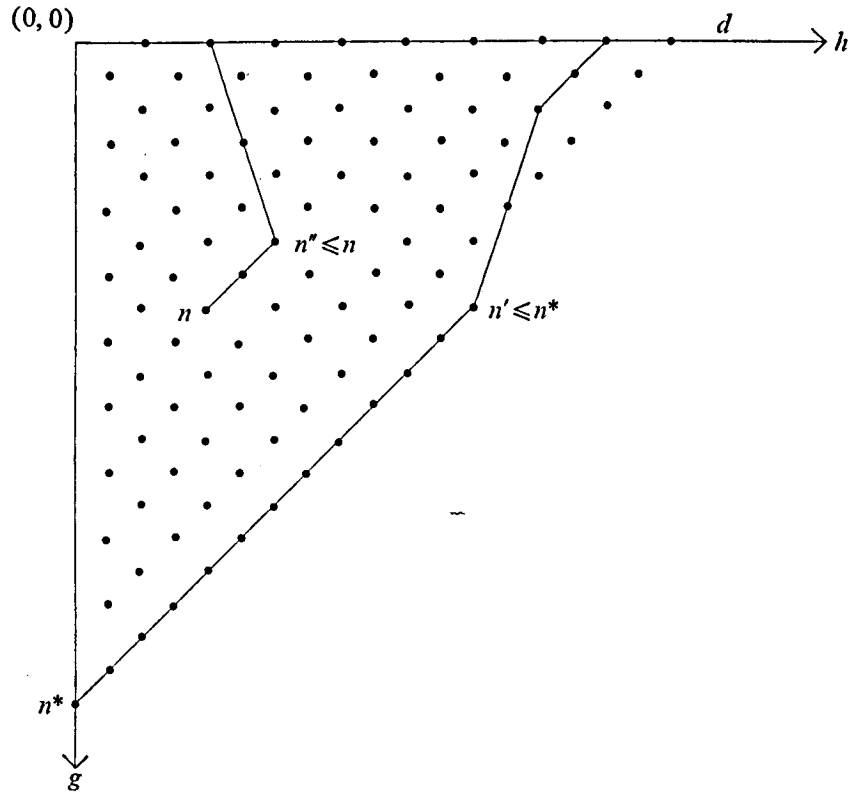



Figure 5

*Lemma* 3

Let $\mathscr{P}=(G, s, \mathscr{T}, g)$ be an abstract theorem-proving problem, let $h$ be a heuristic function for $\mathscr{P}$, and let $f$ satisfy the monotonicity condition, where $f(n)=g(n)+h(n)$, $n \in G$. Then

(a) $h$ satisfies the lower bound condition,
(b) if $\Sigma \in \mathscr{D}(\mathscr{P}, h)$, $n_1 \in \Sigma_i$ and $n_2 \in \Sigma(\Sigma_i)$ then $f(n_1) \leqslant f(n_2)$.

*Proof.* (a) $h$ satisfies the lower bound condition if $h(n) \leqslant g(n^*)-g(n)$ whenever $n^* \in \mathscr{T}$ and $n \leqslant n^*$.

But monotonicity of $f$ implies that

$$f(n)=g(n)+h(n)\leqslant f(n^*)=g(n^*).$$

So $\quad h(n)\leqslant g(n^*)-g(n).$

(b) Suppose the contrary, namely that $n_1\in\Sigma_i$, $n_2\in\Sigma(\Sigma_i)$ and $f(n_1)>f(n_2)$. But then, since $f(n')\leqslant f(n_2)<f(n_1)$ for all $n'\leqslant n_2$, it follows that $n'\prec n_1$ for all $n'\leqslant n_2$. By Lemma 1(a), $n_2\in\Sigma_{i-1}$, contradicting the assumption that $n_2\in\Sigma(\Sigma_i)$.

For the case of ordinary graphs, the optimality theorem (Theorem 2) of Hart, Nilsson and Raphael (1968) compares, in effect, search strategies $\Sigma\in\mathscr{D}(\mathscr{P},h)$ with strategies $\Sigma'\in\mathscr{D}(\mathscr{P},h')$ where $h'(n)\leqslant h(n)$ for all $n\in G$ and where $f=g+h$ is monotonic. [In Hart, Nilsson and Raphael (1968) the search strategy $\Sigma'$ is assumed only to be 'no better informed' than $\Sigma$ – we interpret this to mean that $h'(n)\leqslant h(n)$ and $\Sigma'\in\mathscr{D}(\mathscr{P},h)$.] If $\Sigma_i$ and $\Sigma'_{i'}$ are the first sets which contain nodes $n^*\in\mathscr{F}$ then $\Sigma_i\subseteq\Sigma'_{i'}\cup G'$ where $G'$ is the set of nodes $n\in\Sigma_i$ which have diagonal merit equal to $n^*\in\Sigma_i\cap\mathscr{F}$, i.e., before termination $\Sigma'$ generates all the nodes generated by $\Sigma$ except possibly for unlucky choices by $\Sigma$ of nodes tied for merit with the solution node $n^*\in\Sigma_i$. Theorem 3 below generalizes Theorem 2 of Hart, Nilsson and Raphael (1968) and implies in addition that $\mathscr{D}^u$ is an optimal subclass of $\mathscr{D}$.

It should be noted that the monotonicity condition on $f$ in Theorem 3 can be replaced by the lower bound condition on $h$ with the result that $\Sigma'$ may now fail to generate nodes in the larger set $G'$ of nodes $n\in\Sigma_i$ where some $n''\leqslant n$ has diagonal merit tied with the solution node $n^*\in\Sigma_i$. A special case of this modification of Theorem 3 is illustrated by the example of figure 6, following the proof of Theorem 3.

*Theorem* 3

Let $\mathscr{P}=(G,s,\mathscr{F},g)$ and let $h$ and $h'$ be heuristic functions for $\mathscr{P}$ such that

$$h'(n)\leqslant h(n) \text{ for } n\in G.$$

Let $f(n)=g(n)+h(n)$ and $f'(n)=g(n)+h'(n)$. Suppose that $f$ is monotonic. Given $\Sigma\in\mathscr{D}^u(\mathscr{P},h)$ and $\Sigma'\in\mathscr{D}(\mathscr{P},h')$, suppose that

$$n_1\in\mathscr{F}\cap\Sigma_i,\ \mathscr{F}\cap\Sigma_{i-1}=\varnothing,$$
$$n_2\in\mathscr{F}\cap\Sigma'_{i'} \text{ and } \mathscr{F}\cap\Sigma'_{i'-1}=\varnothing.$$

Then $\Sigma_i\subseteq\Sigma'_{i'}\cup G^*$ where

$$G^*=\{n\mid n\in\Sigma_i \text{ and for some } n'\leqslant n_1,\ f(n)=f(n')=f(n_1) \text{ and } h(n)\leqslant h(n')\}.$$

*Proof.* $\Sigma'$ satisfies the lower bound condition since $h'(n)\leqslant h(n)$ for all $n\in G$ and since $\Sigma$ satisfies the lower bound condition. Therefore both $\Sigma$ and $\Sigma'$ are admissible and $g(n_1)=g(n_2)$, $f(n_1)=f(n_2)$. Suppose that $n\in\Sigma_i$ and that $n\notin\Sigma'_{i'}$. It suffices to show that $n\in G^*$.

By Lemma 1(b), $n\in\Sigma_i$ implies that $n\leqslant_{dw}n'$ for some $n'\leqslant n_1$. But by Lemma 3(b), since $f$ is monotonic

$f(n) \leqslant f(n_1)$,
$f(n') \leqslant f(n_1)$,
$f(n'') \leqslant f(n)$ for all $n'' \leqslant n$.

But $h'(n'') \leqslant h(n'')$ implies

$f'(n'') \leqslant f(n'')$. So
$f'(n'') \leqslant f(n)$ for all $n'' \leqslant n$.

Also $n \notin \Sigma_{i'}$ and $n_2 \in \Sigma_{i'}$ imply by Lemma 1(a) that for some $n'' \leqslant n$, $n'' \geqslant_d n_2$, i.e.,

$f'(n'') \geqslant f'(n_2) = f(n_1)$. So
$f(n) \geqslant f(n_1)$ and
$f(n) = f(n_1)$.

$n \leqslant_{au} n'$ implies

$f(n) \leqslant f(n') \leqslant f(n_1)$. So
$f(n) = f(n') = f(n_1)$ and
$h(n) \leqslant h(n')$, i.e.,
$n \in G^*$.

Figure 6 compares nodes generated, before the generation of a given $n^* \in \mathscr{F}$, by different search strategies $\Sigma_i \in \mathscr{D}(\mathscr{P}, h_i)$ for a fixed problem $\mathscr{P} = (G, s, \mathscr{F}, g)$ and for different heuristic functions $h_i$. $h_1(n)$ is assumed to be a greatest lower bound on the value of $h^*(n)$ when $n \leqslant n^*$, where $n^*$ has least cost in $\mathscr{F}$. Nodes $n \in G$ are represented as points with co-ordinates $(h_1(n), g(n))$. The node $n'$ has worst upwards diagonal merit in the set consisting

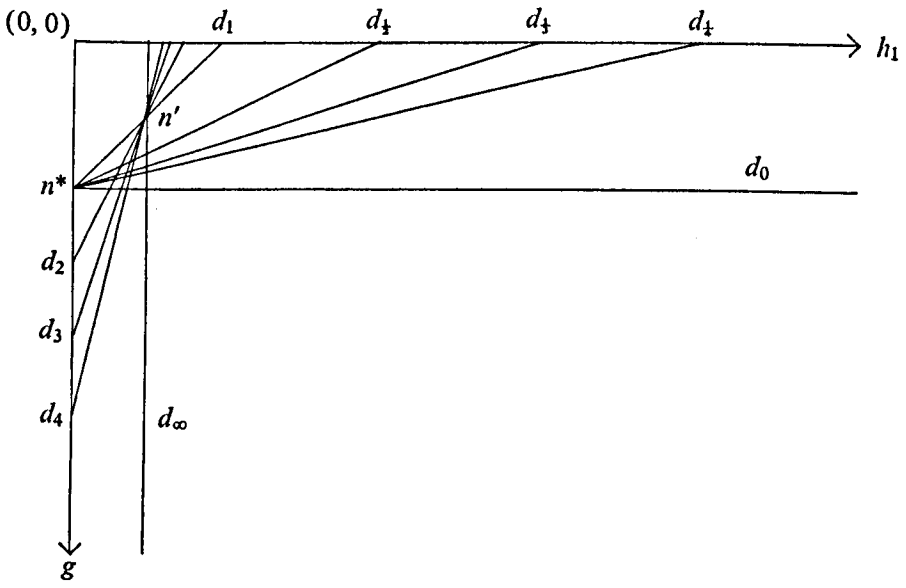



Figure 6

of $n^*$ and the ancestors of $n^*$. The functions $h_i$ are defined by $h_i(n) = ih_1(n)$ for all $n \in G$, $0 \leqslant i \in \mathbb{R}$.

For $0 \leqslant i \leqslant 1$, $h_i$ satisfies the lower bound condition for $\mathscr{P}$ and $\Sigma_i$ is admissible for $\mathscr{P}$. $\Sigma_i$ need not be admissible for $\mathscr{P}$ when $i > 1$. The area to the left of the line $d_i$ contains nodes generated by $\Sigma_i$ before the generation of $n^*$. For $0 \leqslant i \leqslant 1$, $\Sigma_i$ generates all the nodes generated by $\Sigma_1$. For $i > 1$, $\Sigma_i$ generates all the nodes left of $d_i$ which have been generated by $\Sigma_1$. No $\Sigma_i$ is more efficient than $\Sigma_1$, if $i > 1$. Some $\Sigma_i$ may generate fewer nodes than $\Sigma_1$, if $i > 1$, but this possibility becomes more remote as $i$ increases. However even for large $i$, $\Sigma_i$ may be more efficient than $\Sigma_1$ for generating solution nodes of arbitrary cost. A more thorough analysis of relationships similar to those discussed here has been made by Ira Pohl (1969, 1970).

**Acknowledgements**

The author wishes to acknowledge helpful discussions with Dr Bernard Meltzer, Dr Ira Pohl, Miss Isobel Smith, Mr Pat Hayes and Mr Donald Kuehner. Special thanks are due to Miss Isobel Smith for implementing diagonal search for resolution problems and to Dr Nils Nilsson and Mr Donald Kuehner for suggestions made for improving an earlier draft of this paper.

This research was supported by an IBM fellowship and grant from Imperial College, and more recently by the Science Research Council.

## REFERENCES

Burstall, R.M. (1968) A scheme for indexing and retrieving clauses for a resolution theorem-prover. *Memorandum* MIP-R-45. University of Edinburgh: Department of Machine Intelligence and Perception.

Doran, J. & Michie, D. (1966) Experiments with the graph traverser program. *Proceedings of the Royal Society (A)*, **294**, 235–59.

Green, C. (1969a) 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.

Green, C. (1969b) The application of theorem-proving to question-answering systems Ph.D. thesis. Stanford University.

Hart, P.E., Nilsson, N.J. & Raphael, B. (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Sys. Sci. and Cybernetics*, July 1968.

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.

Kowalski, R. (1970) Studies in the completeness and efficiency of theorem-proving by resolution. Ph.D. thesis. University of Edinburgh.

Nilsson, N.J. (1968) Searching problem-solving and game-playing trees for minimal cost solutions. *IFIPS Congress preprints*, H125–H130.

Pohl, I. (1969) Bi-directional and heuristic search in path problems. Ph.D. thesis. Stanford University.

Pohl, I. (1970) First results on the effect of error in heuristic search. *Machine Intelligence 5*, pp. 219–36 (eds Meltzer, B. & 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.

Sandewall, E. (1969) Concepts and methods for heuristic search. *Proc. of the International Joint Conference on Artificial Intelligence*, pp. 199–218 (eds. Walker, D. E. & Norton, L. N.).

Sibert, E.E. (1969) A machine-oriented logic incorporating the equality relation. *Machine Intelligence 4*, pp .103–33 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Wos, L.T., Carson, D.F. & Robinson, G.A. (1964) The unit preference strategy in theorem-proving. *AFIPS*. **25**, 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.

# 11

# An Experiment in Automatic Induction

R. J. Popplestone
Department of Machine Intelligence and Perception
University of Edinburgh

## INTRODUCTION

The problem discussed in this paper, namely that of finding a function to satisfy a given argument-value table, is by no means new to computing science, or to mathematics. Thus, for example, the problem of fitting a curve to a set of points is a part of numerical analysis. However, I am concerned with finding a function over a non-metric space, and so my work is closer to that of Feldman *et al.* (1969) in what they call 'grammatical inference' or to the automaton-synthesizing programs described by Fogel, Owens and Walsh (1966).

The idea of putting together attributes with boolean connectives, which is part of the apparatus available to the induction engines described in this paper, is to be found in the psychological literature and referred to as 'concept formation', for instance, *see* Bruner, Goodnow and Austin (1956).

There have been some applications of learning devices. Perhaps the best known is Samuel's checkers program (Samuel 1967), but Murray and Elcock (1968) have a system for describing generalized board states in Go-Moku that employs a much richer language to describe the concepts learnt.

Relevant aspects of the problem are mentioned in McCarthy and Hayes (1969).

## INDUCTION ENGINES

Induction is going from the particular to the general. Denote an induction engine by $E$. Here follows a definition of $E$ that is sufficiently formal for my purpose.

Suppose there are two sets $D$ and $R$, and let $F$ be a set of functions from $D$ to $R$, so that $f \in F$ can be regarded as a subset of $D \times R$. Then $E$ is a

mapping which takes a finite subset $s$ of $f$ onto a function $E(s)$ from $D$ to $R$ so that the following conditions hold:

$IE1$ $\quad s \subset E(s)$ that is, $E(s)$ agrees with $f$ on $s$

$IE2$ $\quad \exists s'(s' \in D \times R \;\&\; s' \subseteq f \;\&\; s' \subseteq E(s) \;\&\; s' \gg s)$

where $\gg$ means 'is significantly greater than'. What $IE2$ really says is that $E(s)$ must agree with $f$ over a set significantly larger than $s$ and by significant I mean that a human would prefer to have a function constructed by an induction engine than to construct and program one himself.

The notation may be informally summarized as follows:

$F$ defines the family of functions over which the induction engine is to operate.

$f$ is a function to be guessed.

$s$ is a sample of $f$, in the form of an input–output table.

$E(s)$ is a function synthesized from $s$, – i.e., it is the induction engine's guess and must satisfy the stated conditions $IE1$ and $IE2$.

This definition is akin to that of the convergence of a sequence of functions in classical mathematics, but is in difficulties over defining an adequate metric. In some circumstances, e.g., for perceptrons with $F$ being the set of linearly separable functions and Feldman's grammar learning programs (Feldman *et al.* 1969), a strong condition

$IE3$ $\quad \exists s' \; E(s') = f \;\&\; \mathit{finite}(s')$

can be imposed.

However, if we use a human as an induction engine, we would expect him to satisfy $IE1$ and $IE2$, but not to satisfy $IE3$. That is to say, there would be functions he could not guess definitions for.

For the further discussion of induction engines the concept of *language* is required. A language $L$ is a pair $(T, I)$ where $T$ is a set of *sentences* and $I$ is a function which takes $t \in T$ onto $I(t) \in F$. $I$ is called the interpreter for the language.

Consider, for example, an Adaline (Widrow 1962). This is a simple linear threshold device. Suppose there are $n$ inputs. Then $T$ is the set of sequences of $n+1$ real numbers forming the weights $(w_1, w_2 \ldots w_{n+1})$. $D$, the set of possible inputs, is the Cartesian product $\prod_1^n B$ where $B$ is the set $\{0, 1\}$. Thus a typical member of $D$ would be $(d_1, d_2 \ldots d_n)$. $R$, the set of outputs, is $B$. Let $(w_1 \ldots w_{n+1})$ be a member of $T$, that is to say, a sentence, then

$$I(w_1 \ldots w_{n+1}) = \lambda d \textbf{ if } w_1 d_1 + w_2 d_2 + \ldots w_n d_n > w_{n+1} \textbf{ then } 1 \textbf{ else } 0$$

where $d_1 \ldots d_n$ are the components of $d$. An adaline training procedure is then an induction engine for adalines.

A concept of importance is the *power* of a language. This is simply the range of the interpreter. In particular, if $I$ ranges over all computable functions then the language is said to have full power. Thus if $T$=all LISP expressions

and $I$=a LISP interpreter then we have a language with full power. On the other hand, an Adaline does not have full power.

## AN INDUCTION PROGRAM

In this section I will describe a program to realize an induction engine. The domain $D$ is a set of *boards*, where a board is an $n \times n$ array with entries taken from a set of symbols, $\{.\} \cup A$, where $A$ is a finite set. $n$ is fixed but arbitrary. "." is called the null symbol. Thus for tic-tac-toe $A = \{X, 0\}$. The range $R$ is the set of truth values which we shall take to be (0, 1).

The language $L$ consists of sentences of the predicate calculus formed by the rules *I0–I6* below, and interpreted by a special purpose interpreter. Since the domain is finite, the system is decidable. The sentence formation rules are as follows:

*I0* There is a set $P$ of *primitive* sentences contained in $T$. $P$ consists of sentences such as $occ(1, 2, 'X')$ where $occ$ is a predicate meaning 'is occupied by', so that the sentence $occ(1, 2, 'X')$ means 'square 1, 2 is occupied by an $X$'.

*I1* $t \in T \Rightarrow t \,\&\, t' \in T$

*I2* $t \in T \Rightarrow t \vee t' \in T$

where either (i) $t' \in T$ or (ii) for some predicate $p$, $t'$ is $p(c_1 \ldots c_n)$ where the $c_i$ are constants one of which occurs in $t$.

*I3* $t \in T \Rightarrow \neg t \in T$

*I4* $\mathscr{S}(c) \in T \Rightarrow \exists x_\alpha (x_\alpha \in C(c) \,\&\, \mathscr{S}(x_\alpha)) \in T$

*I5* $\mathscr{S}(c) \in T \Rightarrow \forall x_\alpha (x_\alpha \in C(c) \,\&\, \mathscr{S}(x_\alpha)) \in T$

In *I4* and *I5* $\mathscr{S}(c)$ means a sentence containing the constant $c$, and $C(c)$ means a set containing $c$. Thus if $c$ is a number then $C(c)$ would be the set of numbers $\{1, 2, \ldots, n\}$ $n$ being the size of the board. *I4* and *I5* provide the means of generalizing from a statement about individuals to a statement about a class containing those individuals. $x_\alpha$ is a symbol not occurring in $\mathscr{S}(c)$.

Finally

*I6* $\mathscr{S}(c) \in T \Rightarrow \mathscr{S}(f(c_1 \ldots c_n)) \in T$

where $f$ is drawn from a set of standard functions. Thus *I6* would convert $occ(1, 3, 'X')$ into $occ(1, 1+2, 'X')$.

There are two possible interpretations of *I4* and *I5*. These differ in whether abstraction is of a constant or an instance of a constant. Thus in one interpretation $occ(1, 1, 'X')$ would only give rise to $\exists x_\alpha\ occ(x_\alpha, x_\alpha, 'X')$ by abstraction on 1 using *I4*, whereas in the other, one would also get $\exists x_\alpha\ occ(x_\alpha, 1, 'X')$ and $\exists x_\alpha\ occ(1, x_\alpha, 'X')$. In existing versions of the program, the first interpretation is used.

Two programs have been written which form sentences according to *I0* to *I6*. They are referred to as the Mark-1 and Mark-2 programs.

### THE MARK-1 PROGRAM

This is an induction engine which uses rules $I0$, $I1(i)$, $I2(i)$, $I3$, $I4$ and $I5$ as follows. A set of current base sentences ($CBS$) is maintained. This is initially the primitives of $I0$. Let $s$ be a 'sample' of argument–value pairs (as section 1.) An algorithm called ORDISCRIM is entered which produces, using the propositional rules $I1(i)$, $I2(i)$, and $I3$, a sentence $t$ such that $s \subseteq I(t)$ where $I$ is the interpreter. Thus ORDISCRIM itself satisfies $IE1$ (or more exactly ORDISCRIM $\circ$ $I$ where $\circ$ means function product).

It will also in some measure satisfy $IE2$, because it has a built-in preference for simple explanations, and so will not specify that particular primitives need to be true (or false) if they are not needed for $IE1$. Thus, from Occam's Razor, we would expect the sentence formed by ORDISCRIM to work for some wider class than $s$.

Notice that ORDISCRIM performs a task equivalent to that of designing a digital logic circuit from AND, OR, and NOT gates which produces an output from a set of inputs, where the input–output relationship is tabulated. The inputs correspond to the primitive sentences of $I0$, and the input–output table to the set $s$.

The sentence resulting from the application of the propositional rules is then decomposed into all its subsentences, and the generalizing rules $I4$ and $I5$ are applied to these, and a new $CBS$ is made from the union of the result of this application of $I4$ and $I5$ and the primitives. ORDISCRIM is then applied to the new $CBS$ and the process repeated until no new result is produced by ORDISCRIM.

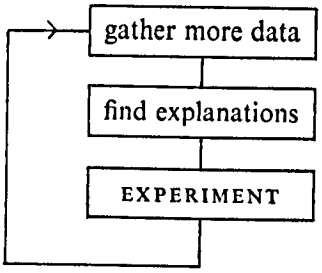
The above description is of the action of the engine when it is working on the first sample set. When working on a sequence of examples, the $CBS$ for the final explanation of one is carried over to the next. This means that it is possible for the engine to find a sentence $s$ for which $I(s) = f$ via a sequence of examples graded in difficulty which could not be found if the engine were presented with the last example straight off.

An annotated run of the mark-1 program is to be found in Appendix 1. In this example, the board is $3 \times 3$ and $f$, the function to be guessed, is the property of there being three $X$s in a line, either row, column, or diagonal.

The limitations of the Mark-1 program were fairly obvious. Firstly, the series of examples by which it 'learned' a complex function had to be carefully chosen. Thus in the example quoted above, it was necessary to get an adequate understanding of a horizontal line before trying to express a vertical line. Also, since the generalization processes, $I4$ and $I5$, worked on the output of ORDISCRIM there was no possibility of redress if ORDISCRIM seized on features of particular boards which did not generalize correctly. Thus it was not possible to get correct generalizations if the board contained other non-null symbols than $X$. Finally, there is a severe restriction in expressive power through the inability to introduce predicates other than *occ*. To overcome some of these limitations, Mark-2 was written.

## THE MARK-2 PROGRAM

The program has the flow-diagram shown in the figure.




In the 'gather more data' phase, the program reads-in some more argument–value pairs to add to the sample. In contrast to the Mark-1 program where all data is entered in this way, relatively little is entered thus.

In the 'find explanations' box, the Graph Traverser (Doran and Michie 1966, Doran 1968) is used to search for an explanation. This is done as follows.

Nodes are sentences in $T$. $I1$ to $I5$ are used as rules to generate sentences, and thus form the basis of the 'develop' function. The primitives are used as starting nodes. For any node $t$, the goodness of $t$ is measured by

(1) how well $I(t)$ agrees with $f$ on $s$

(2) the complexity of $t$.

In the present program, selection of $t'$ for rules $I1(i)$ and $I2(i)$ is done by using the best $k$ nodes to date, for some fixed $k$. This is not optimal, and better methods can be devised by considering the algorithm ORDISCRIM described above.

The Graph Traverser runs until certain limits either of space or of time are exceeded. The routine EXPERIMENT is then entered.

The EXPERIMENT routine consists in examining the nodelist, as follows. If the best two nodes $t_\alpha$ and $t_\beta$ are not both explanations, the routine exits. Otherwise the sentence $t = t_\alpha \,\&\, \neg t_\beta$ is formed, and a Beth tree theorem prover (Popplestone 1967) is entered to find a board position for which $t$ is true. If there is none (and since the domain is finite this is decidable) then $t' = \neg t_\alpha \,\&\, t_\beta$ is formed, and again a board position for which $t'$ is true is searched for. If, again, there is none, then $t_\alpha \equiv t_\beta$ and the sentence with the higher score (the 'worse' sentence) is discarded, and the process repeated with the new nodelist. Otherwise, if a position satisfying $t$ or $t'$ has been found, it is printed out, and the user is asked what the value of $f$ is for it. All nodes are then re-evaluated with the new sample, and EXPERIMENT is re-entered.

The EXPERIMENT routine is important because it enables the machine to decide between hypotheses in the manner of a scientist thinking of a crucial experiment.

**Acknowledgement**

This work was made possible by facilities supplied through a research grant from the Science Research Council.

## REFERENCES

Bruner, J.S., Goodnow, J.J., & Austin, G.A. (1956) *A Study of Thinking*. New York: John Wiley & Sons.

Doran, J.E. & Michie, D. (1966) Experiments with the Graph Traverser program. *Proc. R. Soc.* (A), **294**, 235–59.

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.

Feldman, J., Gips, J., Horning, J.J., & Reder, S. (1969) Grammatical complexity and inference. *Technical Report No. CS* 125, Stanford Artificial Intelligence Project, Stanford, California.

Fogel, L.J., Owens, A.J. & Walsh, M.J. (1966) *Artificial Intelligence through simulated evolution*. New York: John Wiley & Sons.

McCarthy, J. & Hayes, P. (1969) Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, pp. 463–502 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Murray, A.M. & Elcock, E.W. (1968) Automatic description and recognition of board patterns in Go-Moku. *Machine Intelligence 2*, pp. 75–88 (eds Michie, D. & Dale, E.). Edinburgh: Oliver and Boyd.

Popplestone, R.J. (1967) Beth Tree methods in automatic theorem proving. *Machine Intelligence 1*, pp. 119–35 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Samuel, A.L. (1967) Some studies in machine learning using the game of checkers – 2 – recent progress. *IBM Journal Research and Dev.*, **11**, 601–17.

Widrow, B. (1962) Generalization and information storage in networks of adaline neurons. *Self-Organizing Systems* – 1962, pp. 435–61 (eds Yovitz, Jacobi & Goldstein). Washington D.C.: Spartan Books.

## APPENDIX A ABBREVIATED AND ANNOTATED OUTPUT

**The program is here learning to recognize a win at tic–tac–toe.**

| | |
|---|---|
| ** 'TYPE BOARD' | The experimenter types in first position |
| : X . . | |
| : . . . | |
| : . . . | |
| ** 'I SAY WIN' : WRONG | The machine guesses, on no evidence, that this is a winning position and the experimenter tells it that it is wrong. |
| ** [ ORF ] | This is the machine's next explanation which is simply that win($s$) is false for every state $s$. |

```
** 'TYPE BOARD'
: X X X
: . . .
: . . .
```

This time the machine is presented with a winning position and it guesses that it is not a winning position (because its current explanation is that all positions are not winning positions).

```
** 'I SAY NOT WIN': WRONG
```

It is again corrected by the experimenter.

```
** [ ORF [ ANDF [ EXISTS 9400000A [ 1 2 3 ]
ALL 8400000A [ 1 2 3 ] LAMBDA SS OCC SS
9400000A 8400000A [ QUOTE . X ] ] ] ]
```

Its next explanation is that for a win there must be an *i* such that for all *j*, square *ij* is occupied by an X, that is to say, one row of the board must be occupied by Xs.

```
** 'TYPE BOARD'
: . . .
: X X X
: . . O
** 'I SAY WIN': RIGHT
```

The next example confirms this hypothesis.

```
** 'TYPE BOARD'
: X . .
: X . .
: X . .
** 'I SAY NOT WIN': WRONG
```

But this one does not.

```
** [ ORF [ ANDF [ ALL 4500000A [ 1 2 3 ]
EXISTS 3500000A [ 1 2 3 ] LAMBDA SS OCC
SS 4500000A 3500000A [ QUOTE . X ] ] ]
[ ANDF [ ALL 5500000A [ 1 2 3 ] EXISTS
2500000A [ 1 2 3 ] LAMBDA SS OCC SS
2500000A 5500000A [QUOTE . X ] ] ] ]
```

The next explanation offered by the machine is ingenious but wrong, namely that either there must be an X in every row or there must be an X in every column.

```
** 'TYPE BOARD'
: O X O
: . X .
: . X O
** 'I SAY WIN': RIGHT
** 'TYPE BOARD'
: X . .
: . X .
: . . X
** 'I SAY WIN': RIGHT
```

```
** 'TYPE BOARD'
: X . .
: . . .
: . . X
** 'I SAY NOT WIN': RIGHT

** 'TYPE BOARD'
: X . .
: X . .
: . X .
** 'I SAY WIN': WRONG

** [ ORF [ ANDF [ EXISTS 4600000A [ 1 2 3 ]
ALL 0600000A [ 1 2 3 ] LAMBDA SS OCC SS
4600000A 0600000A [ QUOTE . X ] ] ]
[ ANDF [ EXISTS 3600000A [ 1 2 3 ] ALL
1600000A [ 1 2 3 ] LAMBDA SS OCC SS
1600000A 3600000A [ QUOTE . X ] ] ] [ ANDF
[ LAMBDA SS OCC SS 2 2 [ QUOTE . X ] ] ] ]

** 'TYPE BOARD'
: . . .
: . X .
: . . X
** 'I SAY WIN': WRONG

** [ ORF [ ANDF [ ALL 2700000A [ 1 2 3 ]
LAMBDA SS OCC SS 2700000A 2700000A
[ QUOTE . X ] ] ] [ ANDF [ EXISTS 8700000A
[ 1 2 3 ] ALL 4700000A [ 1 2 3 ] LAMBDA SS
OCC SS 8700000A 4700000A [ QUOTE . X ] ] ]
[ ANDF [ EXISTS 7800000A [ 1 2 3 ] ALL
5700000A [ 1 2 3 ] LAMBDA SS OCC SS
5700000A 7700000A [ QUOTE . X ] ] ] ]

** 'TYPE BOARD'
: . . X
: . X .
: X . .
** 'I SAY NOT WIN': WRONG
```

The machine is only disillusioned here,

and now offers the explanation that there must either exist a row of xs or a column of xs or that square 2 2 must be occupied with an X

which is immediately faulted

Finally, an explanation is produced which says that there is a win if there are xs in the leading diagonal or there is a row or column occupied by xs.

At this stage the time-sharing system failed.

## APPENDIX B A RUN OF THE MARK 2 PROGRAM

```
** TYPE BOARD
: X X X
: . . .
: . . .
```
The human types in a board state.

```
** IS THIS AN INSTANCE: YES
```
And says it is an instance of the concept.

```
** DO YOU WANT TO TELL ME MORE: YES
```
The concept (property) to be guessed is 'there is a row or a column or a diagonal of xs'.

```
** TYPE BOARD
: X . .
: . . .
: . . .
```
Then he types in another.

```
** IS THIS AN INSTANCE: NO
```
And says it is not an instance.

```
** DO YOU WANT TO TELL ME MORE: NO
```
The program then starts work

```
** [ 21.19 16 SEPT 1969 ]
```

```
** INDDEVEL
t1: OCC ( 1, 3, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
```
Hypothesis t1 – square 1, 3 is occupied by an X.

```
** INDDEVEL
t2: OCC ( 1, 2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
```
Hypothesis t2 – square 1, 2 is occupied by an X.

```
** INDDEVEL
t3: EXISTS A55000 ( 1 2 3 ) OCC ( A55000,
2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
```
Hypothesis t3 – there is an X in column 2.

```
** MAXNJOBS
```
Space bound exceeded, exit from Graph Traverser.

```
** CHECK JOBLIST
t1: OCC ( 1, 3, QUOTE ( X ) ) 16
t2: OCC ( 1, 2, QUOTE ( X ) ) 16
t3: EXISTS A55000 ( 1 2 3 ) OCC ( A55000,
  2, QUOTE ( X ) ) 17
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) ) 17
t5: EXISTS A45000 ( 1 2 3 ) OCC ( A45000,
  3, QUOTE ( X ) ) 17
```
Print best 5 hypotheses.

```
** EXPERIME
t1: OCC ( 1, 3, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
t2: OCC ( 1, 2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
. . X
. . .
. . .
```

*Experiment.* The machine produces a state for which t1 & ¬t2 is true.

** IS THIS AN INSTANCE: NO

The human says it is not an instance of the concept, thus contradicting t1.

```
** EXPERIME
t2: OCC ( 1, 2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
t4: ALL A45000 ( 1 2 3 ) OCC ( 1, A45000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
. X .
. . .
. . .
```

t4 says:
'Row 1 is full of xs'.

Machine produces state of board for which t2 & ¬t4 is true.

** IS THIS AN INSTANCE: NO

Human says 'not an instance' so contradicting t2 and also t3.

```
** EXPERIME
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
t5: ALL A45000 ( 1 2 3 ) OCC ( 1, A45000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
```

*Experiment* cannot find a state for which t4 & ¬t5.

or for which t5 & ¬t4 and so concludes t5≡t4

```
** EXPERIME
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
t6: ALL A65000 ( 1 2 3 ) EXISTS A55000
  ( 1 2 3 ) OCC ( A55000, A65000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 18
. X X
X . .
. . .
```

t6 says 'there is an X in every column' – so this board is t6 & ¬t4

```
** IS THIS AN INSTANCE: NO
** DO YOU WANT TO TELL ME MORE: NO
```

*Experiment* is finished, and no more boards are given manually. So start Graph Traverser.

```
** [ 21.23 16 SEPT 1969 ]
** INDDEVEL
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
** INDDEVEL
t7: EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) )
TRUECOUN 1 TARGCOUN 1INDVALOF 18
** MAXNJOBS
** CHECK JOBLIST
ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) ) 17
EXISTS A75000 ( 1 2 3 ) ALL A55000 ( 1 2 3 )
  OCC ( A75000, A55000, QUOTE ( X ) ) 18
NOT (( ALL A55000 ( 1 2 3 ) OCC ( 1,
  A55ØØØ, QUOTE ( X ) ) ) 19
NOT ( EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) ) ) 20
OCC ( 1, 1, QUOTE ( X ) ) 21
** EXPERIME
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) )
  TRUECOUN 1TARGCOUN 1INDVALOF 17
t7: EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 18
. . . t7 & ¬t4
X X X
. . .
** IS THIS AN INSTANCE: YES
** DO YOU WANT TO TELL ME MORE: NO
** [ 21.25 16 SEPT 1969 ]
```

t7 says 'there is a row of xs'

```
** INDDEVEL
EXISTS A75000 ( 1 2 3 ) ALL A55000 ( 1 2 3 )
  OCC ( A75000, A55000, QUOTE ( X ) )
TRUECOUN 2TARGCOUN 2INDVALOF 18

** INDDEVEL
NOT ( EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) ) )
TRUECOUN 4TARGCOUN 0INDVALOF 20

** MAXNJOBS

** CHECK JOBLIST
EXISTS A75000 ( 1 2 3 ) ALL A55000 ( 1 2 3 )
  OCC ( A75000, A55000, QUOTE ( X ) ) 18
NOT ( EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) ) ) 20
OCC ( 2, 2, QUOTE ( X ) ) 21 . 0
NOT ( NOT ( EXISTS A75000 ( 1 2 3 ) ALL
  A55000 ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) ) ) ) 22
ALL A65000 ( 1 2 3 ) EXISTS A55000 ( 1 2 3 )
  OCC ( A55000, A65000, QUOTE ( X ) )
  24 . 67

** DO YOU WANT TO TELL ME MORE: YES

** TYPE BOARD

: X . .
: X . .
: X . .

** IS THIS AN INSTANCE: YES

** DO YOU WANT TO TELL ME MORE: NO

** [ 21.28 16 SEPT 1969 ]

** INDDEVEL
OCC ( 1, 1, QUOTE ( X ) )
TRUECOUN 3TARGCOUN 2INDVALOF 24 . 33

** INDDEVEL
OCC ( 2, 2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 24 . 33

** MAXNJOBS
```

Next the idea of a column is introduced, but the capability of the program of analysing disjunctive concepts is limited.

```
** CHECK JOBLIST
OCC(1, 1, QUOTE ( X ) ) 24 . 33
OCC ( 2, 2, QUOTE ( X ) ) 24 . 33
EXISTS A75000 ( 1 2 3 ) ALL A55000 ( 1 2 3 )
   OCC ( A75000, A55000, QUOTE ( X ) )
   24 . 67

NOT ( OCC ( 1, 1, QUOTE ( X ) ) ) 27 . ø
NOT ( EXISTS A75000 ( 1 2 3 ) ALL A55000
   ( 1 2 3 ) OCC ( A75000, A55000,
   QUOTE ( X ) ) ) 28 . 0

** DO YOU WANT TO TELL ME MORE: NO
```

The program now revisits earlier simple-minded proposals which were formerly not even aired.

# MACHINE LEARNING AND HEURISTIC SEARCH

# 12

# First Results on the Effect of Error in Heuristic Search

Ira Pohl
IBM Research Division
Thomas J. Watson Research Center

## INTRODUCTION

In many areas of artificial intelligence, improvement has not been evident over the early paradigms (Minsky 1963). The GPS model (Newell and Simon 1963) has not been superseded and the ideas about what heuristic search is and how to do it have remained the same over the past decade. The situation reminds me of the state of mechanical translation of natural language in the early 1960s. The ideas of a simple syntactic model and dictionary look-up were seen not to be able to bear the weight of automatic high quality machine translation (Bar-Hillel 1964). It was clear that mathematical linguistics had to be better understood. I think that heuristic search is in the same situation. (In both instances, there is increasing recognition of the importance of domain-specific semantics for solving problems.) The formal tools need to be developed to better understand and increase the power of heuristic programming. The precise characterizing of these ideas allows not only a quantitative improvement in computational performance, but through a deeper understanding can lead to a qualitative improvement from generalizing and extending these methods. [One noteworthy example of this is the current sophisticated use of $\alpha-\beta$ by Samuel's checker program (Samuel 1963) and Greenblatt's chess program (Greenblatt, Eastlake and Crocker 1967). Early researchers in game playing (Feigenbaum 1969, Newell, Shaw and Simon 1963) had used the idea without considering it significant enough to explore its ramifications. It is with this spirit and intent that this work is carried out.] The formal model of heuristic search presented here has led to a new understanding of search efficiency in problem solving.

## HISTORY

One of the important models of artificial intelligence is the directed graph model (*see* Amarel 1966a,b; Burstall 1968; Doran and Michie 1966;

Doran 1967; Hart, Nilsson and Raphael 1967; Kozdrowicki 1968; Michie 1967; Michie, Fleming and Oldfield 1968; Nilsson 1968; Pohl 1969; Sandewall 1968; Slagle and Bursky 1968). In this model a node contains a description of a possible problem state. If it is possible to get from some state $x$ to state $y$ in a single move (rule of inference, operator application, etc.) then there is a directed edge from $x$ to $y$. A single node is designated the initial or start node and another node is designated the terminal or goal node. The standard problem is to find a path between these nodes. Such a path is called a solution to the problem. Sometimes, the path is constrained to being the shortest such path in the problem space, but normally any solution path is acceptable.

The work of Amarel (1966a, 1966b), Doran and Michie (Doran 1967, Doran and Michie 1966, Michie 1967) and Hart, Nilsson and Raphael (1967) and Nilsson (1968) contributed to different aspects of formulating problems in this model. Amarel has worked principally on the representations of different problems in this model. Doran and Michie have developed a general problem solving program, called the Graph Traverser, for finding paths using heuristic functions to control the search for the goal node. Hart, Nilsson and Raphael have given sufficient conditions on heuristic functions to guarantee that a class of path finding algorithms will find the shortest solution path in the space. Each of these workers has explored important facets of using the directed graph model efficiently.

## PROBLEM SPACES AND HEURISTIC SEARCH

A directed graph $G$ is a set of nodes $X$ and edges $E$ which are ordered pairs from the node set.

$$G: \quad X=\{x_1, x_2, \ldots, x_n\}$$
$$E=\{(x_j, x_j) | x_i, x_j \in X, x_j \in \Gamma(x_i)\}$$

$\Gamma$ is the successor mapping. The size or cardinality of the graph is denoted by $|G|$ which is the number of nodes in $X$, and can be infinite. In using directed graphs to characterize problem domains we attach to each $x_i$ a data structure which contains the complete description of the problem. For example, in the 15 puzzle (see figure 1) a data structure describing it would be the vector (9,5,1,3,13,7,2,8,14,6,4,11,10,15,12,0) where b denoted

| | | | |
|---|---|---|---|
| 9 | 5 | 1 | 3 |
| 13 | 7 | 2 | 8 |
| 14 | 6 | 4 | 11 |
| 10 | 15 | 12 | b |

Figure 1. 15 puzzle

the blank position. The mapping $\Gamma$ would represent possible single moves from one problem state to another reached by sliding an adjacent tile into the blank position. In this domain we are at some specified starting node (and associated state) and wish to reach a given terminal node. We must produce a path from the initial node to the goal node. Purely exhaustive methods are impractical in complicated spaces with $|G|$ and $|E|$ large or possibly infinite. Hence in most instances we have heuristics which aid in narrowing the search. For our discussion, heuristic information is a function over state descriptions, attached to nodes, into the non-negative reals. Intuitively we want this function to estimate the distance a node is from the goal node.

## AN ALGORITHM FOR HEURISTIC SEARCH

When solving most artificial intelligence problems, we are not ordinarily interested in the most 'elegant' or shortest path, but in how to obtain any path cheaply. A search method visits a number of nodes in $G$ to find a path. We want this number to be as few as possible, so that it may be computationally feasible to find solution paths which are inherently long; i.e., the shortest path is long.

HPA – Heuristic Path Algorithm

$s$ = start node, $t$ = terminal node

$g(x)$ = the number of edges from $s$ to $x$, as found in our search

$h(x)$ = an estimate of the number of edges from $x$ to $t$, our heuristic function

$f(x) = (1-\omega)g(x) + \omega \cdot h(x) \qquad 0 \leqslant \omega < 1$

$S$ = set of nodes already visited. Also called the expanded nodes

$\tilde{S}$ = set of nodes directly reachable (in one edge) from $S$, but not in $S$. Also called the candidate nodes.

1. Place $s$ in $S$ and calculate $\Gamma(s)$ placing them in $\tilde{S}$.
   If $x \in \Gamma(s)$ then $g(x) = 1$ and
   $f(x) = (1-\omega) + \omega \cdot h(x)$.
2. Select $n \in \tilde{S}$ such that $f(n)$ is a minimum.
3. Place $n$ in $S$ and $\Gamma(n)$ in $\tilde{S}$ (if not already in $\tilde{S}$) and calculate $f$ for the successors of $n$.
   If $x \in \Gamma(n) \wedge x \notin S$ then $g(x) = 1 + g(n)$
   and $f(x) = (1-\omega) \cdot g(x) + \omega \cdot h(x)$
4. If $n$ is the goal state then halt, otherwise go to step 2.

*Note.* HPA builds a tree; as each node is reached a pointer to its predecessor is maintained. Upon termination the solution path is traced back from the goal node through each predecessor.

HPA is a typical path finding algorithm and is similar to the algorithms used in the work of Doran and Michie (1966) and Hart, Nilsson and Raphael (1967). The key difference is the evaluation function. HPA operates in a

space with an edge cardinality metric and uses a linear combination of some estimator (heuristic) and the $g$ term to guide the search. The Graph Traverser* relies completely on $h$ and the Hart, Nilsson and Raphael algorithm† uses $g+h$. A minor point is that the goal state is detected only after a node is placed in set $S$ rather than in $\tilde{S}$. This is a theoretical convenience. If the cost of testing for the goal state is negligible, the test would occur when nodes are placed in set $\tilde{S}$; unless one wanted to guarantee that the path is shortest.

## AN EXAMPLE OF THE USE OF HPA

The 15 puzzle is a simple, but combinatorially large problem space. Each space‡ contains $\frac{16!}{2}$ configurations, too large to be searched exhaustively. The average degree (number of moves) of a node is 3, allowing exhaustive search to find solutions of about 10 steps. On the other hand, the space is simple enough to study as an heuristic search problem and heuristic functions are easy to formulate.

The standard problem is – given some initial configuration in the appropriate parity space, how can we push the tiles around to reach the standard goal configuration? In figure 2, we see a possible initial configuration and its state description as given by a 16-tuple. From the initial configuration, there are two successor states possible. These correspond to switching any tile adjacent to the blank into the blank's position. The position of the blank in the center of the board allows four possible moves, on the sides three possible moves and in the corners two possible moves.

In applying HPA to our problem, we ordinarily attempt to find a good heuristic function $h$. If, for example, we chose $h=0$, then we have a breadth first search (Moore 1959) which will ordinarily require too much time and space. Now one simple heuristic measure is a position count. We have a 16-tuple of tile values which are out of order. Any particular tile is so many squares away from its position in the goal configuration.

We can say, as in the Graph Traverser work, that

$p_i$=the Manhattan distance of the tile in position $i$ from its goal position.

$$P=\sum_{n=1}^{16} p_i$$

* The Graph Traverser search could be transformed to an HPA search by using $h^*=(1-\omega)g+\omega h$.

† In the Hart, Nilsson and Raphael case the graph has edge lengths which are positive reals and $h$ is restricted to be bounded above by the actual distance. Then they can demonstrate that the method finds the shortest path.

‡ A particular 15 puzzle configuration may be in one of two spaces. A configuration in one space cannot be manipulated by any sequence of moves into a configuration of the other space.

Note, if $P=0$ then we are finished; also $P$ represents a lower bound on how many moves remain to the goal. In figure 2b, we show the initial configuration with its two successor states and their position count. The first

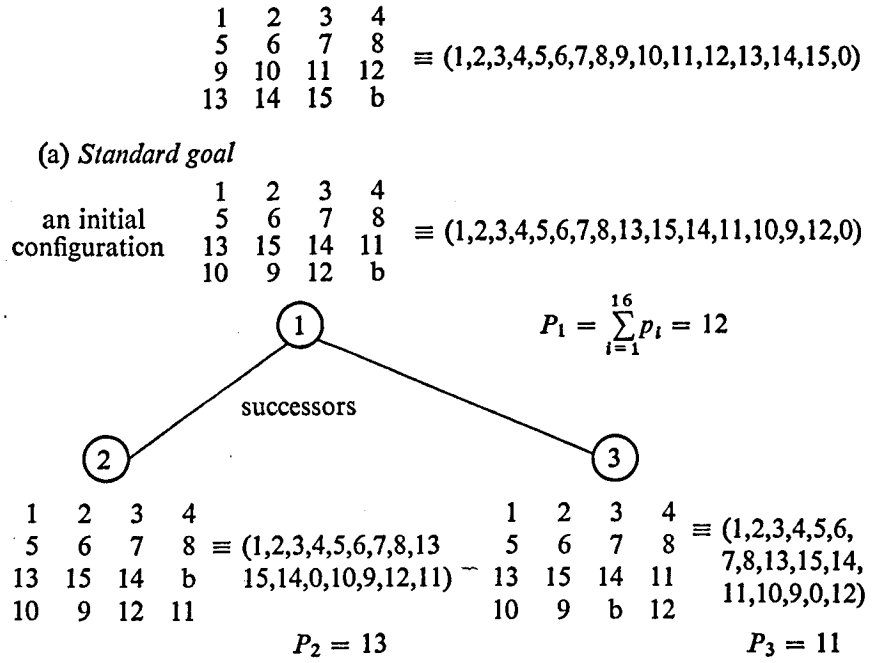



Figure 2. Some puzzle configurations and descriptions

step of HPA would place nodes 2 and 3 (numbers in circles) in set $\bar{S}$ and then node 3 would be placed in set $S$ because it has smaller value. This process continues until the goal state is reached or the computational resources allotted to the problem are exhausted.

| | | |
|---|---|---|
| $\omega=0$, | $f(n)=g(n)$ | exhaustive parallel on breadth first search |
| $\omega=1$, | $f(n)=h(n)$ | simple or pure heuristic search – Graph Traverser search |
| $0<\omega<1$ | $f(n)=(1-\omega)g(n)+\omega h(n)$ | compound heuristic search |

Table 1. Commonly used evaluators

The nature of efficient heuristic search is clearly visible in this type of problem environment, and leads to questions of how to use appropriately the heuristic information. HPA will find a path if one exists and the graph is finite, but can fail if the graph is infinite. Our criterion of search efficiency is how many nodes are expanded before the goal node is found. This is the size of $S$ upon termination of HPA. In Table 1, we list some common weights for $\omega$, and the type of search produced. The extreme of exhaustive parallel search, where no heuristic information is used, is obviously inefficient as it must expand all nodes up to the depth of the solution node. Therefore, it is natural to look at the other extreme, pure heuristic search, which would hopefully give a very narrow search. The intuitive reason for this weighting is that prior distance in reaching a node is so much 'water over the dam'. Indeed, if $h$ is an accurate estimator of distance from the goal, it will indicate the node nearest the goal. This remaining distance is what determines the fewest nodes to visit. This argument is plausible, but relies on the accuracy of the heuristic function. Only domains with inaccurate estimators are interesting, and it is these cases for which efficient use of heuristic information is necessary.

## SOME THEOREMS ON SEARCHING

In examining formally the claims of the above argument two extremes are easily dealt with. First, we could have a heuristic function which always returned the exact distance to the goal, a function having this property we call *perfect*. Secondly, we could have a heuristic function which is completely in error; this would be the reciprocal of the perfect function.

*Theorem 1*

If $h$ is perfect then for $1 \geqslant \omega \geqslant \frac{1}{2}$, the search by HPA is optimal, i.e., includes the fewest nodes in set $S$ in finding a solution path.

*Proof.* Let the shortest path be $k$ steps long.

$\mu = (s, x_1, x_2, \ldots, x_{k-1}, t)$. Since $h$ is perfect then $h(x_i) = k - i$. Now we will show by induction that only nodes along $\mu$ will be placed in set $S$. The induction is with respect to the number of elements of set $S$, which is the same as the number of iterations of HPA.

*Case* 1. Initially $s$ is in $S$ and $\Gamma(s)$ is in $\tilde{S}$.

Since $x_1 \in \Gamma(s)$, we have for $x_1$

$f(x_1) = (1-\omega) + (k-1)\omega$ which we claim is smaller than $f(y)$,
$y \in \Gamma(s)$ and $y \neq x_1$.
$f(y) = (1-\omega) + \omega h(y)$
$y$ is off the shortest path and so $h(y) \geqslant k$
$f(y) \geqslant (1-\omega) + \omega h(y) > f(x_1)$.

So $x_1$ would be chosen on the first iteration of HPA.

*Case* 2. Assume that on the $i$th iteration $x_i$ is placed in set $S$. We must show that $x_{i+1}$ would be placed in set $S$ on $(i+1)$th iteration.

Since $x_i$ was placed in $S$, $x_{i+1}$ must be in $\bar{S}$.

$$f(x_{i+1}) = (1-\omega)g(x_{i+1}) + \omega h(x_{i+1})$$
$$g(x_{i+1}) = i+1,\ h(x_{i+1}) = k-i-1$$

First, we show $f(x_{i+1}) \leqslant f(x_i)$.

$$(1-\omega)g(x_{i+1}) + \omega h(x_{i+1}) \leqslant (1-\omega)g(x_i) + \omega h(x_i)$$
$$(1-\omega)[g(x_i)+1] + \omega(k-i-1) \leqslant (1-\omega)g(x_i) + \omega(k-i)$$
$$1-2\omega \leqslant 0 \text{ true for } \tfrac{1}{2} \leqslant \omega \leqslant 1$$

Of course this is not true for $\omega < \frac{1}{2}$ and hence these bounds in the theorem.

Now since $x_i$ had been chosen on the previous iteration, it was better than the nodes in $\bar{S}$ at that time. Therefore $x_{i+1}$ must also be by the above inequality. It only remains to check if $x_{i+1}$ is better than other nodes in $\Gamma(x_i)$.

Other nodes $y$ will have value

$$f(y) = (1-\omega)g(y) + \omega h(y)$$
$$g(y) = 1 + g(x_i) = g(x_{i+1}) \text{ and } h(y) > h(x_{i+1})$$

$\therefore f(y) > f(x_{i+1})$ and $x_{i+1}$ will be selected on the $(i+1)$th iteration. This process must continue until $t$ is found by only including nodes in $S$ along the shortest path.

We see that for $h$ perfect and for $\frac{1}{2} \leqslant \omega \leqslant 1$ HPA only expands nodes on the shortest path. If $\omega < \frac{1}{2}$ the proof does not carry through and additional nodes may be expanded, with $\omega = 0$ the worst case. However, the key point is that using $g(x)$ in our evaluator does not decrease the efficiency of search, when appropriately weighted; even if the heuristic is the best obtainable. This already in some measure refutes the 'common sense' arguments for pure heuristic search being the most efficient.

*Theorem 2*

If $h$ is the reciprocal of the perfect heuristic function, then the search by HPA using pure heuristic search ($\omega = 1$) will always visit the goal node last. If the space is infinite, the goal will never be found. For this heuristic function $\omega = 0$ gives the best search.

*Proof.* Since we are using the reciprocal of the perfect function, the further from the goal node the smaller $h$. So HPA with $\omega = 1$ will be led away from the goal, and only if it exhausts the rest of the space will it reach the goal. It is obvious that using $\omega = 0$ is the best search possible with this ill-conceived heuristic.

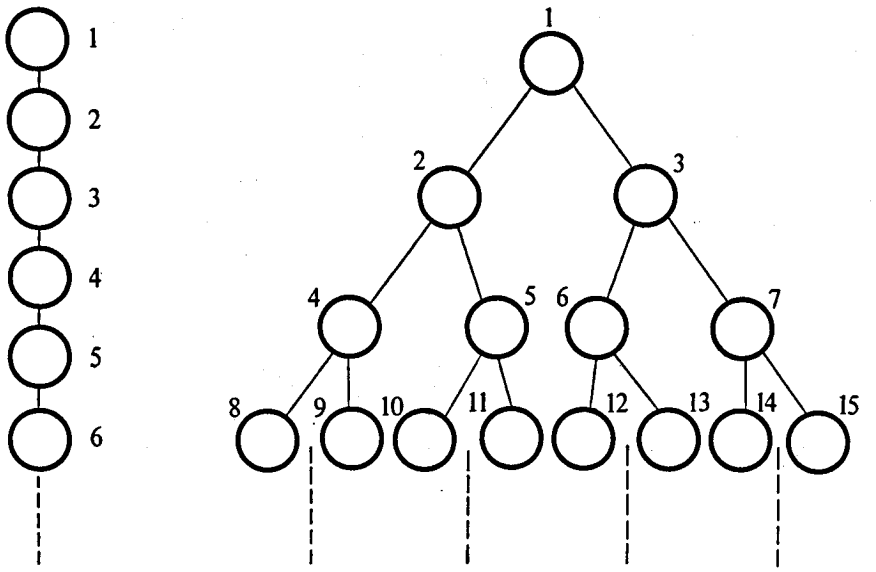
So in the case where the heuristic function is counterproductive, the less we rely on it the better. It now remains to investigate cases where $h$ is somewhere between these extremes in its accuracy.

*Heuristic error*

To do this rigorously will require easily analyzable spaces. The spaces used will be regular infinite rooted trees. A regular tree is a tree in which all the nodes

have the same degree, except the root node which is one less and terminal nodes which have degree one. By an infinite regular tree, we mean a tree without terminal nodes.

The simplest such space is the unary tree (figure 3a). Over this space all functions, representing any heuristic function and weighting, are equivalent. The search always proceeds from node 1 to node 2, . . ., until the goal node is encountered. This case is without interest and we move on to the binary tree space (figure 3b). This is already non-trivial and complex enough to represent reasonable problem domains such as LISP programs (McCarthy 1964).




a (unary) b (binary)

Figure 3. Regular infinite trees, (a) unary, (b) binary

Theorems 1 and 2 apply regardless of the specific graph structure, thus the use of a perfect $h$ in our evaluator $f$ is optimal for $\frac{1}{2} \leqslant \omega \leqslant 1$. Perhaps the intermediate situation exists of knowing no heuristic information in our domain. We have $h$ identically 0 throughout the binary tree space. The evaluators we could use are then:

(a) $f=(1-\omega)g+\omega h=(1-\omega)g$ $\quad 0 \leqslant \omega < 1$

(b) $f=h=0$ $\quad \omega = 1.$

The use of $(1-\omega)g$ for $\omega < 1$ is identical to using $g$ and constitutes a parallel search. The use of 0 is a search where all the nodes in $\tilde{S}$ will be tied. If at each iteration step 2 of HPA randomly chooses from ties, then (b) produces a random search. If instead our tie-break rule was first-in/first-out we would have parallel search. Last-in/first-out would be a depth first rule. These simple distinctions in tie breaking promote vastly different search regimes!

*Theorem 3*

Over an infinite binary tree, a parallel search requires on the average $2^k+2^{k-1}-\frac{1}{2}$ nodes expanded to find a node $k$ steps from the root.

*Proof.* The number of nodes in a full binary tree of depth $k$ is $B_k=2^{k+1}-1$. A full tree has at each level its maximum possible number of nodes, e.g., $2^k$ is that number at level $k$ of a binary tree. Since a node may be anywhere along the $k$th level with equal probability, we must search $B_{k-1}+1$ to $B_k$ nodes with the average being

$$\tfrac{1}{2}(B_{k-1}+1+B_k)=2^k+2^{k-1}-\tfrac{1}{2}.$$

So in parallel search of a binary tree, we have the above formula determining, on average, how many nodes must be visited. It is exponentially varying with the distance from the root; typical behaviour in complex problem spaces. In contrast, let us examine the expected number of nodes using random search (b) in finding a goal node $k$ steps from the root. This would be comparing total reliance on the heuristic (theorem 3) with compound heuristic search for the case that the heuristic is useless. In the simplest non-trivial case $k$ equals 1 ($k=0$ is trivial).

*Theorem 4*

Over an infinite binary tree, a random search expects to visit an unbounded number of nodes to find a goal node 1 step from the root.

*Proof.* $E$=Expected number of nodes visited

$r_i$=Probability of finding the goal node in exactly $i$ steps

$p_i$=Probability of finding the goal node on the $i$th step having reached this step

$l_i$=Probability of not finding the goal node on any step before the $i$th step

$E=1+\sum_{i=1}^{\infty} i\cdot r_i$, the 1 is for the root node

$r_i=p_i\cdot l_i$

With each step of HPA over a binary tree one candidate node is expanded and two new candidates are placed in $\tilde{S}$. At step $i$ of HPA there are $i+1$ nodes in $\tilde{S}$ all with equal $f=0$. So $p_i=\dfrac{1}{i+1}$.

$l_i$ can be shown by induction to be $\dfrac{1}{i}$.

$$\therefore E=1+\sum_{i=1}^{\infty} i\cdot r_i=1+\sum_{i=1}^{\infty} i\cdot p_i\cdot l_i=1+\sum_{i=2}^{\infty}\frac{1}{i}$$

$$=\sum_{i=1}^{\infty}\frac{1}{i}$$ the harmonic series which is unbounded.

This result is similar to gambler's ruin problems (Feller 1950). Essentially the space grows too fast, and when not lucky enough to find the goal node, initially we soon find it disappearing in the growth of $\tilde{S}$.

Normally, a search is restricted by time or space limitations. This is akin to limiting our infinite space to some maximum depth. If we are interested in finding a goal node in a binary tree of diameter $k$, then a maximum of $2^{k+1}-1$ nodes need be searched. If each of these nodes is with equal probability the goal node, then any exhaustive non-repeating search would yield the same expected value for nodes visited $\frac{1}{2}(B_k+B_0)$, or $2^k$. Each method would get better performance for different groups of nodes. The parallel search visits the closer nodes soonest, and for goal nodes near the root this method has a better expected value than random or depth first search.

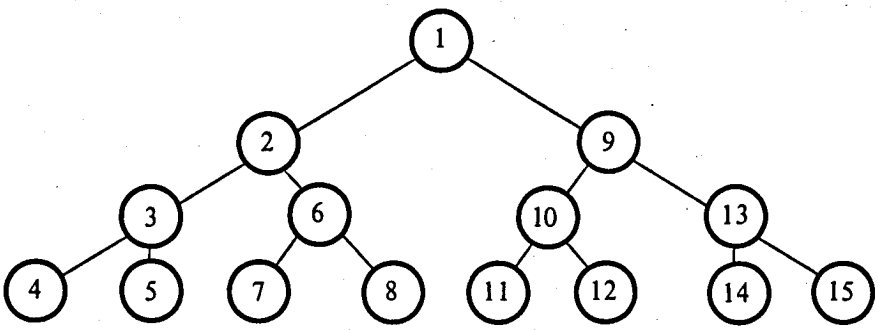



Figure 4. Nodes numbered in order visited by a depth first search to level 3

In our theorem 4, HPA was unaware that the goal was at level 1, and so with finite probability it searched portions of the space which were beyond the solution. A modification on this would be to tell our procedure that the goal was on level $k$. When this is known, the depth first (see figure 4) or backtrack method (Floyd 1967, Golomb and Baumert 1965) is optimal. The algorithm should go down to a depth of $k$ and check to see if this is the goal node. If not it backs up one level and goes down to the next node on level $k$ until it finds the goal. Since this search pattern looks at nodes on the $k$th level as soon as possible, it must be best in the sense of the expected number of nodes visited. It would be the worst search pattern if the goal node was actually at level 1. Here it either finds the goal on the first try (like any other method) or must look at half the tree before returning to the goal node. A parallel search is a conservative strategy, you are guaranteed not to penetrate below the part of the tree containing the goal, while you pay by always investigating the whole subtree up to that level.

## BOUNDED ERROR AND ITS EFFECTS ON SEARCH

In general we have neither a complete lack of information nor perfectly accurate information. Instead, we have a useful estimator which has error.

We wish to resolve for this more typical instance how best to use a heuristic function and the effect of error on search efficiency. To investigate this question, we stay in our binary tree space using HPA. We will do a worst case analysis in the spirit of error analysis in numerical problems.

Consider

$h^*$ = perfect estimator

$\varepsilon$ = a bound on the error 0, 1, 2, ...

$h$ = given heuristic function $(h^*-\varepsilon) \leqslant h \leqslant (h^*+\varepsilon)$ in our problem domain.

We will construct an $h$ satisfying the above limits, but in such manner as to mislead HPA to the greatest extent. In doing this, we assume that HPA will always choose the worst nodes in case of ties, i.e., nodes off the solution path. An example of this analysis is figure 5, where HPA just uses $h$ as the evaluator, $\omega = 1$. The order of search is according to the numbers inside the nodes with $x$, the goal being reached in 5 steps. To make $h$ as bad as possible ($\varepsilon = 1$), we have

$h^* = h+\varepsilon$ for each node on the solution path, and

$h^* = h-\varepsilon$ for each node off the solution path.

If $h^*$ itself was used HPA would only visit the 3 nodes on the shortest path, which is known from theorem 1.

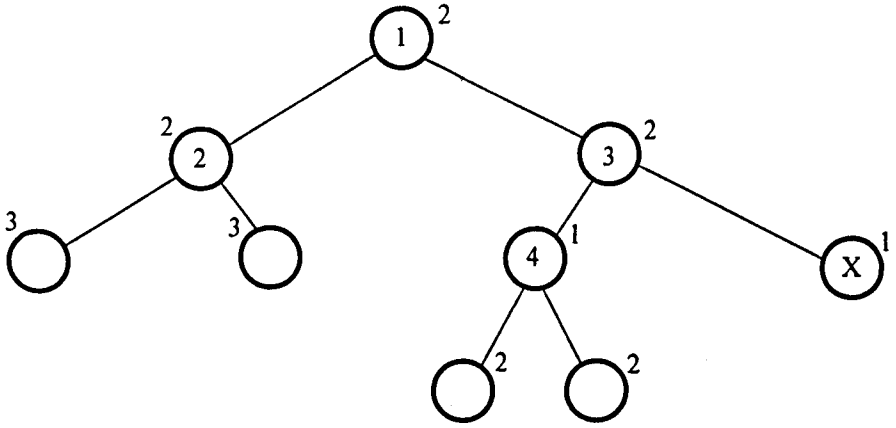
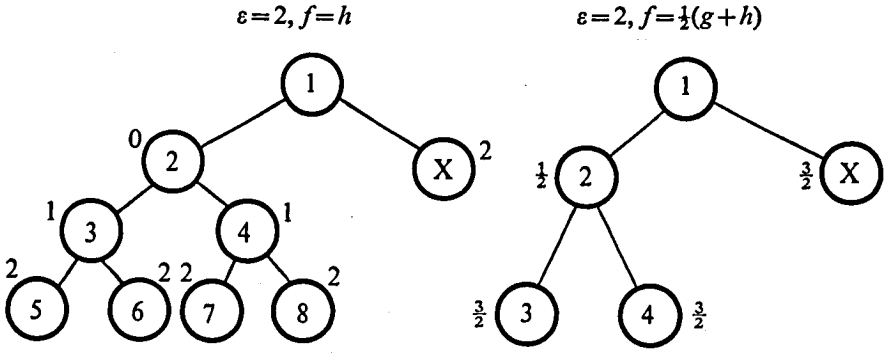



Figure 5. The goal node is marked by an $X$. Other nodes are labelled by order of search (inside) and $f$ value outside. Five nodes are searched when $X$ is found.

One of the principal questions is the comparison between using $\omega = 1 (f = h)$ and $\omega = \frac{1}{2} (f = \frac{1}{2}(g+h))$ as evaluators. These are the end points for the range in which theorem 1 holds. In between values will have in between behavior with respect to error, and smaller values will demonstrate too much breadth first behavior. Both to get more of a flavor of our error analysis and some




(a) 9 nodes visited (b) 5 nodes visited

Figure 6. Comparison between two possible ωs for $f=(1-\omega)g+\omega h$. (a) $\omega=1$, (b) $\omega=\frac{1}{2}$.

inklings as to this comparison, we work through the example of figure 6. Let us examine HPA using $f=\frac{1}{2}(g+h)$ and $\varepsilon=2$, as in figure 6b. At the goal $x$,

$$h^*(x)=0,\ g(x)=1$$
$$h(x)=h^*(x)+\varepsilon=0+2=2$$
$$f(x)=\tfrac{3}{2};$$

while at node 2 we have

$$h^*(2)=2,\ g(2)=1,\ h(2)=h^*(2)-\varepsilon=2-2=0$$
$$f(2)=\tfrac{1}{2}.$$

Node 3 has

$h^*(3)=3$, $g(3)=2$ and so both have increased by 1 from values of its predecessor node 2. Thus

$h(3)=3-2=1, f(3)=\frac{3}{2}$ an increase of 1 from its predecessor.

The fact that both $g$ and $h$ increase along an incorrect path allows the search along such paths to be discontinued sooner. The results of the searches presented in figure 6 are:

distance to goal = 1
maximum error $\varepsilon=2$
nodes visited $f=h$ are 9
nodes visited $f=\frac{1}{2}(g+h)$ are 5.

We can generalize this result and find a formula giving the number of nodes visited for different errors and path lengths in our binary tree space.

In analyzing the worst behavior, we must show that $h=h^*+\varepsilon$ on the solution path and $h=h^*-\varepsilon$ off the solution path leads to the poorest searches. This is intuitively clear when one keeps in mind that $h$ is a distance estimator. By adding a maximum error along the solution path we make the goal node appear farther away. Likewise by subtracting $\varepsilon$ off the solution path we make these appear closer to the goal node.

*Theorem 5*

If $h_1 = h_2$ except on the solution path where

$$h_1(x) \geqslant h_2(x),\ x \text{ on solution path},$$

then the search by HPA using $h_1$ always visits all the nodes visited when using $h_2$. This is proved with respect to tree spaces, but the only necessary condition is that a unique solution path exist.

*Proof.* Let $u = (x_1, x_2, \ldots, x_k)$, $x_1 = s$, $x_k = t$ be the solution path. $S(x_i)$ will be the nodes in set $S$, up to and including the iteration of HPA when $x_i$ is placed in $S$. So $S(x_k)$ is the set of nodes searched when HPA finds the solution path. Let $S_1(x_i)$ be the sets associated with HPA using $h_1$ and $S_2(x_i)$ be the corresponding sets for $h_2$. We show by induction

$$S_1(x_i) \supseteq S_2(x_i).$$

(a) $S_1(x_1) \supseteq S_2(x_1)$ since $S_1(x_1) = S_2(x_1) = \{s\}$

$S_1(x_2) \supseteq S_2(x_2)$ since $h_1(x_2) \geqslant h_2(x_2)$

and all values off the solution path are the same. On the iteration before $x_2$ is placed in $S$, HPA must have behaved the same for $h_1$ and $h_2$. On the next iteration $h_1$ may not include $x_2$, but some other node.

(b) More generally $S_1(x_i) \supseteq S_2(x_i)$ and

therefore $S_1(x_k) \supseteq S_2(x_k)$.

Consider $y \in S_2(x_i)$, $y \notin S_1(x_i)$ and moreover this is the first time this happens, namely $S_1(x_{i-1}) \supseteq S_2(x_{i-1})$. $f_1(y) = f_2(y)$ since it is off the solution path. Also, we know $f_1(x_i) \geqslant f_2(x_i)$ and so the only reason $y$ would not have been included in $S_1(x_i)$ is that it was not reached by HPA. But, the node it was expanded from was reached and included in both $S_1(x_i)$ and $S_2(x_i)$ or else the condition on $y$ being the first such node is violated. However this contradicts the fact that it was not available for expansion by HPA using $f_1$. If it cannot happen a first time then we are finished.

*Theorem 6*

If $h_1 = h_2$ on the solution path, but everywhere else $h_1(x) \leqslant h_2(x)$, $x$ off the solution path, then the search by HPA using $h_1$ always visits all the nodes visited when using $h_2$.

*Proof.* The argument is similar to theorem 5 and will not be described. Now the reason more nodes may be visited using $h_1$ is that values off the solution path are lower using $h_1$ and hence sooner included in set $S$.

Taken together the above theorems show that a worse case search occurs when $h + \varepsilon$ is used on the solution path and $h - \varepsilon$ is used off the solution path.

*Theorem 7*

Let $k$ be the distance from the root node to the goal node and $f = \frac{1}{2}(g + h)$ be the function used by HPA, then the maximum number of nodes expanded in our binary tree space is

$$2^{\varepsilon} \cdot k + 1$$

where $\varepsilon$ is the error bound on $h$.

*Proof.* If the goal node is distance $k$ from the root, then if $h$ is perfect HPA visits $k+1$ nodes (theorem 1). In the worst case with an error of $\varepsilon$, all nodes $\varepsilon$ off the shortest path will be visited, excluding the nodes succeeding the goal node. This is shown in our discussion of figure 6.

Case $\varepsilon=0$

This is as stated above $k+1$.

Case $\varepsilon=1$

These are the nodes on the shortest path plus those one off the shortest path. There are $k$ nodes one off the shortest path so we have

$$k+k+1=2k+1.$$

Case $\varepsilon\geqslant 2$

At this point each candidate node ($\tilde{S}$) has two not yet explored successors. The number of these in a binary tree grows as $2^{\varepsilon-1}\cdot k$. So the tree for error $\varepsilon\geqslant 2$ is size

$$2k+1+2\cdot k+2^2k+\ldots+2^{\varepsilon-1}k=1+2k+k\sum_{i=1}^{\varepsilon-1}2^i$$
$$=1+2^\varepsilon\cdot k\,.$$

Similarly we prove

*Theorem 8*

Let $k$ be the distance from the root node to the goal node and $f=h$ be the function used by HPA, then the maximum number of nodes visited in our binary tree space is

$$k+1 \quad \varepsilon=0,$$
$$\frac{4^\varepsilon}{2}\cdot k+1 \quad \varepsilon\geq 1$$

Case $\varepsilon=0$

Again by theorem 1 the answer is $k+1$.

Case $\varepsilon=1$

Here as in the previous theorem HPA visits nodes one off the solution path and we have

$$2k+1=\frac{4^1}{2}k+1 \text{ nodes visited.}$$

Case $\varepsilon\geqslant 2$

After the first level each increment in the error allows a maximum of two additional levels to be visited. This is because along the solution path we use $h+\varepsilon$ and off the solution path we use $h-\varepsilon$ giving a $2\varepsilon$ leeway. The trees with the maximum number of explored nodes are size

$$\underbrace{2k+1}_{\varepsilon=1}+\underbrace{2k+2^2k}_{\varepsilon=2}+\ldots+\underbrace{2^{2\varepsilon-3}k+2^{2\varepsilon-2}k}_{\varepsilon}$$

$$=1+2k+k\sum_{i=1}^{2\varepsilon-2}2^i=1+\frac{4^\varepsilon}{2}k$$

These results are extendable to any tree structured problem space with a unique goal node. Namely

*Theorem 9*

If HPA is searching any tree structured space for some goal node then
(a) $f=h$ will visit at least as many nodes as $f=\frac{1}{2}(g+h)$ in the sense of the above worst case analysis.
(*b*) If $h_1$ is bounded in error by $\varepsilon_1$ and $h_2$ by $\varepsilon_2$ with $\varepsilon_2>\varepsilon_1$, then in the sense of the worst case analysis $h_2$ can only search more nodes than $h_1$ for the same $\omega$ value.

*Proof.* Part (b) is obvious and follows from theorems 4 and 5. Part (a) comes from the same argument as was used in theorems 7 and 8. However with the spaces being general trees, we do not have the nice binary property allowing us to give simple counting formulas.

In our easily analyzable domains and our notion of worst case search, we have arrived at some concrete results.

1. The more accurate $h$, the fewer nodes expanded by HPA.
2. It can be better and is never worse to use a weighted evaluator.

Conclusion 1 is what we expect and is empirically seen in the work of the Graph Traverser and some experiments of my own. It is also compatible with the Hart, Nilsson and Raphael (1967) theorem on efficiency of search for 'informed' shortest path algorithms. Another point to note is for $\omega=1$, it does not change the search* to use a positive constant multiplying $h$ in place of $h$, but it may change the worst case analysis; it in fact is really a problem of scaling. This problem also arises when one wants to combine different heuristic terms.

Conclusion 2 is at first sight surprising. Theorem 1 shows why taking $g$ into account will not hurt you, while theorems 7 through 9 show that it has a beneficial effect on retarding misleading searches. Nevertheless, the above analysis of how distance estimators work as heuristics is limited in generality. Most problem spaces are not trees, but are problem domains containing circuits offering many alternate solution paths. Also the above analysis is for the worst case, and while these results are attainable, in practice they are unlikely and a statistical theory would be more accurate. Thus it is important to monitor the behavior of HPA in actual problems.

* Any two evaluation functions giving the same ordering on the node set will produce the same searches.

## SOME EXPERIMENTAL FINDINGS

Each problem space and each heuristic function for this space presents a problem in selecting an appropriate $\omega$. Exclusive use of $g$ guarantees finding the shortest solution path. However, a price is paid in the breadth of the search. On the other hand, using only $h$ is possibly unstable; HPA then may run down the search space to great depths before abandoning a worthless search in one vicinity to try another part of the space. This behavior is analogous to the situation described by theorem 4. It is appropriate to look for a middle ground between the pitfalls of these extremes.

Let us consider a specific heuristic function, $h$, used by HPA in some problem space. We characterize its effectiveness for a given problem by the number of nodes $N$ it visits in finding a solution path of length $K$. A further derived measure is the branching rate of the search. (This was suggested by Nils Nilsson.) The branch rate $\rho$ is the number of successors a node has in a regular tree of diameter $K$ and size $N$. So given a particular heuristic function and an evaluator using some value of $\omega$, we solve a number of sample problems in our space, obtaining for each solution

$N_\omega$ = number of nodes searched for this $\omega$.

$K_\omega$ = length of solution path.

We determine

$\rho_\omega$ = branch rate

by solving for $\rho_\omega$ in

$$N_\omega = \frac{\rho_\omega}{\rho_\omega - 1}(\rho_\omega^{k_\omega} - 1).$$

We then use this information to pick $\omega$ such that it solves problems with the smallest $N_\omega$. Intuitively $\omega$ may be thought of as a measure of confidence we place in the heuristic function (when $h$ is appropriately scaled with respect to $g$). The $g$ part of the evaluator is monitoring the performance of the $h$

| $\omega$ | $k_\omega$ | $\rho_\omega$ | $N$ |
|---|---|---|---|
| 0.33 | 38 | 1.129 | 868 |
| 0.43 | 38 | 1.083 | 258 |
| 0.50 | 46 | 1.054 | 203 |
| 0.60 | 46 | 1.029 | 98 |
| 0.66 | 76 | 1.023 | 202 |
| 0.75 | 76 | 1.018 | 163 |
| 0.80 | 76 | 1.017 | 157 |
| 0.94 | 76 | 1.016 | 147 |
| 1.00 | 78 | 1.016 | 155 |

Table 2. Density against path length for function $f_4$, Problem A5.

part; because for a particular search path to be pursued $h$ must decrease commensurate with the increase in $g$.

The 15 puzzle provides a domain with circuits and some known heuristic functions, such as the Manhattan heuristic $P$ described earlier. The tests were not extensive, but provided preliminary confirmation of the usefulness of our model (Pohl 1969). Table 2 is an example of the statistics gathered for a particular problem and heuristic function.

In general as $\omega$ increased, path length $K_\omega$ increased and branching rate decreased. To squeeze the maximum performance out of a heuristic, this model should be used to try to learn $\omega$ for a function and domain. A further point was discovered. If a heuristic is of positive benefit (unlike the one used in theorem 2) it is better to have values of $\omega$ that make the heuristic an overestimate of remaining distance rather than an underestimate. In other words, the consequences of broadening the search are worse than deepening it.

## CONCLUDING REMARKS

Heuristic search is understandably a difficult process to treat formally. Its essence is to use an understanding of a particular problem domain. Different domains give rise to completely disparate collections of tricks. Nevertheless, an improved understanding of a wide class of these methods viewed as distance estimators has been attempted. The results on using the distance from the start in conjunction with the heuristic term runs counter to the widespread practice of strict reliance on the heuristic term. The improved efficiency of the compound evaluator has been demonstrated in a theoretical sense and to some extent in empirical work.

### Acknowledgements

The work presented here was developed as part of my dissertation under Professor William F. Miller of Stanford University and the Stanford Linear Accelerator Center. His guidance and encouragement throughout the course of this research were invaluable. In addition many friends and colleagues provided stimulating discussion and encouragement. Especially helpful were the remarks and criticism of Dr Nils Nilsson and Professor Donald Michie.

The work was supported by the US Atomic Energy Commission under Contract No. AT(04–3)–515.

This paper was written, in part, while the author was a Science Research Council Visiting Research Fellow in the Experimental Programming Unit, Department of Machine Intelligence and Perception, University of Edinburgh.

## REFERENCES

Amarel, S. (1966a) An approach to heuristic problem solving and theorem proving in the propositional calculus. Pittsburgh: Carnegie Institute of Technology.

Amarel, S. (1966b) On machine representations of problems of reasoning about actions – the missionaries and cannibals problem. Pittsburgh: Carnegie Institute of Technology. Also in *Machine Intelligence 3*, pp. 131–71 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Bar-Hillel, Y. (ed.) (1964) *Language and Information*, Palo Alto, California: Addison-Wesley, especially 'Nonfeasibility of FAHQT', pp. 174–9.

Burstall, R. M. (1968) Writing search algorithms in functional form. *Machine Intelligence 3*, pp. 373–85 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Doran, J.E. & Michie, D. (1966) Experiments with the Graph Traverser program. *Proc. R. Soc. A*, **294**, 235–59.

Doran, J.E. (1967) An approach to automatic problem-solving. *Machine Intelligence 1*, pp. 105–35 (eds. Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Feigenbaum, E. A. (in press) Artificial intelligence: themes in the second decade. *Preceedings of the IFIP 68 International Congress, Edinburgh, August 1968.*

Feller, W. (1950) *An introduction to probability theory and its applications 1.* London: John Wiley and Sons.

Floyd, R. (1967) Nondeterministic algorithms. *J. Ass. comput. Mach.*, **14**, 636–44.

Golomb, S. & Baumert, L. (1965) Backtrack programming. *J. Ass. comput. Mach.*, **12**, 516–24.

Greenblatt, R.E., Eastlake, D.E. & Crocker, S. (1967) The Greenblatt chess program. *Proceedings Fall Joint Computer Conference*, pp. 801–10.

Hart, P., Nilsson, N. & Raphael, B. (1967) A formal basis for the heuristic determination of minimum cost paths. *Stanford Research Institute report.* Also *IEE Trans. on Sys. Sci. and Cybernetics* (July 1968).

Kozdrowicki, E. (1968) An adaptive tree pruning system: A language for programming heuristic tree searches. *Proceedings of the ACM*, pp. 725–35.

McCarthy, J. (1964) *LISP 1.5 Programmer's Manual.* Cambridge, Mass: MIT Press.

Michie, D. (1967) Strategy building with the Graph Traverser. *Machine Intelligence 1*, pp. 135–52 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Michie, D., Fleming, J.G & Oldfield, J.V. (1968) A comparison of heuristic, interactive and unaided methods of solving a shortest-route problem. *Machine Intelligence 3*, pp. 245–55 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Minsky, M. (1963) Steps toward artificial intelligence. *Computers and Thought*, pp. 406–50 (eds. Feigenbaum, E. & Feldman, J.). New York: McGraw-Hill Company.

Moore, E. (1959) The shortest path through a maze. *Proceedings of an International Symposium on the theory of switching, Part II, April 1957*, pp. 285–92. Cambridge, Mass.: Harvard University Press.

Newell, A. & Simon, H. (1963) GPS, a program that simulates human thought. *Computers and Thought*, pp. 279–93 (eds Feigenbaum, E. & Feldman, J.). New York: McGraw-Hill Company.

Newell, A., Shaw, J.C. & Simon, H. (1963) Chess-playing programs and the problem of complexity, *Computers and Thought*, pp. 39–70 (eds. Fegenbaum, E. & Feldman, J.). New York: McGraw-Hill Company.

Nilsson, N. (1968) Searching problem-solving and game playing trees for minimal cost solutions. *IFIPS Congress preprints*, pp. H125–30.

Pohl, I. (1969) Bi-directional and heuristic search in path problems. Ph.D. thesis, Stanford University and *Report No. 104.* Stanford Linear Accelerator Center. Stanford University, Stanford, California.

Samuel, A. L. (1963) Some studies in machine learning using the game of checkers. *Computers and Thought*, pp. 71–105 (eds Feigenbaum, E. & Feldman, J.). New York: McGraw-Hill Company.

Sandewall, E. (1968) A planning problem solver based on look-ahead in stochastic game trees. *Report No. NR-13.* Department of Computer Science, Uppsala University, Uppsala, Sweden. Also in *J. Ass. comput. Mach.*, **16**, 364–82.

Slagle, J. & Bursky, P. (1968) Experiments with a multipurpose theorem-proving heuristic program. *J. Ass. comput. Mach.*, **15**, 85–99.

# 13

# A Set-Oriented Property-Structure Representation for Binary Relations SPB

Erik Sandewall
Computer Sciences Department
Uppsala University

## INTRODUCTION

Early question–answering systems often used *ad hoc* representations for their data bases, and corresponding *ad hoc* deduction methods for the question answering. Many systems, such as the SIR system (Raphael 1964) represent binary relations on *property-lists*. This term will be defined below. In recent years, it has been argued (e.g., Slagle 1965, Green and Raphael 1968) that a dialect of predicate calculus should be used instead. This would have two advantages: (1) predicate calculus is a richer language, i.e., more things can be said in it; (2) for predicate calculus, one knows reasonably efficient proof procedures, e.g., resolution (see Robinson 1965, Nilsson 1969).

The distinction between these two approaches is not clear-cut, and it is tempting to try to incorporate the use of property-lists to speed up resolution. For example, one may find it useful as the data base grows to construct, for each object symbol $c$, a chained list of all literals or clauses where $c$ occurs. This chained list is then (in a sense) a property-list for $c$. However, if we consider it as given that certain information is to be retrieved on property-lists, then it is not obvious that the best way to use this information is to feed it into the resolution black-box. It is natural to look for deduction procedures which utilize efficiently the kind of information that is stored and can be retrieved from property-lists, and which can be used instead of or together with conventional theorem-proving methods.

In this paper, we shall suggest one such alternative, namely a 'clean' property structure where binary relations $uRv$ are stored as address references. We shall show how formulas that involve quantifiers can sometimes be expressed simply by giving a certain structure to the '$R$'. We shall also specify a fast deduction procedure for this notation. An ultimate goal for work along these lines is that property-lists shall no longer be considered as an *ad hoc* notation.

## 1. PROPERTY-STRUCTURES AND PROPERTY-LISTS

The purpose of this section is to define what we mean by a property-structure and a property-list, and to give some notation. It will not contain any new results.

Let $U = \{u, v, w, \ldots\}$ be a set of objects, and let $\Delta = \{P, Q, \ldots\}$ be a set of binary relations on $U$ (i.e., subsets of $U \times U$). Consider the problem to design a computer program which has partial knowledge of these relations, which can accept more information about the relations and store it in a *data base*, and which can also accept questions about these relations and answer them from the data base. In the simplest case, assertions and questions are given as triples $\langle u, P, v \rangle$, meaning '$\langle u, v \rangle$ is a member of $P$', viz., 'is $\langle u, v \rangle$ a member of $P$?'. We want to organize the data base in such a way that new assertions can be added easily, and answers to questions can be retrieved easily and quickly.

Let us first discuss the use of such systems. Several previous question–answering programs are essentially of this type. In Lindsay's program (Lindsay 1962), $U$ is a set of people and families, and the relations are '$u$ is the husband in the family $v$', '$u$ is offspring in the family $v$', etc. Raphael encountered the same problem (Raphael 1964) with $U$ being a set of objects and people, and the relations being e.g., '$u$ is physically part of $w$', '$v$ is the owner of $x$', etc. Levien saw it (Levien and Maron 1965) with $U$ being a set of people, meetings, institutions, and documents, and relations such as '$u$ is the author of $v$', '$u$ is employed by $w$', etc. We met the same kind of problem ourselves when we decided to translate natural-language sentences like 'A gives B to C' into an expression

$$(\exists x) R_1(x, \mathrm{A}) \wedge R_2(x, \mathrm{Give}) \wedge R_3(x, \mathrm{B}) \wedge R_4(x, \mathrm{C})$$

where $x$ is the activity described by the sentence, $R_1$ can crudely be described as an 'activity-to-its-subject' relation, $R_2$ is an 'activity-to-its-verb' relation, etc.

One standard way of representing binary relations in the computer is through *property-structures*. We formally define a property-structure as a mapping

$$\sigma: U \rightarrow 2^{\Delta \times U}$$

i.e., a mapping which assigns a set of pairs $Rv$ to each member $u$ of $U$. ($Rv$ is an abbreviation for $\langle R, v \rangle$. We shall often use this abbreviation of the notation for tuples.) A property-structure $\sigma$ *corresponds to* a set $\phi$ of facts if

$$uRv \in \phi \equiv Rv \in \sigma(u)$$

If $\sigma$ is a property-structure and $Rv \in \sigma(u)$, we shall say that $u$ *has* the property $Rv$ in $\sigma$.

To represent a property-structure $\sigma$ in memory, one usually does as follows: a unique cell is associated with each member of $U$. A cell which is so associated is called an *atom*. The atom associated with $u$ will itself be called $u$. A property $Rv$ is represented as an indicator for $R$ plus the address of $v$.

All the properties that an atom $u$ has are stored in such a way that they can be accessed from $u$ as easily as possible. This may be done using a chained list (in which case each $\sigma(u)$ is represented as a *property-list*), through hash-coding, or by other means.

It is usually necessary to represent each fact through two properties. If $\mathcal{R}$ is the reverse relation of $R$, then a fact $uRv$ (equivalent to $v\mathcal{R}u$) is represented in a property-structure $\sigma$ through

$$Rv \in \sigma(u)$$

and

$$\mathcal{R}u \in \sigma(v)$$

If $R$ is symmetric, then $R$ and $\mathcal{R}$ are of course the same relation.

Figure 1 illustrates how $\phi = \{uPv, vRw\}$ can be represented as a property-list structure. Arrows stand for address references.

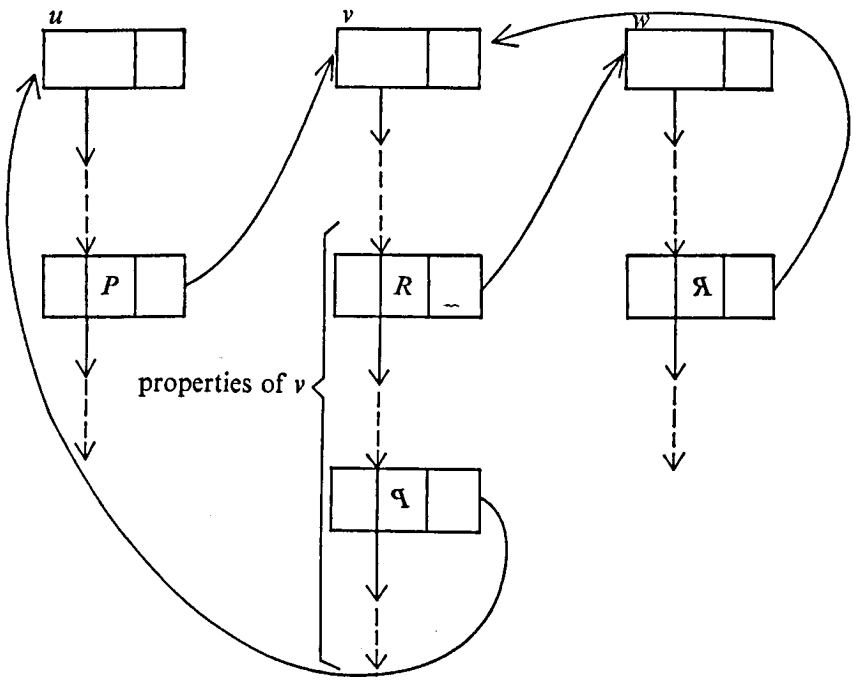



Figure 1

Let us contrast the property-structure representation of facts with the *linear* representation, where the data base consists of a straight list of all facts that have been accumulated. Clearly, the property-structure representation is superior for honoring simple information requests like 'is $uRv$ stored (explicitly) in the data base?'. However, all interesting question–answering programs must be able to perform logical deduction, i.e., to retrieve facts which are stored implicitly in the data base. It is at least not trivial to modify

conventional deduction procedures (e.g., resolution) so that they can make efficient use of a property-structure.

However, for certain types of inference rules, one can write deduction procedures which are particularly adapted for a property-structure data base. In particular, this is true about rules of the type

$$uPv, vQw \vdash uRw$$

(where $P$, $Q$, and $R$ need not be distinct). Such rules shall be called *chaining rules*, and shall be written

$$P \times Q \rightarrow R.$$

Chaining rules can be handled by the following.

**Linear chaining procedure**

To determine whether $uRw$ is stored (explicitly or implicitly) in a property-structure $\sigma$, do the following:

(a) See whether $Rw \in \sigma(u)$. If so, exit with success.

(b) For all pairs $\langle P, Q \rangle$ such that $P \times Q \rightarrow R$, and for all $v$ such that $u$ has the property $Pv$, set up the sub-problem to prove that $vQw$ is stored in $\sigma$. If this subproblem has not been attempted before, attempt it now by the linear chaining procedure. When no new sub-problems remain, exit with failure.

We call this procedure 'linear' because it only considers those properties $Pv$ which are explicitly stored under $u$. A more careful procedure would also consider properties $Pv$ that $u$ can be proved to have. Linear chaining corresponds to linear proofs in resolution. The linear chaining procedure is complete only if certain assumptions are made about the set of chaining rules. This question will be more thoroughly treated in another context.

It is easily seen that if the number of chaining rules is finite, and the number of facts in $\sigma$ is also finite, then this procedure will always terminate.

To summarize, a major advantage of the property-structure representation is that it permits fast retrieval and fast deduction, in some simple cases. A major *disadvantage* is that the notation is so restricted: no logical connectives are allowed (except the trivial *and* which connects all facts in the data base), and no quantifiers or other logical operators can be used.

The notation which we shall define in the next few sections consists essentially of a few operators on the relations $P$, $Q$, . . ., which enable us to express connectives and quantifiers in some cases. Using these operators, each fact would be written $uGv$, where $G$ has the form

$$A_1(A_2(\ldots A_j(R)\ldots)).$$

Such facts can of course be stored easily in a property-structure. Besides specifying a set of interesting operators, we shall give a set of chaining rules

$$F \times G \rightarrow K$$

where $F$, $G$, and $K$ have been formed using these operators.

## 2. QUANTIFICATION OF BINARY RELATIONS

We repeat that $U=\{u, v, w, \ldots\}$ is a set of objects, and $\Delta=\{P, R, \ldots\}$ is a set of binary relations on members of $U$. Starting from the relations in $\Delta$, we shall now define further relations on members of $U$ (*ground relations*) and on subsets of $U$ (*set relations*). The operators *Id*, *Rev*, *Neg*, *Aa*, *Ae*, *At*, *Ea*, *Ee*, and *Ta* on relations are defined as follows:

$$
\begin{aligned}
Id(R) &= R \\
Rev(R) &= \lambda(x, y)R(y, x) \\
Neg(R) &= \lambda(x, y)\neg R(x, y) \\
Aa(R) &= \lambda(b, c)(\forall x \in b)(\forall y \in c)R(x, y) \\
Ae(R) &= \lambda(b, c)(\exists y \in c)(\forall x \in b)R(x, y) \\
At(R) &= \lambda(b, c)(\forall x \in b)(\exists y \in c)R(x, y) \\
Ea(R) &= \lambda(b, c)(\exists x \in b)(\forall y \in c)R(x, y) \\
Ee(R) &= \lambda(b, c)(\exists x \in b)(\exists y \in c)R(x, y) \\
Ta(R) &= \lambda(b, c)(\forall y \in c)(\exists x \in b)R(x, y)
\end{aligned}
$$

In these definitions, we have used the abbreviations

$$
\begin{aligned}
(\forall x \in b)\mathscr{P} &\equiv (\forall x)[x \in b \supset \mathscr{P}] \\
(\exists x \in b)\mathscr{P} &\equiv (\exists x)[x \in b \wedge \mathscr{P}]
\end{aligned}
$$

We immediately obtain

$$Rev(Neg(R)) = Neg(Rev(R))$$

which can more compactly be written as

$$Rev \circ Neg = Neg \circ Rev$$

With similar notation, we obtain

$$Neg \circ Neg = Rev \circ Rev = Id.$$

For combinations of *Rev* and *Neg* with other operators, we obtain the table shown in figure 2.

| $\gamma$ | $Rev \circ \gamma$ | $Neg \circ \gamma$ |
|---|---|---|
| *Aa* | *Aa* o *Rev* | *Ee* o *Neg* |
| *Ae* | *Ea* o *Rev* | *Ta* o *Neg* |
| *At* | *Ta* o *Rev* | *Ea* o *Neg* |
| *Ea* | *Ae* o *Rev* | *At* o *Neg* |
| *Ee* | *Ee* o *Rev* | *Aa* o *Neg* |
| *Ta* | *At* o *Rev* | *Ae* o *Neg* |

Figure 2

Let $I$ be the equality relation (on members of $U$). We obtain the following set relations:

$$Ee(I) = \lambda(b, c)(b \text{ and } c \text{ overlap})$$
$$Aa(Neg(I)) = \lambda(b, c)(b \text{ and } c \text{ are disjoint})$$
$$At(I) = \lambda(b, c)(b \text{ is a subset of } c)$$
$$Ea(Neg(I)) = \lambda(b, c)(b \text{ is not a subset of } c).$$

In particular, $b[Ee(I)]b$ means '$b$ has some member', and $b[Aa(Neg(I))]b$ means '$b$ is the empty set'. The relation $Ea(I)$ may also be useful; $b[Ea(I)]b$ means '$b$ is a set of exactly one member'.

Consider now the set $\Gamma$ of all set-relations that can be constructed from $\Delta$ using these operators. Clearly, every member of $\Gamma$ can be written in exactly one way as

$$q(d(s(R)))$$

where $R$ is a member of $\Delta$, $s$ is either *Neg* or *Id*, $d$ is either *Rev* or *Id*, and $q$ is one of the six operators *Aa* through *Ta*.

A data base where members of $2^U \times \Gamma \times 2^U$ are represented in a property structure can be characterized as a *set-oriented property-structure representation for binary relations*. We introduce the acronym SPB for this representation.

Let us finally comment on how to visualize an SPB property structure. It should obviously be thought about as a network, where each node represents a subset of $U$, and each arc $aGb$ is a member of $2^U \times \Gamma \times 2^U$. If $G = q(d(s(R)))$, it is useful to think about the arc as directed from $a$ to $b$, and labeled with $d(s(R))$, whereas the $q$ is manifested by two connection codes $A$, $E$, or $T$ which indicate how the arc is connected to the nodes. $A$ stands for 'all members', $E$ stands for 'one and the same member', and $T$ stands for 'one variable member'. For example,

$$b[Ae(Rev(Neg(R)))]c$$

would be visualized as in figure 3:

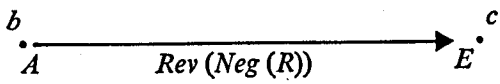



Figure 3

The same relation can be written as

$$c[Ea(Neg(R))]b$$

which is visualized as in figure 4:

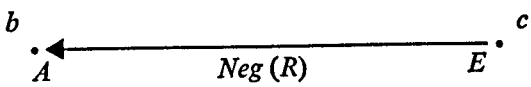



Figure 4

Thus the table for $Rev \circ \gamma$ above is thought about as saying 'the connection codes do not move when we reverse the arrow'.

The SPB property structure has the advantage of offering a simple and efficient way to store information about classes of objects in a data base.

Other approaches to the same problem (such as present attempts to find good refutation procedures for higher-order logic) are certainly more general, but their maturation to practical use seems to lie quite some way in the future.

## 3. GROUND LEVEL CHAINING RULES

By a *chaining rule*, we still mean a rule of the type

$$uPv, vQw \vdash uRw$$

which is written compactly as

$$P \times Q \rightarrow R$$

If $P$, $Q$, and $R$ are ground relations, then this is a *ground level* chaining rule. In this section, we shall study some equivalences which permit us to consider several, apparently dissimilar chaining rules as essentially the same.

We immediately notice

### 3.1. Proposition

If $P \times Q \rightarrow R$, then the following chaining rules also hold:

$$Rev(Q) \times Rev(P) \rightarrow Rev(R)$$
$$Neg(Rev(R)) \times P \rightarrow Neg(Rev(Q))$$
$$Q \times Neg(Rev(R)) \rightarrow Neg(Rev(P)).$$

*Proof* from the definitions of *Rev* and *Neg*.

### 3.2. Corollary

Iterated use of proposition 3.1 on a chaining rule $P \times Q \rightarrow R$ gives exactly six rules, viz., the four mentioned in the proposition, plus

$$Rev(P) \times Neg(R) \rightarrow Neg(Q)$$
$$Neg(R) \times Rev(Q) \rightarrow Neg(P)$$

*Proof* by inspection of all possible cases.

Notice that if the *Rev* and *Neg* operators are ignored, the six forms of a chaining rule are exactly the six permutations of the three elements $P$, $Q$, and $R$. Let us now look for a pattern among the *Rev* and *Neg* prefixes.

The rule $P \times Q \rightarrow R$ is clearly equivalent to

$$(\forall u)(\forall v)(\forall w) \neg (uPv \wedge vQw \wedge wR'u)$$

where $R' = Neg(Rev(R))$. We can illustrate the latter statement as a 'forbidden triangle' (Figure 5):

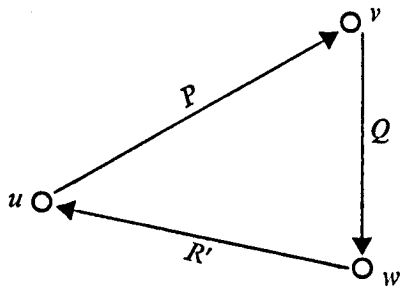



Figure 5

Moreover, if we write this forbidden triangle as a triple

$$\langle P, Q, Neg(Rev(R))\rangle$$

then it is immediately obvious that the same triangle can be written in five other ways, e.g.,

$$\langle Q, Neg(Rev(R)), P\rangle$$

or $$\langle Rev(P), Neg(R), Rev(Q)\rangle$$

Moreover, since each triple $\langle P, Q, R'\rangle$ is equivalent to a chaining rule $P\times Q\rightarrow Neg(Rev(R'))$, these six ways of writing the forbidden triangle immediately give us the six forms of the chaining rule derived in corollary 3.2.

Since the forbidden triangle is clearly the basic thing, we shall change our habits with respect to the symbol $R$ and talk about a forbidden triangle $\langle P, Q, R\rangle$, which is equivalent to the six chaining rules

$$P\times Q\rightarrow Neg(Rev(R))$$
$$Q\times R\rightarrow Neg(Rev(P))$$
$$R\times P\rightarrow Neg(Rev(Q))$$
$$Rev(R)\times Rev(Q)\rightarrow Neg(P)$$
$$Rev(Q)\times Rev(P)\rightarrow Neg(R)$$
$$Rev(P)\times Rev(R)\rightarrow Neg(Q).$$

Chaining rules are needed for the linear chaining procedure in section 1. However, it is more convenient to bypass the chaining rule and to describe the procedure directly in terms of forbidden triangles.

#### Linear chaining procedure (new description)

To determine whether $uRw$ is stored (explicitly or implicitly) in a property structure $\sigma$, perform the following:

(a) See whether $Rw\in\sigma(u)$. If so, exit with success.

(b) Otherwise, for all properties $Pv$ that $u$ has in $\sigma$, and for all $Q$ such that $\langle Rev(Neg(R)), P, Q\rangle$ is a forbidden triangle, set up the sub-problem to prove that $vQw$ is stored in $\sigma$. If this subproblem has not yet been attempted, attempt it by the linear chaining procedure. When no new subproblems remain, exit with failure.

The idea in step (b) is to prove that if $u[Neg(R)]w$, or equivalently, $w[Neg(Rev(R))]u$ were in the data base, then this would at least implicitly close a forbidden triangle there. This is indeed the case if we can prove that $vQw$ is in the data base, which is true if $v[Neg(Q)]w$ closes a forbidden triangle, etc. The process is illustrated in figure 6.

The chaining procedure runs through a sequence of forbidden triangles with a common vertex $w$. Triangles are connected by an inversion relationship ($Q$ to $Neg(Rev(Q))$) between adjacent sides of two successive triangles. Rays coming to the vertex $w$ stand for successive questions (things to be proved); rays going out stand for things to be disproved. Arcs along the

perimeter are in the data base. The procedure ends successfully when some ray coming into $w$ is also in the data base.

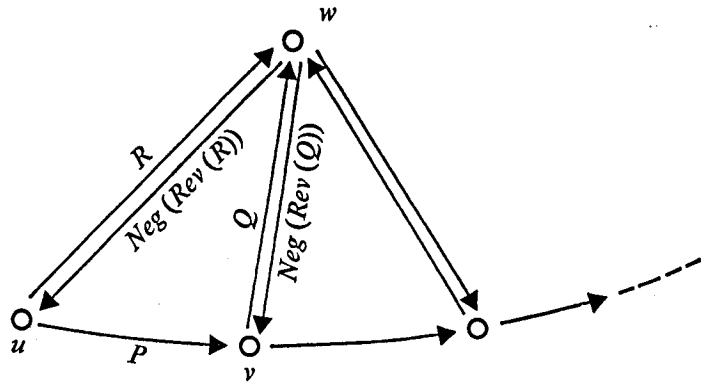



Figure 6

Notice that $w$ is not needed during chaining, but only in the termination criterion. Therefore, essentially the same chaining procedure can be used to answer questions of the type: 'which $w$ satisfies $uRw$?' (open questions, WH questions).

## 4. SET LEVEL CHAINING RULES

Let $\langle P, Q, R\rangle$ be a forbidden ground level triangle. In this section, we shall derive forbidden triangles on set relations obtained from $P$, $Q$, and $R$.

We can immediately write down a few of them:

$$\langle Aa(P), Ee(Q), At(R)\rangle$$
$$\langle Aa(P), Ee(Q), Ae(R)\rangle$$
$$\langle At(P), At(Q), Ae(R)\rangle$$
$$\langle At(P), Ae(Q), Ae(R)\rangle$$
$$\langle Ae(P), Ae(Q), Ae(R)\rangle$$

The reader should verify that these are indeed forbidden. For example, if the first triangle were not forbidden, we could have

$$b[Aa(P)]c \wedge c[Ee(Q)]d \wedge d[At(R)]b$$

i.e., all $b$ are $P$ing all $c$, some $c$ is $Q$ing some $d$, and for every $d$, there exists some $b$ which it is $R$ing. Consider one $c1$ in $c$ and one $d1$ in $d$ which are $Q$ing together. $d1$ is $R$ing some $b1$ in $b$. This $b1$, like all $b$, is $P$ing with all $c$ including $c1$. But this was a forbidden situation.

Now, if $\langle P, Q, R\rangle$ is a forbidden triangle, so is $\langle Q, R, P\rangle$, so we have five more set level forbidden triangles, e.g.,

$$\langle Aa(Q), Ee(R), At(P)\rangle.$$

In fact, there is no point in writing out all the set level forbidden triangles. Instead, we have the following *forbidden operator triangles*:

$\langle Aa, Ee, At\rangle$
$\langle Aa, Ee, Ae\rangle$
$\langle At, At, Ae\rangle$
$\langle At, Ae, Ae\rangle$
$\langle Ae, Ae, Ae\rangle$

plus the following rule:

*Rule of triangle composition.* If $\langle P, Q, R\rangle$ is a forbidden ground level triangle, and $\langle A, B, C\rangle$ is a forbidden operator triangle, then $\langle A(P), B(Q), C(R)\rangle$ is a forbidden set level triangle.

We have now compacted our chaining rules quite a bit. Let us illustrate this with an example. Consider the ground level chaining rule

$$I \times R \rightarrow R$$

(where $I$ is the equality relation, and $R$ is any relation). This corresponds to the forbidden ground level triangle

$$\langle I, R, Neg(Rev(R))\rangle.$$

Combining the six forms of this triangle with each of the five operator triangles, we obtain in principle 30 set level triangles, which means 180 set level chaining rules. Some of these are of course identical (basically because triangles like $\langle Ae, Ae, Ae\rangle$ do not change when they are rotated), and some are not very interesting, but we do obtain e.g.,

*from* $\langle At, Aa, Ee\rangle$:

1. $At(I) \times Aa(R) \rightarrow Neg(Rev(Ee(Neg(Rev(R))))) = Aa(R)$
   i.e., $\subset \times Aa(R) \rightarrow Aa(R)$
2. $Ee(I) \times At(R) \rightarrow Neg(Rev(Aa(Neg(Rev(R))))) = Ee(R)$
   where $Ee(I)$ is the 'overlap' relation
3. $At(R) \times Aa(Neg(Rev(R))) \rightarrow Neg(Rev(Ee(I)))$
   which is the 'disjoint' relation

*from* $\langle At, At, Ae\rangle$:

4. $At(I) \times At(R) \rightarrow Neg(Rev(Ae(Neg(Rev(R))))) = At(R)$
   where $At(I)$ is the 'subset' relation
5. Specializing (4) to the case $R = I$ we obtain
   $\subset \times \subset \rightarrow \subset$
6. $At(R) \times Ae(Neg(Rev(R))) \rightarrow Neg(Rev(At(I)))$
   which is the 'is not superset' relation

and quite a number of others.

If the five forbidden operator triangles are visualized with connection codes as at the end of section 2, we notice

(1) there is one '*A*' connection and one '*T*' or '*E*' connection at each corner of the triangle;
(2) there is at least one '*E*' connection in each triangle;
(3) every triangle that satisfies (1) and (2) is a forbidden one.

Observations such as these can be used in programming the linear chaining procedure for set level triangles. This will be the topic of next section.

## 5. THE LINEAR CHAINING PROCEDURE ON SET LEVEL

The purpose of this section is to specify in detail the algorithm for linear chaining with set level chaining rules.

We assume that expressions

$$b[q(d(s(R)))]c$$

(with $q$, $d$, and $s$ as in section 2) are stored in a property structure so that $b$ obtains the property

$$\langle q, d, s, r, c\rangle$$

and $c$ obtains the property

$$\langle Rev(q), \neg d, s, r, b\rangle$$

where $Rev(q)$ is defined to be the $q'$ in the equality

$$Rev(q(P)) = q'(Rev(P)).$$

This definition should be natural. Our notation is helpful: we have $Rev(Ae) = Ea$, etc. In the above fivetuples, it is assumed that $d$ and $s$ are represented as boolean variables. Moreover, we assume $q$ to be represented as a triplet $\langle q1, q2, q3\rangle$ of boolean variables according to the conventions shown in figure 7.

| $q$ | $q1$ | $q2$ | $q3$ |
|---|---|---|---|
| $Aa$ | 0 | 0 | 0 |
| $Ee$ | 1 | 1 | 1 |
| $At$ | 0 | 1 | 1 |
| $Ea$ | 1 | 0 | 0 |
| $Ae$ | 0 | 1 | 0 |
| $Ta$ | 1 | 0 | 1 |

Figure 7

With these codes, we have

$$Rev(q1, q2, q3) = \langle q2, q1, q3\rangle.$$

Moreover, if $Neg(q)$ is defined similarly to $Rev(q)$, we have

$$Neg(q1, q2, q3) = \langle \neg q1, \neg q2, \neg q3\rangle$$

If the components of a property are packed into one computer word, then complementation in given bit positions may be faster than permutations of bits. We can define the $Rev$ operation through complementation if $Aa$ is given a double representations as (0, 0, 0) or (1, 1, 0), and $Ee$ is given a double representation as (0, 0, 1) or (1, 1, 1). In this case,

$$Rev(q1, q2, q3) = \langle \neg q1, \neg q2, q3\rangle.$$

The 'meaning' of the various bits in the representation of $q$, are:
$q1$: is there an $E$ or $T$ connection at this end of the arrow?
$q2$: is there an $E$ or $T$ connection at the other end of the arrow?
$q3$: if there is exactly one $E$ or $T$: is this a $T$? (otherwise, conventions are arbitrary).

However, these interpretations only apply to the case of single representation of *Aa* and *Ee*.

Let us now introduce one more operation on the $q$, besides *Rev* and *Neg*. With given operators $q$ and $q'$, we shall write $q+q'$ for that operator for which $\langle q, q', q+q' \rangle$ is a forbidden operator triangle. $q+q'$ may have no, one, or two values, and is given in the figure 8.

| | $q'=$ | 000 | 111 | 011 | 100 | 010 | 101 |
|---|---|---|---|---|---|---|---|
| $q=$ | | *Aa* | *Ee* | *At* | *Ea* | *Ae* | *Ta* |
| 000 | *Aa* | | *Ae, At* | | *Ee* | | *Ee* |
| 111 | *Ee* | *Ea, Ta* | | *Aa* | | *Aa* | |
| 011 | *At* | *Ee* | | *Ae* | | *Ae, At* | |
| 100 | *Ea* | | *Aa* | | *Ea, Ta* | | *Ea, Ta* |
| 010 | *Ae* | *Ee* | | *Ae, At* | | *Ae, At* | |
| 101 | *Ta* | | *Aa* | | *Ea, Ta* | | *Ea* |

Figure 8. Table for $q+q'$

As we see, we sometimes obtain both *Ae* and *At*, or both *Ea* and *Ta* as a 'sum'. We shall make the $+$operation unambigous by defining $q+q'$ as *At*, viz., *Ta* in these cases. This convention will be explained below.

Blank squares in the table indicate that no sum exists. It is easily verified that with our encodement of the $q$ (with the single representation for *Aa* and *Ee*), we obtain

$$\langle q1, q2, q3 \rangle + \langle q1', q2', q3' \rangle =$$
**if** $q2 = q1'$ **then** *undefined*
**else** $\langle \neg q2', \neg q1, \neg (q1 \wedge q2' \vee q3 \wedge q3') \rangle$.

It follows that

$$Neg(Rev(\langle q1, q2, q3 \rangle + \langle q1', q2', q3' \rangle)) =$$
**if** $q2 = q1'$ **then** *undefined*
**else** $\langle q1, q2', (q1 \wedge q2' \vee q3 \wedge q3') \rangle$.

We now have all the preliminaries for the chaining procedure. Given a question *bGd*, we attempt to disprove

$$d[Neg(Rev(G))]b$$

Let this relation be represented by assigning the tuple

$$\langle q1, q2, q3, d, s, r, b \rangle$$

to $d$, as usual. For each property

$$\langle q1', q2', q3', d', s', r', c \rangle$$

that $b$ has in the data base, we shall clearly do the following:

**Set level chaining procedure (details)**

(1) Determine a suitable ground level forbidden triangle. This is essentially an 'addition' of $\langle d, s, r\rangle$ and $\langle d', s', r'\rangle$. It will have to be handled by a programmed routine, or by table look-up.

*Example*: The ground level chaining rule

$$R \times I \rightarrow R$$

is handled by the function

$\langle d, s, r\rangle + \langle d', s', r'\rangle =$
**if** $r = r'$ **and** $d \neq d'$ **and** $s \neq s'$ **then** $\langle 0, 1, I\rangle$
**elseif** $r = I$ **and** $s = 1$ **then** $\langle \neg d', \neg s', r'\rangle$
**elseif** $r' = I$ **and** $s' = 1$ **then** $\langle \neg d, \neg s, r\rangle$
**else** *undefined.*

We assume here that $s = Neg$ is stored as 0, and $s = Id$ is stored as 1.

(2) If one or more ground level triangles exist, determine an operator triangle $\langle q1, q2, q3\rangle + \langle q1', q2', q3'\rangle$ that can be applied to it.

(3) Combine the third side of the ground level triangle and the operator triangle (this is essentially a concatenation of the bit sequences). We wish to prove that the resulting relation $G'$ satisfies $c[G']d$. To prove this, attempt to disprove $d[Neg(Rev(G'))]c$. This is done by running through steps (1)–(3) again for each property that $c$ has.

We can now see why + could be disambiguated in figure 8. The + operation is used to construct something that we are to prove, and it is always easier to prove a relation of the type $At(P)$ than of the type $Ae(P)$. Let us see what happens in the two cases! After neg-reversion, $Ae$ has gone into $At$, and $At$ into $Ae$. Thus our strategy tells us to disprove $Ae(P)$ type relations, and to ignore $At(P)$ types. This is all right, because if $\langle At, B, C\rangle$ is a forbidden triangle, so is $\langle Ae, B, C\rangle$. When we use the disambiguated table for +, we need only do one thing to account for the double sum that was removed: modify the termination criterion for the chaining procedure. It shall now go as follows:

*Termination of set level chaining*

The chaining procedure terminates successfully if either of the following things happen:

(a) we are told to prove $cHd$, and this relation is already stored in the data base (explicitly);

(b) we are told to prove $c[At(P)]d$, and $c[Ae(P)]d$ is already in the data base;

(c) we are told to prove $c[Ta(P)]d$, and $c[Ea(P)]d$ is already in the data base.

The procedure terminates with failure when there are no new subproblems.

With the above chaining procedure, we will perform addition on the $\langle d, s, r\rangle$ and the $q$ independently, concatenate the results, and then perform

the *Neg–Rev* operations on the entire relation. Since the *Neg* and *Rev* operations go independently on $q$ and $s$, viz., on $q$ and $d$, we can perform the *Neg–Rev* operation immediately after addition in steps (1) and (2), and concatenate the results after *Neg–Rev*-ing. This is attractive because the operation $Neg(Rev(q+q'))$ looks simpler than $q+q'$, when expressed in terms of bit operations. The same thing holds for the addition of $\langle d, s, r\rangle$; the example in step (1) above can be rewritten as

$$Neg(Rev(\langle d, s, r\rangle + \langle d', s', r'\rangle)) =$$
**if** $r=r'$ **and** $d\neq d'$ **and** $s\neq s'$ **then** $\langle 0, 0, I\rangle$
**elseif** $r=I$ **and** $s=1$ **then** $\langle d', s', r'\rangle$
**elseif** $r'=I$ **and** $s'=1$ **then** $\langle d, s, r\rangle$
**else** *undefined.*

(Notice that direction $d$ is immaterial for equality.) Unfortunately, it is difficult to manage completely without the 'clean' (non-*Neg–Rev*-ed) sums, since they are needed in the termination criterion. Therefore, we either need to do a Neg–Reversion operation in each cycle of the chaining procedure, or to store both a 'clean' and a 'neg–reversed' copy of the relation in each property in the data base.

## 6. REPRESENTATION OF MORE COMPLEX LOGICAL RELATIONSHIPS

Although the introduction of operators *Aa*, *Ae*, etc., increases the expressive power of property structure notation, it still does not approach the power of predicate calculus. In this section we shall say a few words about how more complex logical relationships can be represented.

Let us assume that the system which uses the SPB data base (usually, a question–answering system) has a *fixed* set $\Delta$ of ground level relations, and that only constants (for subsets of $U$) may be added to the system while it is running. This assumption is usually valid. It is then reasonable to distinguish between two types of logical expressions:

### (a) Expressions that do not contain any constant

*Example* 1. We may have ground relations $P$ and $Q$ for which $uPv$ always implies $uQv$.

*Example* 2. On the set level, we have two rules that arise from the forbidden situation

$$b[Ee(R)]d,\ d[Aa(Neg(I))]d$$

where $I$ is the equality relation and $R$ is any ground level relation.

*Example* 3. Again on the set level, we have the general rule that $b[Ae(R)]d$ implies $b[At(R)]d$.

Expressions of this kind clearly should not have to be added to the system's data base at run time, since all such expressions can be analysed at system design time, when the set of relations $\Delta$ is selected. For efficiency reasons, such deduction rules should be represented implicitly in the deduction

procedure, e.g., as specialized subgoal generators that co-operate with the chaining procedure. We have already carried this out for example 3. The other examples can be handled in a similar fashion. There is then no need to design a property-structure representation for such expressions.

**(b) Expressions that do contain constants**

*Example*: 'If you drop an object, it will hit the ground.' (With the representation of natural-language information that we use in our project, and that was outlined in section 1, 'drop', 'object', 'hit', and 'the ground' will be taken as constants here.)

It must certainly be possible to add statements such as this one to an SPB data base during a conversation. It is proposed that this is done in the following way:

1. Nodes on the SPB data base may be of two kinds: *constants* and *variables*. Constants behave as we are used to from previous sections in this paper; for variables there are some new conventions.
2. Variables have arcs coming and going, just like other nodes. However, we introduce one more connection code besides *A*, *E*, and *T* (cf. section 2). The new code is written *D*, which stands for Definition. Correspondingly, we introduce new operators *Ad*, *Da*, *Dd*, *Dt*, and *Td*.
3. The use of variables is outline by the following example: the rule

$$(\forall\alpha) \qquad \alpha[At(P)]b \wedge \alpha[Aa(Q)]c \supset \alpha[Aa(R)]d$$

(where *b*, *c*, and *d* are constants) is represented in the SPB data base by a variable *a*, which is a node in the arcs (or 'relations')

$$a[Dt(P)]b$$
$$a[Da(Q)]c$$
$$a[Aa(R)]d$$

In this case, the variable *a* can be interpreted as the lowest upper bound of all sets $\alpha$ which satisfy $\alpha[At(P)]b$ and $\alpha[Aa(Q)]c$. It follows that it is meaningful to consider the set of all the *D*-connected arcs that go to a variable node, but not a *D*-connected arc in isolation.

The idea of using variables in this way has one important advantage: it integrates well with the chaining procedure and other deduction methods in the SPB data base. Thus, in a simple case, the variable *a* may be used as follows:

(a) It is desired to prove that $e[Aa(R)]d$.
(b) Through chaining, we establish the subgoal to prove that *e* is a subset of *a*.
(c) The deduction procedure notices that *a* is a variable, and generates the two AND-connected subgoals: prove that $e[Aa(Q)]c$ and $e[At(P)]b$. (These are the *D*-connected arcs in *a*, with an *A* inserted instead of the *D*.)

A more systematic treatment of the use of variables will be given in later papers.

**Acknowledgements**

This research has been supported in part by the Swedish Natural Science Research Council (contract Dnr 2711-6) and by the (Swedish) Research Institute of National Defense (bestalln. 719925).

## REFERENCES

Green, C.C. & Raphael, B. (1968) The use of theorem-proving techniques in question-answering systems. Paper at 1968 ACM Conference, Las Vegas.

Levien, R. & Maron, M.E. (1965) Relational Data File: A tool for mechanized inference execution and data retrieval. *RM-4793-PR.* Santa Monica, California: RAND Corp.

Lindsay, R.K. (1962) A program for parsing sentences and making inferences about kinship relations. *Proc. of Western Management Science Conference on Simulation* (ed. Hoggatt, A.).

Nilsson, N.J. (1969) *Predicate Calculus Theorem Proving.* Menlo Park, California: SRI.

Raphael, B. (1964). SIR – *A computer program for semantic information retrieval* Ph.D. thesis). Cambridge, Mass.: MIT Math Dept.

Robinson, J.A. (1965) A machine-oriented logic based on the resolution principle. *J. Ass. comput. Mach.*, **12**, 23–41.

Slagle, J.R. (1965) A proposed preference strategy using sufficiency-resolution for answering questions. *UCRL*-14361. California: Lawrence Radiation Laboratories.

# 14

# Rediscovering some Problems of Artificial Intelligence in the Context of Organic Chemistry

B. G. Buchanan
G. L. Sutherland and
E. A. Feigenbaum
Computer Science Department
Stanford University

## 1. THE MASS SPECTROMETRY PROGRAM

The set of computer programs known as HEURISTIC DENDRAL is an attempt to develop machine intelligence in a scientific field. In particular its task domain is the analysis of mass spectra, chemical data gathered routinely from a relatively new analytical instrument, the mass spectrometer. HEURISTIC DENDRAL has been developed as a joint project of the Departments of Computer Science, Chemistry, and Genetics at Stanford University. This collaboration of chemists and computer scientists has produced what appears to be an interesting program from the viewpoint of artificial intelligence and a useful tool from the viewpoint of chemistry.

For this discussion it is sufficient to say that a mass spectrometer is an instrument into which is put a minute sample of some chemical compound and out of which comes data usually represented as a bar graph. This is what is referred to here as the mass spectrum. The instrument itself bombards molecules of the compound with electrons, thereby producing ions of different masses in varying proportions. The $x$-points of the bar graph represent the masses of ions produced and the $y$-points represent the relative abundances of ions of these masses.

The HEURISTIC DENDRAL process of analysing a mass spectrum by computer consists of three phases. The first, preliminary inference (or planning), obtains clues from the data as to which classes of chemical compounds are suggested or forbidden by the data. The second phase, structure generation, enumerates all possible explicit structural hypotheses which are compatible with the inferences made in phase one. The third phase, prediction and testing, predicts consequences from each structural

hypothesis and compares this prediction with the original spectrum to choose the hypothesis which best explains the data. Corresponding to these three phases are three subprograms. The program(s) have been described in previous publications, primarily in *Machine Intelligence 4*, and in a series of Stanford Artificial Intelligence Project Memos (54, 62, 67, 80).

The PRELIMINARY INFERENCE MAKER program contains a list of names of structural fragments, each of which has special characteristics with respect to its activity in a mass spectrometer. These are called 'functional groups'. Each functional group in the list is a LISP atom, with properties specifying the necessary and/or sufficient conditions (spectral peaks) which will appear in a mass spectrum of a substance containing that fragment. Other properties of the functional group indicate which other groups are related to this one – as special or general cases.

The program progresses through the group list, checking for the necessary and sufficient conditions of each group. Two lists are constructed for output: GOODLIST enumerates functional groups which might be present, and BADLIST lists functional groups which cannot be in the substance that was introduced to the mass spectrometer.

GOODLIST and BADLIST are the inputs to the STRUCTURE GENERATOR, which is an algorithmic generator of isomers (topologically possible graphs) of a given empirical formula (collection of atoms). Each GOODLIST item is treated as a 'superatom', so that any functional group inferred from the data by the PRELIMINARY INFERENCE MAKER will be guaranteed to appear in the list of candidate hypotheses output by the STRUCTURE GENERATOR.

The STRUCTURE GENERATOR's operation is based on the DENDRAL algorithm for classifying and comparing acyclic structures (Lederberg, unpublished). The algorithm guarantees a complete, non-redundant list of isomers of an empirical formula. It is the foundation for the development of the whole mass spectrometry program.

The third subprogram is the MASS SPECTRUM PREDICTOR, which contains what has been referred to as the 'complex theory of mass spectrometry'. This is a model of the processes which affect a structure when it is placed in a mass spectrometer. Some of these rules determine the likelihood that individual bonds will break, given the total environment of the bond. Other rules are concerned with larger fragments of a structure – like the functional groups which are the basis of the PRELIMINARY INFERENCE MAKER. All these deductive rules are applied (recursively) to each structural hypothesis coming from the STRUCTURE GENERATOR. The result is a list of mass-intensity number pairs, which is the predicted mass spectrum for each candidate molecule.

Any structure is thrown out which appears to be inconsistent with the original data (i.e., its predicted spectrum is incompatible with the spectrum). The remaining structures are ranked from most to least plausible on the

basis of how well their spectra compare with the data. The top ranked structure is considered to be the 'best explanation'.

Thanks to the collaboration of Dr Gustav Schroll, an NMR (Nuclear Magnetic Resonance) PREDICTOR and INFERENCE MAKER have been added to the program. Thus the program can confirm and rank candidate structures through predictions independently of mass spectroscopy, bringing the whole process more in line with standard accounts of 'the scientific method'. Thus the HEURISTIC DENDRAL program is expanding from the 'automatic mass spectroscopist' to the 'automatic analytical chemist'. Other analytical tools, such as infra-red spectroscopy, will be incorporated eventually.

Three papers have appeared in the chemical literature (Duffield *et al.* 1969, Lederberg *et al.* 1969, Schroll *et al.*, in press) in the past year. The first paper describes the HEURISTIC DENDRAL program and tabulates numbers of isomers for many compounds. This is of particular interest to chemists because it indicates the size of the search space in which structures must be found to match specific data. The second paper explains the application of the program to ketones: the subclass of molecular structures containing the keto radical (C=O). The whole process from preliminary inference (planning) through structure generation and prediction of theoretical spectra was applied to many examples of ketone spectra. The results, in terms of actual structures identified, were encouraging. The third paper explains the application of the program to ethers. Introducing the NMR PREDICTOR contributed to the successful results which are described in the ether paper.

Acceptance of these papers by a chemistry journal is some measure of the program's capability, but indicates more its novelty and potential. A better measure of its performance level is provided by comparing the program with professionals. In July (1969) Professor Carl Djerassi, an eminent mass spectroscopist, asked the members of his graduate mass spectrometry seminar to interpret three mass spectra, giving them only the empirical formulas of the structures and stating the fact that they were acyclic structures – just the information given to the program. On the first problem, the program and one graduate student got the correct structure; another graduate student and a post-doctoral fellow were both close, but not correct. On the second problem, the program got the correct answer; two graduate students included the correct answer in undifferentiated sets of two and four structures; while the post-doctoral fellow missed the answer. On the last problem, the program missed the correct structure and the post-doctoral fellow included it in a pair of equally likely structures. The computer spent approximately two to five minutes on each problem; the chemists spent between fifteen and forty minutes on each. From this small experiment and their own observations, (admittedly sympathetic) mass spectroscopists have said the program performs as well as graduate students and post-doctoral fellows in its limited task domain.

One obvious reason for the encouragingly high level of performance of the computer is the large amount of mass spectrometry knowledge which chemists have imparted to the program. Yet this has been one of the biggest bottlenecks in developing the program. When there was only one theory of mass spectrometry in the program, viz., the complex theory in the PREDICTOR, we were relatively insensitive to the difficulty of adding new information to the theory. Although it was a time-consuming process, it was still manageable by one programmer, working with one chemist, with most of the time spent programming as opposed to criticizing. By the time the planning phase was added to the program, it was easier to see how to shorten the task of programming by separating the chemical theory from the routines which work on the theory. The separation was by no means complete here, but it was successful enough to reduce the programming time drastically for the addition of new pieces of theory. Because the theory could be changed by changing an entry in a table, many iterations with the expert were now possible in a single one or two hour session at the console. The preponderance of time was now spent by the chemist deciding how to change the rules in the table to bring the program's behaviour more in line with real data.

The organization of the PRELIMINARY INFERENCE MAKER made the process of examining its chemical knowledge relatively simple, compared to the process of putting knowledge into the STRUCTURE GENERATOR and PREDICTOR programs. Both of these programs are on their way to becoming 'table driven' in much the same way as the PRELIMINARY INFERENCE MAKER is now. (See Part 4.) Yet, re-designing the programs to allow easy additions and changes to the chemical knowledge will not solve all our problems. Because mass spectroscopy is a relatively young discipline, the theory does not exist in any sort of comprehensive codified form. Part 2 will discuss some of the problems of obtaining the chemical theory that has been incorporated into the programs so far. Further, the presence of any body of knowledge in the programs brings up questions of how and where this knowledge is to be represented, stored, and referenced within the programs. Part 3 will elaborate on these issues.

## 2. ELICITING A THEORY FROM AN EXPERT

As in the case of the Greenblatt chess program, the proficiency of the mass spectrometry program is due in large measure to the great number of times the behaviour of the program has been criticized by good 'players', with subsequent modifications to the program. In both cases, the heuristics of good play were not born full-blown out of the head of the programmer; they were built up, modified and tuned through many interactions with persons who were in a position to criticize the performance of the program. Yet one of the greatest bottlenecks in our total system of chemists, programmers and program has been eliciting and programming new pieces of information about mass spectrometry. One problem is that the rate of information

transfer is much slower than we would like. And another problem is that the theory itself is not as well defined as we had hoped. Since these two problems are common to a broad range of artificial intelligence programs, our encounter with them will be described in detail.

It should be understood from the start that there presently is no axiomatic or even well-organized theory of mass spectrometry which we could transfer to the program from a text book or from an expert. The theory is in very much the same state as the theory of good chess play: there exists a collection of principles and empirical generalizations laced throughout with seemingly *ad hoc* rules to take care of exceptions. No one has quantified these rules and only a few attempts have been made to systematize them. Thus the difficulty in eliciting rules of mass spectrometry from the expert lies only partly in the clumsiness of the program; the primitive state of the theory certainly contributes to our difficulty too. In our case, this problem has been compounded by having the theory of mass spectrometry in two different forms in the program: one in the prediction phase, and a less complex – but hopefully compatible – theory in the planning phase. The implications of this added difficulty will be discussed in Part 3.

The following dialog illustrates some of the difficulties we encountered at the console, apart from machine troubles and programming problems. It is not a literal transcript, but both parties to the actual dialog agree that it is a fair condensation of some of the sessions in which they focused on the PREDICTOR's theory of mass spectrometry. The sessions, typically, were one or two hours long. Because much of the process depended on what the program could do, both parties sat at a teletype tied to the PDP 10 time-sharing system in which the LISP programs resided. The expert in this dialog is A, the programmer is B, and meta-comments are bracketed.

**First session**

A: Since El Supremo and the rest want us to work on ketones, I guess we should get started.

B: OK. Incidentally, why are ketones important?

A: Besides being very common in organic chemistry we also know something of their mass spectrometry because they've been studied a lot.

B: What subgraph exactly will cause a molecule to be classed as a ketone?

A: The keto, or carbonyl, radical. That is —C=O (noticing B's puzzled look).

B: Then all of these are ketones?

$CH_3—CH_2—C=O—R$
$CH_3—C=O—R$
$H—C=O—R$

A: Wait a minute. The first two are ketones, but the last one is a special case which we should distinguish in the program. It defines the class of aldehydes.

B: So can we formulate the general rule that a ketone is any molecule containing C—C=O—C (thinking of the LISP list '(C(2 O)(1 C)(1 C))').

A: That's it.

B: Now what mass spectrometry rules do you have for ketones?

A: Three processes will dominate: alpha-cleavage, elimination of carbon monoxide from the alpha-cleavage fragments, and the McLafferty rearrangements.

B: OK. I wrote those down – now tell me exactly what each one means. Start with alpha-cleavage – do you mean the bond next to the heteroatom?

A: (Digression on notation – often alpha-cleavage would mean this bond, but not for mass spectrometry.) . . . Here alpha cleavage is cleavage of the C—C=O bond, i.e., cleavage next to the carbonyl radical – on both sides don't forget.

B: All right. That's an easy rule to put in (translating to a new LISP function which defines alpha-cleavage as cleavage which results in a fragment (i.e., a list) whose first atom has a non-carbon atom on its property list). Shall we say the peaks are always high?

A: That will do as a start. We don't really pay much attention to intensities just as long as the peaks show up.

(Reasons why exact intensities cannot be computed are explained briefly – B's interpretation is that chemists just don't know enough about them.)

B: Now let's get on to the second process – loss of carbon monoxide from the alpha-cleavage fragments. Would you write that out in detail? Exactly what happens to the fragment $CH_3—CH_2—C=O$ for instance?

A: Let's see, that is mass 57. You will see a high 57 peak for this fragment and you'll also see a 29 peak because of this process:

$$CH_3-CH_2-\overset{57}{C}=O-CH_2-CH_3 \xrightarrow{-CO} CH_2-CH_3 \qquad (m/e = 29)$$

B: Is that all there is to it, just drop off the C=O from the fragment (thinking of making a second call to the LISP function which breaks bonds and returns fragments). Does this happen in every case?

A: Yes it's that simple.

B: What about the intensities of these new peaks?

A: Well, as far as we know they'll be pretty strong all the time. Let me check some spectra here. (A looks through a notebook containing some mass spectra of simple ketones to check on the relative abundance of alpha-cleavage minus 28 peaks.) Well some of the time they're not recorded below mass 40 so it's a little hard to say. But it looks like the

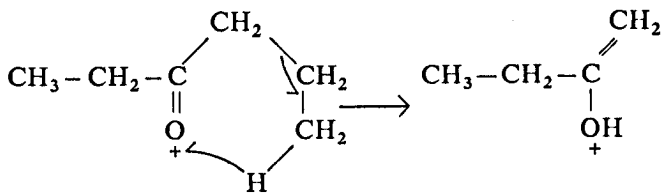
alpha-cleavage minus 28 peaks are about half as strong as the alpha-cleavage peaks in most cases.

(A and B digress on the generality of the process; A thinks of the chemical processes, while B thinks of their LISP representation.)

A: (Finally.) Now the last important process for ketones, and this also holds for aldehydes too, is the McLafferty rearrangement. That is just beta-cleavage with migration of the gamma hydrogen.

B: You lost me again. Would you write down an example?

A: Take the case we've been working with, but with a normal propyl on the one side. Here's how we would show what's going on:




m/e = 100 m/e = 72

B: I guess I still don't understand. Would you mind going through that step by step?

A: We can't really say what the sequence of events is, just that from the molecular ion of mass 100 you get another ion of mass 72 – the McLafferty rearrangement is just one way of explaining how that happens.

(Digression on how chemists can be confident of what the process is, including some discussion of deuterium labeling, and meta-stable transition peaks.)

B: Suppose we wanted to tell the program about McLafferty rearrangements, as I guess we do. What do I tell it in this case?

(A and B work out the details step by step as best they can. Both A and B suffer from B's lack of experience.)

B: Let's see if I have this straight with another example.

(B picks an example which is too difficult for the first approximation to the rules which he understands at this point. This leads to a lengthy discussion of the conditions under which just one McLafferty rearrangement will occur, and conditions under which a 'double McLafferty' will occur. At the end, B's most valuable possession is a piece of paper on which A has sketched several examples with cryptic notes. B promises to program these three rules, knowing full well that he won't get them right the first time but knowing that it will be easier for A to correct specific errors than to understand everything at once. A promises to review the published spectra of simple ketones to come up with some closer estimates of the relative intensities of the peaks resulting from these processes.)

**Second session**

B: The program and I are a little smarter than last time. But we both need some help. Let me show you what it does with a few specific examples. (B calls the program, and types in a few examples.)

(At this point, A looks at the examples and their corresponding entries in the notebook of actual mass spectra. As he looks he diagrams the processes – typically all processes for a molecule are superimposed on the graph structure of the molecule, with arrows pointing out of the main graph to the graphs of 'daughter ions'.)

A: In all these cases the alpha-cleavages are pretty good, the alpha-cleavage minus 28 peaks are OK most of the time, but I don't understand what the program is doing with McLafferty rearrangements. Also, there are a couple of things that I didn't mention last time – I remembered them as I reviewed the ketone literature last night; so naturally the program doesn't know about them.

B: Let me write these down.

A: Two things: there is a difference in relative abundance of the alpha-cleavage peaks depending on whether it is major alpha or minor alpha, and second, very often you will see a McLafferty plus one peak after the McLafferty rearrangements.

B: Let's come back to those after you've told me what is wrong with the program as far as it goes.

A: (Looking at the examples run by the program.) In the first case you have the alpha-cleavage and alpha minus carbon monoxide peaks. But what are these others?

B: Let's see. (B inputs the example again with a switch turned on which allows him to see which major functions get executed and what their results are.) The program thinks it can do a double McLafferty rearrangement – isn't that right?

A: It should do one McLafferty rearrangement, but I don't see the right peak for that. Here is the one it should do (sketching it out). It looks like you've tried to do something quite different.

(After much time the errors are traced to a basic misunderstanding on B's part and some programming errors.)

B: Well I guess I'd better take care of those things before you look at more examples. Perhaps I can add those other things you mentioned earlier. What's this business about major alpha and minor alpha?

A: It is just a way of bringing the intensities predicted by the program more in line with the actual intensities. In these examples the major alpha-cleavage is the alpha-cleavage in which the larger alkyl fragment is lost.

(A sketches several examples to illustrate his point.)

B: What sort of general principle defines the minor alpha?

A: The larger alkyl fragment lost.

(B agrees to put this in the program after getting it clear. A new LISP function is mostly conceptualized by now. Within a few months, however, some poor results were traced to this form of the principle, so it had to be reformulated to consider more than merely the size of the fragment.)

B: Now what about the other thing – the McLafferty-plus-one-peaks?

A: Well, we don't know much about it, but it seems that in almost all cases where you see a McLafferty rearrangement peak you also see a peak at one mass unit higher. Of course we can't say where the extra mass comes from, but it doesn't really matter.

B: Suppose the program just sticks in the extra peak at $x+1$ for every $x$ from a McLafferty rearrangement?

(B's suggestion is motivated by the existing LISP code. The only time the program knows it has a McLafferty peak is inside one function. After a brief discussion of this, both A and B decide that the next step is to get the program to make more accurate predictions. The discussion switches, then, to adding this ketone information to the planning phase of the program.)

After deciding upon an interesting class of organic molecules, such as ketones, ethers, or amines, the first step toward informing the program about the mass spectrometry theory for that class is to ask a mass spectroscopist what rules he generally uses when considering molecules of the class. His first answer is that he expects specific fragmentations and rearrangements to dominate the entire process, with different mass numbers resulting in different contexts. He expects just four processes to explain all significant peaks in the mass spectra of acyclic ketones: (1) cleavage next to the C=O (keto) group, i.e., alpha-cleavage, (2) loss of carbon monoxide (CO) from the ions resulting from alpha-cleavage, (3) the rearrangement process known as the 'McLafferty rearrangement' (migration of the gamma hydrogen to the oxygen with subsequent beta-cleavage), and possibly (4) addition of a proton to ions resulting from McLafferty rearrangements. The last process is given far less weight than the first three, seemingly because there are still too many exceptions to put much confidence in it. But it is still useful, enough of the time, to warrant inclusion in the list. It is impossible to identify a process with any specific mass number because these processes result in different spectral lines when applied to different structures. For example, alpha-cleavage (next to the C=O) in C—C—C—C—C=O—C—C results in peaks at mass points 57 and 71 while in C—C—C—C—C—C=O—C—C the alpha-cleavage peaks are at mass points 57 and 85.

These four rules were put into the PREDICTOR's complex theory and, in a different form, into the rough theory of the planning stage. The problems we encountered with these rules are typical of three fundamental problems we have learned to expect: (1) unanticipated concepts require additional programming, (2) counter-examples to the first rules force revisions, and (3) a false start leads to a change in strategy.

The first difficulty is just a variation on the old adage 'Expect the unexpected'. In our case one root of this problem is lack of communication between expert and non-expert. Because the expert tries to make his explanations simple enough for the layman he leaves out relations or concepts which very often turn out to be important for the performance of the program.

Initially the PREDICTOR's theory treated each cleavage independently of the others. But the introduction of the concepts of major and minor alpha-cleavages destroyed this independence and forced revisions on the program. Since the expert measured the relative abundance of minor alpha-cleavage peaks in terms of the major peaks, it was essential to calculate the abundance of the major alpha-cleavage peaks first. The technique for handling this was to introduce a switch indicating whether the major alpha-cleavage had been encountered yet (with appropriate tests and settings in various places). The underlying reason for using this technique rather than another was to plug the hole as quickly as possible (and as a corollary to fix things with a minimum of reprogramming).

In the planning stage, the anticipated form of a rule was a list of peaks at characteristic mass points (where these could be relative to the molecular weight). But in order to identify alpha-cleavage peaks in ketones the program needed to find a pair of peaks at masses $x1$ and $x2$ which satisfied the relation $x1+x2=$molecular weight$+28$. So the program was extended in two ways to account for this: first, a LISP function was allowed to stand in place of an $x,y$ pair as an acceptable rule form in the table of planning rules and, second, a function was added to the set of available rules. The function looks for $n$ peaks $x1, \ldots, xn$ which sum to the molecular weight plus $k$, where $n$ and $k$ are different for different functional groups ($n=2$, $k=+28$ for ketones).

The second fundamental difficulty in this whole process has come after the additional programming was completed to take care of new concepts, when we are in a position to try out the programs on real data. Typically these first trials uncover counter-examples to the initial set of rules: we have often been surprised at the low quality of the inferences on this first pass. For example, we quickly found that the theoretical ketone rules did not always hold for methyl ketones, i.e., for structures containing the radical $CH_3—C=O$. The alpha-cleavage on the methyl side produced a much weaker peak than was originally expected, and methyl ketones often failed to show significant McLafferty rearrangement peaks, contrary to expectations. Thus it was necessary to alter the original rule that both alpha-cleavage peaks for ketones must be high peaks, to allow for the virtual absence of the peak corresponding to loss of the methyl radical. Also, because of the methyl case it was necessary to alter the conditions which determined the strength of McLafferty rearrangement peaks in ketones.

Experimental mass spectra often contain peaks which the theory either cannot account for or would have predicted were absent and the spectra often fail to show peaks where the theory predicts there should be some.

Because of this, the first attempts to use almost strictly theoretical rules in the context of real data often reveal counter-examples to the rules. A theoretical chemist, however, wants to sweep away these discrepancies – we have heard such comments as 'typing error', 'recording error', 'impure sample', 'insensitive instrument', 'uncareful operation of the instrument', and so on. In tracking down the source of the discrepancies we first check the original data to see that the computer has looked at what we wanted it to. Occasionally, our friends have even re-run samples in their own laboratory to check the reliability of the data. But our limited experience indicates that the data are seldom in error: it is the theory that needs more work.

From the chemists' point of view, the dialog process is also helpful for discovering gaps in the theory. Only when they started stating their theoretical rules as precisely as the computer program demands did they realize how little their theory of mass spectroscopy says about some simple classes of molecules. For example, when considering the class of amines, a chemist wrote out 30 interesting amine superatoms* which he believed exhausted the possibilities. A program which was developed later to generate superatoms convinced him there were, in fact, 31 possibilities. Even Professor Carl Djerassi, author of a comprehensive book on mass spectroscopy, terms his exposition 'woefully inadequate' in places because of the gaps discovered in the computer model. (Research is underway to fill these gaps.)

Making a false start is the third type of problem, which is usually discovered only after a few iterations of examining new results and patching rules. Because this requires backtracking and reprogramming, it is painful to realize that some early decisions were bad in light of subsequent developments. We have had courage enough to label only a few of our decisions as false starts. For example, in the planning phase we quickly got into trouble with identification rules for ether subgraphs by over-specifying the subgraphs. We had successfully attacked a previous class of molecules (ketones) by dividing the class into an elaborate hierarchy of subgraphs, each with its own set of identifying rules. But this approach was not transferable to the new class, apparently because the mass spectrometry of ethers follows a different pattern. By the time we had defined rules for C—O—C, $CH_2$—O—$CH_2$, $CH_3$—O—$CH_2$, and $CH_3$—$CH_2$—O—$CH_2$ we were no longer able to make sound inferences. Thus it was necessary to start at the beginning and define a less hierarchical, broader, and smaller set of ether subgraphs.

Typically it has taken weeks of interaction with a chemist at a console to proceed past the first two difficulties, never knowing whether we were making a false start. However, the iterative process itself is not finished when

* As readers of the *Machine Intelligence 4* description of HEURISTIC DENDRAL will remember, a superatom is a structural fragment which is treated as a single unit. For example, when given the amine superatom —$CH_2$—NH—$CH_3$, the program will use this structure as an atomic element without considering any structural variants of it such as —$CH_2$—$CH_2$—$NH_2$. Thus several atoms in the graph can be replaced by a single superatom, at a considerable saving for the STRUCTURE GENERATOR.

a set of rules is found which seems to 'do the right thing'. Because of the number and the complexity of the subgraphs we often run into trouble because we do not have the patience to grind out the consequences of the inferences which the planning phase makes. For many examples of spectra our rules excluded so many subgraphs that, even though the program was properly instructed to put a particular superatom into every structure generated, it could not generate any structures at all. In these cases we have had to weaken the identifying rules still more – with the result that we often let in incorrect classes of molecules to insure that we never excluded the correct ones.

The end of the iterative process to establish planning rules for a class of molecules comes when we have a set of rules which correctly identifies substructures contained in all available examples of mass spectra for that class, e.g., for all acyclic ethers. Similarly, the end of the process to establish the deductive rules comes when the chemists satisfy themselves that the predicted mass spectra agree in significant respects with the published mass spectra of a broad range of examples.

It should be mentioned that we recognize the need to clear up the bottleneck of getting new information into the computer. Here, as elsewhere, many alternative designs are open to us. For instance, we could get rid of the 'middle man' in the information transfer by educating a programmer in mass spectroscopy or by educating a chemist in LISP. Or we could replace the middle man with a program designed to perform the same function as B (the layman/programmer) in the dialog above. In effect, we have been moving slowly in all three of these directions at once. But what we would most like to pursue is the design of a program to elicit information from an expert who is not also a programmer. (This seems especially attractive to the real-life B, needless to say.)

In many areas of science – especially the rapidly expanding frontier areas – the rules which will someday be incorporated into a unified theory exist only in an uncodified morass of recent papers and unpublished notes, and in the heads of researchers on the frontier. Because of the number and complexity of the rules, they are easy to forget, especially so in a collection that is messy. The process of codifying this collection is thus both tedious and important. For this reason automation of the dialog is of general interest: B is not the only one who stands to gain.

Because B's function is more than translating from chemical language to LISP, the program must be more than a compiler. Writing the compiler and, before that, designing a rich enough chemical language, seem unavoidable in the general problem. B does even more than an interactive compiler which asks for clarifications of statements. B also asks questions to fill in gaps, he uses analogies (and occasionally even sees one), he constructs possible counter-examples, and he puts new information into all parts of the system which can use it.

Each one of these additional functions adds another level of complexity to the problem of automating the dialog. Yet the language of any particular science may be sufficiently formal and constrained that the whole problem is still tractable. In our task area these problems may be as well in hand as anywhere. The next few remarks will briefly show how they are manifested in the DENDRAL system. B's experience has been that the expert can easily overlook a logical possibility, for example, one of all possible permutations of carbon, hydrogen, and nitrogen atoms in a terminal radical. Because of the exhaustive STRUCTURE GENERATOR within the program – in fact, at the heart of the program – it is possible to enumerate all structures within a specified class. Thus it is possible to use a program to check for gaps in any list of structures provided by a chemist. An important but non-trivial problem, then, is finding heuristics which will select 'interesting' missing structures, that is, structures the chemist would like to know he missed.

Frequently the discussion of a new functional group will call in analogies with what has been discussed before. 'Amines are like ethers', was one specific remark that B had to make sense of; a smart program should at least know what questions to ask to make sense of the analogy. It will take a much smarter program to recognize these analogies itself. The point is that the dialog will move much faster if the program can at least use analogical information.

Constructing counter-examples may often require a thorough understanding of the theory. But B has been of some help to A even though he has only a little knowledge of mass spectrometry. The dialog program might easily watch to see what kinds of cases the expert needs to patch up. This strategy now leads B to ask 'But what about the methyl case?' for every set of rules that doesn't explicitly consider methyls. And, surprisingly, this reminder is often helpful.

Finally, the 'middle man' in the process is sometimes expected to put pieces of theory in appropriate places of the program, and sometimes to shift information from one place to another. The difficulty here, of course, is that different parts of the program require different representations of the knowledge: the planning phase is written in terms of transforming spectral lines into structural pieces while the PREDICTOR is written for transforming structural pieces into spectral lines. As the theory becomes more complex and as the representations diverge, it becomes more difficult to assess the consistency of the different representations. Human intelligence now decides the questions of where to put new information, how to represent it, and how to make it consistent with other statements. These questions will be discussed in the next section. Let it suffice here to say that a dialog routine cannot be blind to how and where the information will be used.

In sum, eliciting a theory from an expert is a tedious process that is worth automating. It has been our key to the wealth of knowledge not yet accessible in textbook packages. And it has benefited the scientist since it provides a

means of codifying a loose collection of empirical generalizations into a theory. Automating half of the information transfer should add confidence in results as well as speed to the process. Our concern is not so much building a program which teaches itself mass spectrometry as building one which has the capacity to be taught.

## 3. GENERAL PROBLEMS OF DESIGN, SEARCH, AND REPRESENTATION

Behind the discussion of the information transfer process is the unquestioned assumption that the performance of the HEURISTIC DENDRAL system depends critically on the amount of knowledge it has about mass spectrometry. Thus it is necessary to be able to add more and more theory to the program in the easiest possible way – through some such process as the dialog just discussed.

In addition to the *amount* of information the system has, the performance of the system also depends upon how and when that information is used during the problem solving process. Writing a program to use the theory of mass spectrometry presupposes making a choice about how and where to reference the theory. That is, it presupposes choosing one design for the system over others, choosing an efficient search strategy, and choosing appropriate representations for the theory.

In systems science the best design is the one which maximizes the stated objective function. Thus an objective function provides a measure of performance for any design of the system, when the function is available. Unfortunately, there is no epistemological theory which allows us to define one objective function and alter the design of HEURISTIC DENDRAL systematically to bring its level of performance closer and closer to the objective. Our criteria for evaluating the performance of the system are admittedly intuitive: we say that a design, manifested in a computer program, is better the less time the program takes, the more compact the program is, and the more problems it can solve. (Also, an intuitive concept of elegance may lie below the performance measure as a means of judging between programs which seem to perform equally well with respect to the other measures.)

The larger problem of designing the system efficiently cannot be ignored by anyone writing complex computer programs. But design questions involve more than just programming considerations. As with other large programs, HEURISTIC DENDRAL is broken into segments, with each segment expected to contribute to the solution of the whole problem in such a way that the performance of the entire system is efficient over a broad class of problems. If we were given just one design to implement on a computer, the questions would be questions of coding and running efficiency. But we have been forced to realize that our first choice of design was not the best one after all, that we must concern ourselves with choosing among all possible designs for systems which perform the same task.

Apart from the fact that no completely satisfactory measure of performance is forthcoming, there remains a problem of relating the performance of the components of the system with the performance of the whole system. In some systems the parts are completely independent; thus maximizing the performance of each part results in maximizing the performance of the whole system. But in the case of this program, as in other complex systems, the components are so interrelated that the best total system is different from a collection of the 'best' independent parts, because the measure of each part's contribution must bring in the goals of the other parts.

The problem of where to put theoretical knowledge into the system is one aspect of the design problem which is of particular interest to us. There are several components of this system which might profit from access to the theory of mass spectrometry if we chose to represent the theory suitably for each part. But we must balance benefits to a part of the system against cost to the whole system. For example, the addition of theory to the planning stage increases its contribution, and benefits the total system, as mentioned earlier, with only a small increase in program space. Approximately three-quarters of a second spent scanning the data to make a rough plan resulted in the saving of ten or more minutes of computer time in the successive stages of the program. By our intuitive measures of good performance, we took that as an improvement, as long as the reliability of the later parts was not undermined by hasty planning. However, in the case where we gave the planning program identifying conditions for thirty amine subgraphs we did run into serious time trouble, but not where we expected it. We expected trouble to show up in a slow-down of the planning program, when it showed up at all. But in the amine case, the slow-down came in the GENERATOR because of the number of generation constraints added by the planning program: three to eight subgraphs, typically, would be added to GOODLIST and the rest of the thirty subgraphs added to BADLIST. The generator just had too much information to process. Our solution was to reduce the number of BADLIST additions, since (a) this was the major source of trouble in the GENERATOR, and (b) we could be assured that we never deleted correct answers this way. Although we did increase the number of wrong answers from the GENERATOR, they would be ruled out when the predictive theory of mass spectrometry was applied later.

Woven through the pattern of alternative designs for the system are alternative search strategies which are available to the system designers. In the designs actually programmed, the over-all search strategy has been to define a subspace, generate all hypotheses in that subspace, and test each. But at least two different strategies are available to the program: A, test each node in the subspace during generation (i.e., test partial hypotheses), and B, generate one candidate hypothesis then use a GPS-like difference-reducing strategy to generate better hypotheses. Both of these alternatives will be

discussed as a means of bringing out some of our design problems, and as a weak means of justifying the strategy used in the program.

The alternative strategy, A, has, in fact, been tried in one version of the program with only incomplete results so far. In the simplest application of this strategy, the generator consults the deductive theory at each node in the generation tree to determine whether the data indicate that an unproductive branch has just been initiated. That is, the theory is consulted to determine which partial hypotheses are not worth expanding. Unproductive branches are pruned, another node is added to each partial hypothesis, and the test is repeated. For example, part way down the search tree one branch (partial hypothesis) might be an oxygen atom with unbranched carbon atoms on either side (—$CH_2$—O—$CH_2$—), and the next move for the GENERATOR might be to attach a terminal carbon to one of the carbons resulting in the partial hypothesis —$CH_2$—O—$CH_2$—$CH_3$. Consulting the theory will tell the GENERATOR that this is a fruitful branch only if the data contains peaks at 59 and the molecular weight minus 15 (M−15), otherwise the branch would be pruned at this point. Because of the large number of nodes in the unconstrained hypothesis space, it was quickly evident that this strategy could be applied in this simple way only when the planning phase had indicated a relatively small subspace.

One reason why this alternative strategy, A, will not work well in this task area is that the theory of mass spectrometry in the program, as in the heads of chemists, is highly context-dependent. The theory can say very little about the behavior of isolated atoms or small groups of atoms in the mass spectrometer without knowing their environment in the molecule. An ethyl group, ($CH_3$—$CH_2$—) for instance, usually produces some small peaks in the spectrum at masses 29 and M−29, but when it is adjacent to a keto radical (C=O) it will produce strong M−29 and 29 peaks (depending, of course, on the structure attached to the other side of the keto radical). When an ethyl is attached to an oxygen in an ether ($CH_3$—$CH_2$—O—), on the other hand, the theory predicts a peak at M−15 but not at M−29, and no peak at mass 29. More importantly, the theory can say very little about pieces of structure which do not contain at least one terminus. But the canons of structure generation begin with a node at the *center* of the structure, working down toward the termini. The theory can say almost nothing, for example, about a chain of carbon atoms in the center of a molecule without knowing what is at the ends of the chain. In short, it must know the context.

For any class of problems where it is difficult to validate partial hypotheses, the node-by-node search strategy is not the best of alternatives. The current design with no theory used inside the GENERATOR (and thus no node-by-node testing) is superior to the node-by-node test strategy with respect to confidence, and almost certainly with respect to time.* Only after branches

* Those familiar with earlier versions of the HEURISTIC DENDRAL system may recall that a rough deductive test was once applied at each node, using what we called the

of the search tree terminate, i.e., when complete chemical structures are generated, can the theory be called with confidence, for only then is the context of each piece of the molecule completely determined. But the intermediate calls to the theory will then either be incorrect or a waste of time.

Adding one or both of two levels of complexity to the node-by-node testing strategy, A, however, may make it competitive with the current test-at-the-end strategy for our problem. First, we can add some meta-theory to the testing routine or, second, we can reorganize the GENERATOR to make the theoretically significant nodes come at the top of the generation tree.

(A1) Adding meta-theory to the testing routine is relatively simple since it is possible to say *a priori* that the theory is uninformative or perhaps misleading on certain classes of partial structures. Thus the first test on a partial hypothesis is to determine whether the theory can say anything about it – whether this partial hypothesis warrants the expense of calling the full deductive theory. In this way, the number of calls to the theory is considerably reduced. The moral seems to be that a little meta-theory goes a long way.

(A2) Reorganizing the STRUCTURE GENERATOR is a second way to maximize the pruning ability of the deductive theory in node-by-node checking. As mentioned earlier, the canons of generation initiate each structure at the center so that generation is from the center out to the termini. So in most cases near the beginning of the generation process the testing routine provides no information which allows pruning. Testing begins to pay off only after termination of one of the branches of the partial structure. By starting the generator at a terminal atom (instead of at a central atom) the deductive theory could often prune very effectively at the top of the search tree where it is most desirable. One reason why we have not pursued this strategy, however, is that we now have no way to decide which end of the structure will make the most informative terminal radicals. In those cases where the oxygen of an ether molecule, for example, lies close to one end and far from the others, as in $CH_3—CH_2—O—CH_2—CH_2—CH_2—CH_2—CH_3$, the savings would be positive for the terminal atom near the oxygen, but negative for the other choice.

(B) Another completely different search strategy which the program might have used is a GPS-like difference-reducing strategy, mentioned above as the

---

'zero-order theory of mass spectrometry'. The simplicity of the tests was both the beauty and the downfall of the zero-order theory. Because it was not a complex theory, the test was very cheap, and thus could be applied to every node. But it was such an oversimplified theory that it very often returned incorrect answers to the tests. We have not abandoned hope of finding heuristics which indicate circumstances under which cheap tests are reliable. We are also asking ourselves how to call the complex theory efficiently, as described in (A1) and (A2) of the text to follow. Just asking questions of this sort, and asking how to incorporate their answers (if found) into the LISP program, incidentally, have led to a successful reformulation of the program. The new code, designed to allow reference to a more general theory than the zero-order theory, runs about twice as fast with about three-fourths the number of instructions.

second alternative to the current test-at-the-end strategy. The STRUCTURE GENERATOR could construct any molecule as an initial hypothesis – preferably within some constraints set by a smart planning program – and the rest of the time would be spent finding differences between the predicted and actual mass spectra and then reducing those differences by changing the structure of the candidate. Chemists find this suggestion attractive because they use somewhat the same strategy in analysing mass spectra, since they are without the benefit of an exhaustive GENERATOR. However, they have been unable to articulate a measure of progress toward the goal or a description of the process of finding relevant differences.

Another reason the GPS strategy does not fit our problem is that unless the program keeps a precise record of hypotheses already considered, it will have trouble avoiding loops. The structural changes would be made in pieces, in response to the salient differences at any level. Thus it is quite likely that a sequence of changes, each meant to reduce one of a set of differences, would soon be in a loop because changing one piece of structure to reduce the one difference might well introduce other differences in the mass spectra.

Another important reason why the GPS framework is not suited for this problem is that the chemist does not necessarily work incrementally toward the goal, as GPS does. He may add a feature to the hypothesis at one stage which seems to introduce more differences than it reduces. And then, because of that, he may finish the problem in a few swift strokes. For example, shifting the position of a functional group in a candidate molecule may explain some puzzling spectral lines but introduce puzzles about other lines that the previous structure had explained. This strategy of temporarily retreating from the goal, so to speak, is also common in synthetic chemistry and in theorem proving. In both cases, expressions (or molecules) are introduced at one stage which are more complex than the one at the previous step, because the remainder of the problem-solving activity is thus simplified. In other words, there are certain problems for which step-by-step movement toward a goal is not the best strategy; mass spectrum analysis appears to be one of them.

Although the two alternative search strategies A and B introduce new difficulties, modifying the current strategy may well improve the program without adding serious problems. One extreme is to use a powerful enough theory in the planning stage to produce only a single unambiguous hypothesis: that is, plan the hypothesis generation process so carefully in light of data and theory that just one structure meets the constraints. This means adding much more new theory to the planning program. The planning stage now has a table of interesting and relatively common subgraphs each coupled with a set of identifying conditions. Pieces of structure for which the theory has too little context to identify their presence or absence are left out of the table entirely. The rest of the table is organized hierarchically.

However, using a powerful enough theory requires enumerating whole

molecules (because the theory cannot be applied unambiguously to pieces of molecules out of the total context), resulting in an enumeration which would be far too large to catalog or search. On the other hand, enumerating subgraphs – or pieces of molecules – in a much more manageable list leaves ambiguities in the ways the pieces can be put together in a complete molecule. That is, if we want to plan carefully enough to isolate exactly one structure for any number of atoms, the entries in the table must specify the total context for each piece of structure. In this case the planning program must do a table look-up on spectrum-molecule pairs, obviating the need for the STRUCTURE GENERATOR or PREDICTOR at all. (Much work in the application of computers to analytic chemistry has this flavor.) Cataloging anything less than whole structures will result in looser constraints, since some contextual information must be omitted, and thus will result in generating more than one whole structure in those cases where there is more than one way to put the identified pieces together.

While we cannot rigorously justify our design decisions, and in particular our decision to use one search strategy over another, we have been able to explore some alternative designs. Perhaps more importantly, we have found that the HEURISTIC DENDRAL system is fertile ground for exploring these general problems.

Another class of problems which the system forces on us has been called 'The Representation Problem'. There appear to be several problems under this rubric: choosing a convenient representation for the theory, deciding when to proliferate representations, deciding when two representations are consistent, and switching from one representation to another. None of these appears to warrant the title 'the problem of representation' any more than the others; they all require solution in any system which admits any of them.

Initially, the only theory of mass spectrometry of any complexity in the program was the deductive theory in the PREDICTOR. The most crucial aspect of the representation problem at that time – and probably the only aspect we saw – was choosing a convenient representation. And then, also, we held a simplistic view of what made a representation convenient. We meant, roughly, a representation that was easy to code and write programs for.

Since then it has become obvious that convenience is also conditional on the persons adding statements to the theory, as discussed in the second section. For the sake of communicating with the expert, for example, it may be necessary to cast the theory in terms of bonds and atoms at the level of the dialog, but then transfer those statements to a representation in terms of electron clouds and charge localization for the efficient operation of the program. That is, there may be a need for two representations even though there is only one theory. With only one representation it is very possible that either communication with the expert or execution of the program will become cumbersome. On the other hand, separating the internal representation from the one which is convenient for communication makes it more

difficult to find mistakes in the program and to explain mistakes to the expert who must ultimately correct them.

With the addition of planning to the program, it was expedient to introduce a new representation of mass spectrometry theory which could be read easily by the planning program. Even though all of the information was already in the PREDICTOR's theory, it was not in a form which could be easily used for planning. For example, the PREDICTOR's theory indicates that a pair of peaks (at least one of which is high) will appear in the mass spectra of ketones as a result of breaks on either side of the keto (C=O) group. Thus, because of the appearance of C=O (mass 28) in each resulting fragment, the peaks will add up to the molecular weight plus 28. The theory in the planning program also knows this, but it uses the theory in reverse. The planning program looks for a pair of peaks in the data (at least one of which is high) which sum to M+28 as a necessary condition for the appearance of the keto group. That is, the PREDICTOR uses structural information to infer pieces of the bar graph, while the planning program uses bar graph information to infer pieces of structure.

Duplication of information may be the preferred means to processing efficiency, even at an obvious cost in space, as it almost certainly is in this case where conditionals are read left to right in the prediction (deductive) phase and re-representations are read the other way in the planning phase. Even more critical than the space *versus* processing time question, though, is the question of consistency. The system has no way of checking its own theories for inconsistencies. Worrying about the consistency of different representations of the theory may be considered a waste of time, but we see this as a serious issue because of the complexity of the body of knowledge about mass spectrometry. We even have to be careful now with the internal consistency of each representation, because of complexity. For example, the rules of the planning program have occasionally put a subgraph on GOODLIST and a more general form of that subgraph on BADLIST: to say something like 'this is an ethyl ketone but it is not a ketone'. Our solution to this particular problem avoids the consistency issue by allowing the planning program to check only as far as the first 'no' answer in the family tree. In general, however, because of the complexity of the theory, we are not confident that the programs are internally consistent, let alone consistent with each other.

The consistency problem would evaporate if there were just one representation of the theory which could be read by all parts of the system which use the theory. But it may be unreasonable to expect to find one representation which is suitable for all purposes. Another solution to the consistency question is to add either (1) a program which can read both representations of the theory to check for inconsistencies, or (2) a different representation to which modifications will be made and a program which writes the other two representations from the third after each set of changes. At the least, the consistency of the whole system can be checked empirically by running

examples. It may well be that this is also the best that can be done; there may be no logical proof of consistency for this vaguely stated body of knowledge. In any case, the system should be designed in such a way that the opportunities for introducing inconsistencies are minimized.

If the consistency problem is dismissed by disposing of all but one representation of the theory in a system, then the problems of representation become vacuous for that system. When different representations of the same body of knowledge remain, however, it is possible that switching from one to another inside the program will be desirable. In this system, for instance, it would be very desirable to be able to move information automatically from the PREDICTOR's complex theory of mass spectrometry to the planning program's theory. The convenience and consistency questions just mentioned have directed attention to the benefits of switching representations. There are at least two ways of carrying it out here. First, and more generally, if the theory were suitably represented, for example in a table, a program could conceivably move pieces of information from one place to another making appropriate transformations on the way. This is very difficult for any complex body of knowledge, though, since it is difficult to put it into a perspicuous form and to write a program which can interpret it. The less general way of moving mass spectrometry theory from PREDICTOR to PRELIMINARY INFERENCE MAKER also appears slightly less difficult. In effect, the program can be asked to perform a 'Gedanken experiment', i.e., to pose questions about mass spectrometry and answer them itself without outside help. The program already has almost all the necessary equipment for such an experiment. The major power of the idea is that there is already a systematic STRUCTURE GENERATOR for producing the instances of molecules of any class, for example, all methyl ketones. Moreover, the STRUCTURE GENERATOR can also produce the exemplars, or superatoms, which define the class. The PREDICTOR tells what happens to each particular molecule in the mass spectrometer. All that remains is a program to classify the predicted mass spectra and find the common spectral features. These features are just what the planning program needs to identify the class. In this way the PREDICTOR's theory is transferable to the planning program.

Much of our current effort is directed to just these points: set up one central theory which the expert modifies and automatically move the new information to appropriate places. This effort requires much reprogramming, some of which is described in the next part of the paper, it requires improving the communication with experts as described in the second part, and it requires answering the critical design questions just discussed.

## 4. TABLE DRIVEN PROGRAMS AND RECENT PROGRAMMING CHANGES IN HEURISTIC DENDRAL

Parts 2 and 3 have discussed the problems of obtaining and representing scientific theories for a computer program. Designing the actual computer

programs to access the theory is another problem, which, fortunately, seems easier to solve than the others. The general programming approach, adopted after several trials, is summed up in the phrase 'table driven program'. The idea (which is worked out in detail in Donald Waterman's program to learn the heuristics of draw poker) is to separate the theory from the program which works with the theory by putting specific items of theory on lists and in global variables. Changing the theory, then, involves little actual reprogramming. This allows experiments to be carried out with different versions of the theory, a very useful feature when dealing with a subject which is as uncodified as mass spectrometry.

A. The first of the DENDRAL programs to be written as a table driven program was the planning program (PRELIMINARY INFERENCE MAKER) which bases most of its operation on a list of names and their associated properties. The planner has a list of functional groups and subgroups arranged in family hierarchies, e.g., (A) ketone, (A1) methyl-ketone, (A2) ethyl-ketone, etc. Associated with each group and subgroup is a set of identifying conditions. The program picks the first main functional group on its list and checks its identifying conditions against the given mass spectrum, e.g., for the subgroup $C_2H_5—C{=}O—CH_2—C—CH$, we have x1+x2=M+28 (alpha cleavage) and 72 high (McLafferty rearrangement). If any condition fails to be satisfied, the group and all its subgroups are ruled out – their structures are put on BADLIST. If all conditions are satisfied, the structure of this group is put on GOODLIST – a list of preferred subgraphs. Then subgroups will be checked in a similar way. All groups known to the program are thus considered either explicitly or implicitly. Modifying either the list of subgroups or their properties will drastically affect the behavior of the program. Yet all the theory of mass spectrometry in this program is contained in one or the other place.

B. The STRUCTURE GENERATOR program has been table driven to a small extent; in particular, three lists, ORDERLIST, BADLIST, and GOODLIST, function as tables which determine the structures which will be generated and their order. ORDERLIST contains a list of all chemical atoms which the program can use. Each atom has properties such as valence, weight, symmetries, etc. Removing an atom from ORDERLIST effectively removes it from the domain of the STRUCTURE GENERATOR. The relative order of atoms on ORDERLIST determines, to a small extent, the order of structures in the output list. BADLIST is another table which controls output of the STRUCTURE GENERATOR. If BADLIST is NIL, all topologically possible structures will appear. Otherwise, any structure containing one of the BADLIST subgraphs is pruned from the generation tree as soon as the BADLIST item first appears. This does not change the generating sequence, but rather eliminates structures from the unfiltered output list. GOODLIST serves two purposes: it can determine the order in which structures are generated and it can limit generation to a specified class of structures. Those

structures containing preferred substructures present on GOODLIST will be generated first, while structures containing none of the preferred substructures will be generated last or not at all if generation is to be limited.

One of the basic problems inherent in the STRUCTURE GENERATOR, however, has been its rigid insistence on following the canons of DENDRAL order as they existed four years ago when the program was written. These canons specified the canonical form of a structure, and thus the implicit generating sequence, by stating the following rules:

> Count, degree, apical node, and afferent link are the attributes in decreasing order of importance. 1 is lowest count, increasing integer values are higher. The value of apical nodes follows ORDERLIST, usually C<N<O<P<S, with superatoms added at the end. 1 is minimum degree, the highest degree is the maximum valence of all the atoms on ORDERLIST. 1 is the minimum link, 3 is the highest link.

These specifications were programmed into the STRUCTURE GENERATOR LISP code in such a widespread way that changing even the allowable ranges for attributes (let alone trying to change the order of attributes) required many separate small programming changes. Thus, it was difficult to determine all the places to change the code whenever even slight variations of generating strategy were desired.

The rigidity of the program in this respect made it very difficult to change the generating order for structures. It had occasionally been suggested that non-branching structures should be given preference, but such a suggestion was difficult to implement with the former STRUCTURE GENERATOR. This problem has now been overcome by a substantial reworking of the STRUCTURE GENERATOR program. A basic change in operating procedure made this possible. This is the evaluation, at each level of structure generation where a node and link are picked and recursion is about to occur, of each choice of partial structure, and a consequent ordering of choices in a plan list. The program follows the DENDRAL canons through all values of node, link, and degree, and makes a plan list of all possible ways to add the next node to the emerging structure. It orders these plans according to plausibility scores calculated by a single LISP function. Some plans may be eliminated because of 'implausibility'. Only then does the recursion take place, operating according to a single one of these plans, and then the process is repeated for the next node to be added to the emerging structure.

The result of this reorganization is a tremendous simplification of the generating algorithm. Instead of having six functions to generate the complete list of structures, two are now sufficient. Of the six functions (GENRAD, MAKERADS, UPRAD, UPLINKNODE, UPCOMPNODE, and UPDEGNODE), only two remain. The other four, whose jobs were to change a single structure, have disappeared. Previously GENRAD constructed the single 'lowest' canonical structure which could be made from an empirical formula. This

structure had to be 'incremented' by UPRAD many times in order to obtain the entire output list. The current version of GENRAD does all this for itself and returns a list of structures as its answer. Incidentally, this reduced the size of the STRUCTURE GENERATOR by about 25 per cent, a substantial saving; and cut execution time about in half.

This reorganization quickly caused us to notice that it would now be relatively easy to make the GENERATOR into an almost completely table driven program, by putting the DENDRAL canons (attributes and their values) on a global list. This is now possible because the canons are mainly invoked by the function GENRAD and only a few other utility functions. The new idea is to form a global list of the form

((LINK 1 2 3) (NODE C N O) (DEGREE 1 2 3 4))

which will be accessed during the process of making plans about how to enlarge the structure that is being built. In the example of the list above, the link is the least important attribute, and 1 is its least value; thus LINK=1 is always the first thing to be tried in generating structures. If, for some reason, it was felt that highly branched structures with heteroatoms (non-carbon atoms) near the center of the structure were the most likely, the revised form of this global list might appear as

((DEGREE 4 3 2 1) (NODE O N C) (LINK 1 2 3))

or if desired, unbranched structures could be eliminated entirely by revising the list as

((DEGREE 4 3 2) (NODE O N C) (LINK 1 2 3)).

This table driven program will have great use whenever some data or some chemist's special application indicate that structure generation should be limited to a very specialized class of structures.

C. The PREDICTOR program is currently being revised in the form of a table driven program. This will permit a great simplification in the process of adding new chemical theory, as well as making the program easier to understand and correct. One large part of the effort of re-programming the PREDICTOR is in switching representations of structures. Previously, three different representations of structures had existed there: the list notation which is characteristic of the STRUCTURE GENERATOR (and the graph matching algorithm which the PREDICTOR inherited), a variant of the list notation with unique numbers assigned to the nodes of the graph, and a connection list representation of structures. In the connection list representation the unique names of nodes are stored as global LISP atoms with properties declaring the bonds coming to and from each atom. Five reasons are given for switching to a complete connection list representation in the PREDICTOR.

1. *Keep the legal move generator simple.* The primary motivation for using connection lists was to represent bonds uniquely, because the legal move generator in the PREDICTOR is of the form 'move to the next bond and

decide whether it breaks'. In the connection list, the directedness of acyclic chemical graphs is maintained with separate indicators for the links to other nodes and the one link from another node. The list of links under the 'from' indicator for all nodes, then, is a complete and irredundant list of the links in the graph. The list notation puts bonds and atoms in a hierarchy which makes this process difficult.

2. *Represent fragments uniformly.* Since the PREDICTOR sometimes needs to know what was connected to a new fragment over the broken bond, it was necessary to keep track of the names of the atoms connected by that bond. So connection lists were necessary even when the list structure of a fragment was available. But the connection list representation of structures alone is sufficient for these purposes.

3. *Avoid building up and tearing apart list structures.* All connections are represented once and for all in the connection lists; temporary changes, e.g., the result of removing an atom and breaking a bond, can be represented by temporarily 'pushing down' the appropriate properties. Previously, the PREDICTOR built new list structures for each primary cleavage result and for each result of rearrangements. Then each of these had to be searched for such features as the number of double bonds one or two bonds removed from any atom in the structure. Even the common function of assigning a mass number to a fragment was messy in the list structure, partly because of the branching list structure and partly because the number of implicit hydrogens in the list structure had to be calculated each time.

4. *Speed up graph matching.* In the PREDICTOR, atoms in the list structure needed node numbers in order to specify the places at which a match occurred. This was essential because the secondary processes being modeled in the PREDICTOR affect specific atoms. And the structure of the result is important because the result is itself checked for important subgraphs. Besides adding node numbers to the atoms in the list, it was also essential to put all hydrogen atoms into the list explicitly each time a new fragment was produced. Hydrogen atoms are often important conditions for the occurrence of secondary processes. So the list structure was no longer easy to search with the modified graph matching algorithm of the STRUCTURE GENERATOR. A new algorithm has been written for the connection list representation.

5. *Represent rings in the same notation as trees.* Since circular lists are generally undesirable, a fragment containing a ring could not be represented in the same way as an acyclic fragment. Thus the functions which searched for structural features could not be the same in both cases. Adding one additional property to show the links which make the acyclic structure into a cycle allow us to retain a list of unique bonds. At the same time, we can still find all connections for any atom quickly.

D. Interaction and interdependence of the three sub-programs of HEURISTIC DENDRAL must also be considered when writing and revising these computer programs. Because of the size of the combined programs, it is

more practical to run them separately than to run them together. One supervisor takes care of the interaction by having each subprogram write an output file which is then the input file for the next phase of program operation. The PRELIMINARY INFERENCE MAKER writes the file containing the empirical formula and the GOODLIST and BADLIST to be used by the STRUCTURE GENERATOR. That program, in turn, reads this file, and writes another file containing the single output list of structures which it generates according to the GOODLIST and BADLIST specifications. The PREDICTOR, then, reads this file to obtain its input, and calculates a mass spectrum for each structure in the file. If other tests such as an NMR prediction are to be made on the candidate structures, the supervisor interfaces the appropriate program to these others in the same way.

Although it is painful to rewrite a set of programs as large as those in HEURISTIC DENDRAL, the cost of modifying old programs seems to increase sharply as the number of new ideas increases. The primary motivation for completely rewriting large portions of the LISP code is to increase the program's flexibility. The major emphasis is on separating the chemical theory and heuristics from the rest of the code by putting chemical information into tables.

## 5. CONCLUSION

A few general points of strategy have emerged from the DENDRAL effort for designing a program which will explain pieces of empirical data. With regard to the theoretical knowledge of the task domain in the program, we believe that the following six considerations are important.

1. *Convenient representation.* As discussed in Part 2, the effort of eliciting a theory from an expert can be alleviated by choosing a representation of the theory in which he can converse easily. Although this may not be the best representation for internal processing, our experience has been that it is expeditious to write interface routines between the communication language and the internal one, rather than force the expert to converse in the scheme which suits the machine. This is also preferable to forcing the machine to carry on its problem solving in the framework of the dialog.

2. *Unified theory.* For reasons of consistency, the theory (or set of facts, or axioms) should be collected in one place in the program, with modifications made to this unified collection. This is compatible with having different representations of the theory for different applications, if this is desirable, as long as there are lines of communication between the special representations and the central one. If changes to the theory must be made by hand to every special representation there is a strong possibility that inconsistencies will be introduced between two representations which are intended to be equivalent. Having just one central theory to change from the outside will greatly reduce this possibility.

3. *Planning.* In this program there is no question of the desirability of using

some knowledge of the task domain, mass spectrometry, to construct a plan for hypothesis generation. However, it is not clear how much knowledge to use nor where to use that knowledge. Our one experience with using too much knowledge in the planning stage, when we were using 31 amine (nitrogen-containing) subgraphs, indicated that the planning stage could accommodate a great number of rules; but the GENERATOR was the part which became overburdened. This is only one example of the problems caused by the lack of a meta-theory for system design.

4. *Deductive tests.* Despite the efficacy of the planning stage, there remain ambiguities in the data which cannot easily be resolved prospectively. In task areas such as this one, where testing at each node in the search space is not feasible, deductive tests on the terminal nodes become especially important. The STRUCTURE GENERATOR often constructs several structures consistent with the plan because the planning stage does not reference an exhaustive table of subgraphs. Thus it is necessary to bring in deductive tests upon specific hypotheses to resolve ambiguities. The program deduces consequences of a hypothesis (together with the theory) and looks at the available data for confirmation or disconfirmation.

5. *Generation of planning cues.* Because the theory in the planning phase is part of the more complex theory in the PREDICTOR it should be possible to generate planning cues automatically from the more comprehensive theory. Not only does this relieve (if not remove) the consistency worry, it also opens the possibility of generating cues which might not otherwise have been noticed. Although our own work is barely under way on this problem, the potential benefits are encouraging. In effect the program is asked to look at its theory to say what would happen if structures of a specified class were put in a mass spectrometer. Its answer is a set of identifying conditions for structures of the given class. Hitherto it has been necessary to gather experimental data to answer this question, but here exists the apparatus to generate identifying rules independently of the laboratory data.

6. *Table driven programs.* Separating the theory from the routines which use it facilitates changing the theory to improve it, on the one hand, or to experiment with variations of it, on the other. Although embedding the theory in the program's LISP code increases running efficiency, it seems more desirable, at this point, to increase the program's flexibility. In the STRUCTURE GENERATOR it is useful to be able to change the canons of generation. In the PRELIMINARY INFERENCE MAKER, the identifying rules for groups, as well as the groups themselves, change frequently and so should be easily manipulated. The PREDICTOR's theory also needs modifying frequently, which cannot be done easily if all the theoretical statements are scattered throughout the code. A complex body of knowledge is rarely easy to modify with confidence that the result is accurate and consistent. But the confidence should increase if the statements of the theory are at least separable from the rest of the program.

Although each one of these general points provides direction for future research, each gives rise to numerous problems ranging from global design, search and representation problems to minute programming considerations. We shall know we are making progress in artificial intelligence when we can look back on these problems and wonder why they seemed difficult.

**Acknowledgements**

This research was supported by the Advanced Research Projects Agency (SD-183). We gratefully acknowledge the collaboration of Professor Joshua Lederberg, Mr Allan Delfino, Dr Alan Duffield, Dr Gustav Schroll, and Professor Carl Djerassi.

## REFERENCES

Buchanan, B.G., Sutherland, G.L. & Feigenbaum, E.A. (1969) HEURISTIC DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry. *Machine Intelligence 4* (eds Meltzer, B. & Michie, D.) Edinburgh: Edinburgh University Press (also Stanford Artificial Intelligence Project Memo No. 62).

Churchman, C.W. & Buchanan, B.G. (1969) On the Design of Inductive Systems: Some Philosophical Problems. *British Journal for the Philosophy of Science* **20.**

Duffield, A.M., Robertson, A.V., Djerassi, C., Buchanan, B.G., Sutherland, G.L., Feigenbaum, E.A., & Lederberg, J. (1969) Application of Artificial Intelligence for Chemical Inference II. Interpretation of Low Resolution Mass Spectra of Ketones. *J. Amer. Chem. Soc.*, **91**, 11.

Feigenbaum, E. A. (in press) Artificial Intelligence: Themes in the Second Decade. *Proceedings of the IFIP68 International Congress*, Edinburgh, August, 1968 (also Stanford Artificial Intelligence Project Memo No. 67).

Lederberg, J. (unpublished) DENDRAL-64 A System for Computer Construction, Enumeration and Notation of Organic Molecules as Tree Structures and Cyclic Graphs (reports to NASA).

Lederberg, J. & Feigenbaum, E.A. (1968) Mechanization of Inductive Inference in Organic Chemistry. *Formal Representations for Human Judgment* (ed. Kleinmuntz, B.). New York: Wiley (also Stanford Artificial Intelligence Project Memo No. 54).

Lederberg, J., Sutherland, G.L., Buchanan, B.G., Feigenbaum, E.A., Robertson, A.V., Duffield, A.M. & Djerassi, C. (1969) Application of Artificial Intelligence for Chemical Inference I. The Number of Possible Organic Compounds: Acyclic Structures Containing C.H.O. and N. *J. Amer. Chem. Soc.*, **91**, 11.

Schroll, G., Duffield, A.M., Djerassi, C., Buchanan, B.G., Sutherland, G.L., Feigenbaum, E.A. & Lederberg, J. (in press) Application of Artificial Intelligence for Chemical Inference III. Aliphatic Ethers diagnosed by their Low Resolution Mass Spectra and NMR Data

Sutherland, G. A Family of LISP Programs, to appear in (D. Bobrow, ed.), *LISP Applications* (also Stanford Artificial Intelligence Project Memo No. 80).

Waterman, D.A. Machine Learning of Heuristics. Ph.D. Dissertation (Stanford University Computer Science Department) (also Stanford Artificial Intelligence Project Memo No. 74).

# 15

# Memo Functions, the Graph Traverser, and a Simple Control Situation

David Marsh
Visiting Research Associate to the Dept. of Machine Intelligence and Perception, University of Edinburgh from International Computers Ltd

## INTRODUCTION

Memo functions are a device for speeding up the evaluation of functions on a computer by attaching a small dictionary (or rote) to a function which will hold argument–result pairs obtained from previous evaluations of that function. Before each argument is evaluated the rote is searched and, if the argument already occurs on the rote, the result is found directly, otherwise the function will be evaluated and a new argument–result pair inserted in the rote.

Memo functions were proposed by Michie (1967a, 1968) and initially implemented by Popplestone (1967). This implementation, and the programs described here, were written in the POP-2 programming language (Burstall and Popplestone 1968).

The ideas behind memo functions come from the work by Samuel (1959) into machine learning using the game of checkers. He implemented a rote learning scheme where values associated with board states encountered during the games were stored on magnetic tape and then utilized in the evaluation of subsequent states so as to reduce the size of the search space currently required to achieve the same result. States encountered most frequently were credited with extra life – a process called refreshing – while those hardly used at all were discarded – forgetting.

These techniques of refreshing and forgetting are easily implemented in the memo function apparatus. New entries are placed at the top of the rote. Each time an entry is used it is promoted one place. The rote is limited to a specified size so that the less frequently used items will drop to the bottom and be discarded as new entries are inserted.

Depending on the organization of the rote, memo functions offer not only a speed-up aid, but a fairly general rote-learning device. This paper seeks to investigate their use in each situation. To achieve this generality the structure of the memo apparatus has been extended. Many of the ideas behind this

new formalization are due to Michie. To investigate the rote-learning aspect a very simple control situation has been devised. At the same time comparisons are made between planning by depth first search and by the heuristic search algorithm known as the Graph Traverser (Doran and Michie 1966).

## GENERALIZED MEMO FUNCTIONS

Rather than hold rote entries as argument–result pairs there are many situations in which it would be desirable to group the entries. This extension is in line with a suggestion by McCarthy (personal communication) that the rote look-up should be able to handle predicates, not just individual argument values. For example, if the results of a function such as *log gamma* (which maps from reals to reals) were only required to say two significant figures there would be a whole range of argument values which map onto the same result. These arguments could be grouped together on the rote either explicitly or represented by some form of argument set description. In this case a suitable set description would be a predicate which would test whether a particular argument is within the range of a pair of arguments which map onto the same result. As new arguments are evaluated these ranges can be extended. There could also be an exception list involved with the set description to cater for those arguments which map onto a particular result but do not satisfy the appropriate predicate. *log gamma* is a monotonic function so that for its POP-2 implementation, LOGGAMMA, an exception list would not be required.

Function results would be stored as canonical elements–in the case of LOGGAMMA this would be a real number rounded to the appropriate number of significant figures. Figure 1 is an illustration of how part of the rote of the LOGGAMMA function may appear.

```
[[55.18 . 57.41] . 170.0]
[[47.46 . 50.09] . 140.0]
[[50.10 . 52.53] . 150.0]
[[44.77 . 47.44] . 130.0]
[[52.55 . 55.14] . 160.0]
[[39.75 . 42.12] . 110.0]
[37.85 . 99.0]
[[57.61 . 59.88] . 180.0]
[[42.24 . 44.76] . 120.0]
[[59.96 . 62.41] . 190.0]
[[65.86 . 66.52] . 210.0]
[[38.13 . 39.39] . 100.0]
[36.36 . 94.0]
[[62.77 . 64.45] . 200.0]
```

Figure 1. Sample rote after evaluation of the LOGGAMMA function. The pair of numbers to the left represent the bounds of the range of values whose result is the right-hand number. Where no range has been encountered there is only a single entry (e.g., [37.85 . 99.0])

It is important to realize that this generalization takes place in the rote; the actual function still maps from individual arguments to individual results.

### Implementation of generalized memo functions

The overall organization of the memo function apparatus is shown in figure 2. A memo function is created in POP-2 by a function called NEWMEMO. The parameters of this function are specified by the user and include all the functions needed for controlling the search and structuring the rote. The parameters of NEWMEMO are:

1. The function to be memo-ized.
2. The maximum size of the rote.
3. The number of arguments that the function takes.
4. A function (SEREQUIV) used when searching the rote and deciding whether the result for a particular argument value should come from an appropriate rote entry.
5. A function (UPEQUIV) used when updating the rote with a new argument–result pair, and deciding whether a given entry should be extended. If no entries are to be extended, or UPEQUIV is undefined, new entries will be placed directly on the top of the rote.
6. A function (ROTEUPDATE) used both for forming new entries and for extending already existing entries.
7. A function (YSTAND) which converts the function result to a canonical representation.

To memo-ize the LOGGAMMA function with the argument sets represented either as single entries or as range pairs YSTAND will be a straightforward function for rounding off real numbers, and UPEQUIV will test for equality between the standardized function results and the results entered on the rote. SEREQUIV will test whether an argument equals a rote entry or falls within the range of the bounds entered in the rote. ROTEUPDATE will either enter a new argument singly on the rote, or form a new pair of bounds, or extend the bounds if an argument lies outside them.

If memo-izing the LOGGAMMA function in this way is to speed-up evaluation time it will be partly because new arguments may occur which fall into a previously defined range, so that the result may be taken directly from the rote even though this particular argument has not been evaluated previously. Further gain may come if the function is defined recursively for the rote will be searched at every function call, so that, though the argument may not be found at the top level call of the function, it may be found part way through the evaluation, before the recursion has terminated. However, though the rote is searched at every call it is only updated with the result from the top level. This is to prevent the rote from becoming swamped by a whole sequence of entries resulting from a single argument.

Consider for example a POP-2 FACTORIAL function. If the value of the first argument is 10 and the rote size is 5, the entries in the rote after evaluation

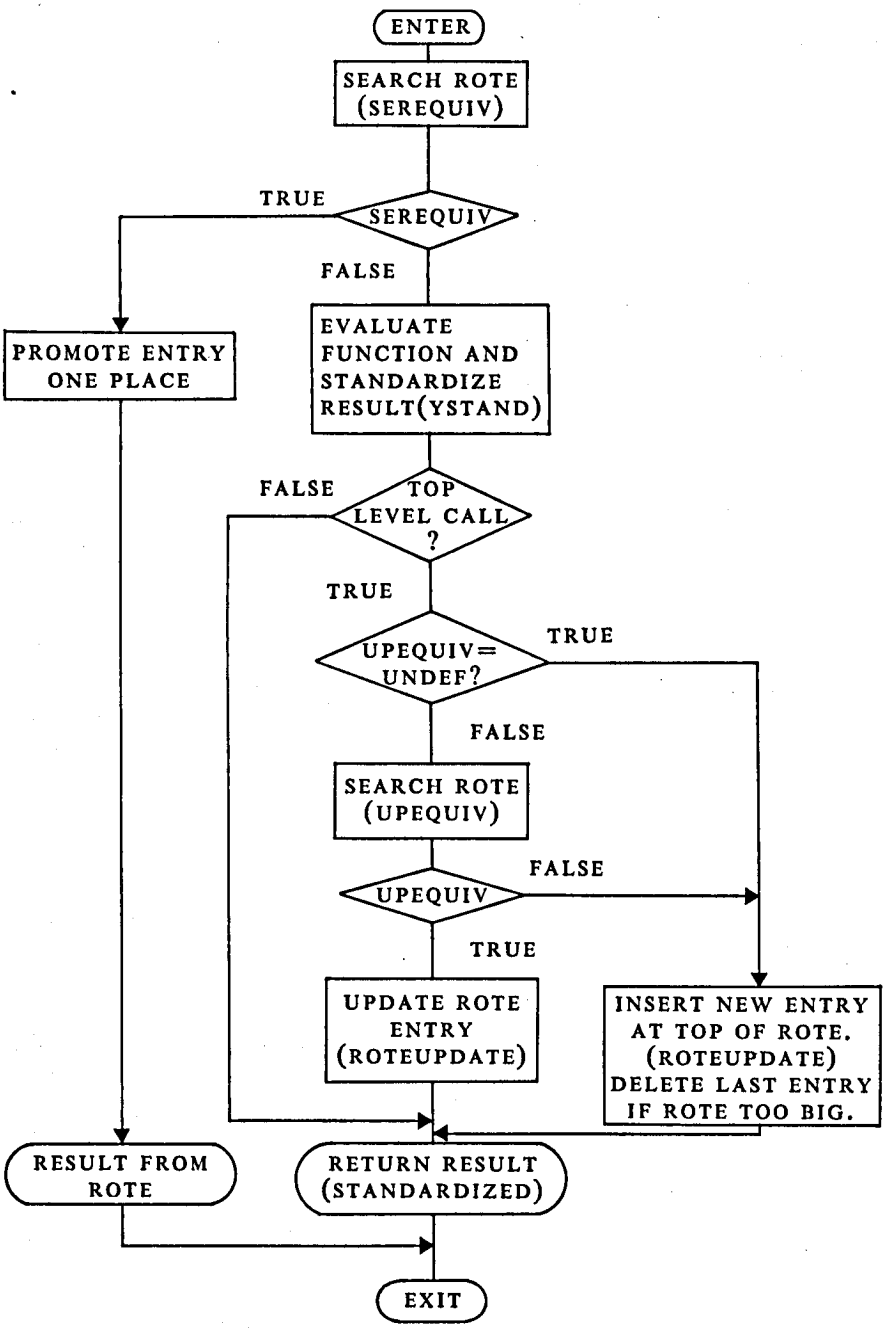



Figure 2. Outline flow-diagram of a memo function

will correspond to the arguments 6, 7, 8, 9, and 10, if the rote is updated at all levels. If the next argument has a value of 50 the rote will contain entries for 46, 47, 48, 49, and 50 after it is evaluated. The entries for 6, 7, 8, and 9 were never used, and that for 10 is now also lost. If the next argument value is 30 the evaluation will have to be complete. On the other hand, if the rote only contained top level entries (i.e., 10 and 50), the evaluation could have terminated early. So while containing arguments from all levels does produce some gain, there is a greater chance of an evaluation terminating early if only top level arguments are entered. However, while this is true for all the functions memo-ized here, it is not necessarily true for all functions.

Unfortunately, in the case of the LOGGAMMA function which takes real numbers as arguments the overheads of the current implementation of memo functions are too great for a pay-off in terms of increased speed. However, the generalized formalization so derived has been of use when using memo functions in a rote-learning context.

## MEMO FUNCTIONS AS A SPEED-UP AID

Memo functions do produce significant speed-ups in the evaluation of integer functions such as LOGFACTORIAL. Experiments have been made with this function, holding the argument–result pairs as individual items, and omitting the updating search of the rote. The LOGFACTORIAL function was chosen because it is easy to define either iteratively or recursively, and because it takes a moderately long time to evaluate.

It was defined (not defined for zero argument for the present purpose) in POP-2 as:

1. Iterative

```
FUNCTION LOGFACTORIAL N;
VARS ANS; 0—>ANS;
L1: IF N=1 THEN ANS EXIT
ANS+LOG(N)—>ANS;
N-1—>N;
GOTO L1
END;
```

2. Recursive

```
FUNCTION LOGFACTORIAL N;
IF N=1 THEN 0 ELSE LOG(N)+LOGFACTORIAL (N-1) CLOSE
END;
```

The experiments were conducted with five different argument populations.

1. Uniform distribution, range 1–100, standard deviation 30.
2. Uniform distribution, range 25–75, standard deviation 15.
3. Normal distribution, mean 50, standard deviation 15.
4. Normal distribution, mean 50, standard deviation 10.
5. Normal distribution, mean 50, standard deviation 5.

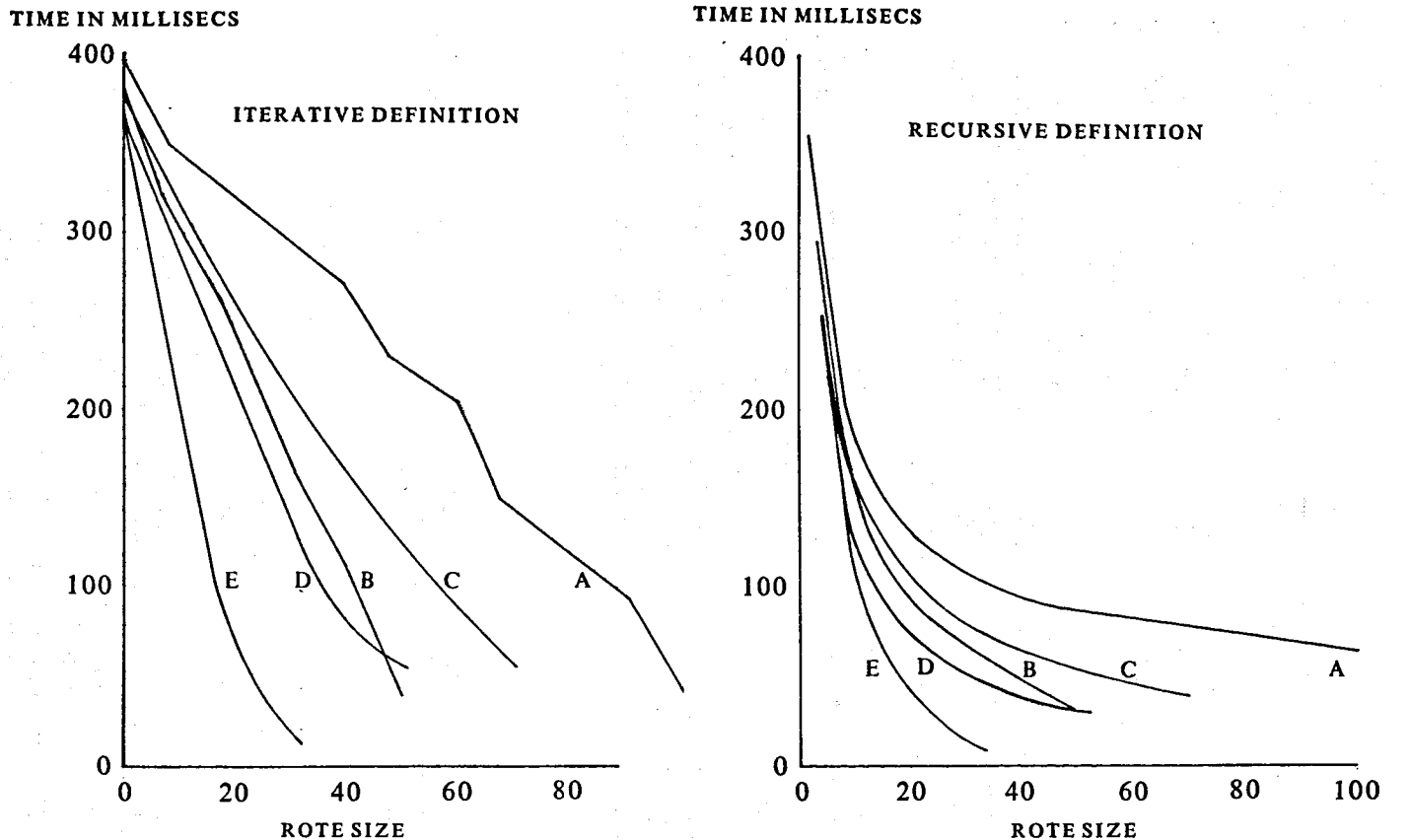



Figure 3. Average time taken by memo-ized LOGFACTORIAL function to evaluate arguments from various distributions A=Uniform, 1-100. B=Uniform, 25-75 C=Normal, mean 50, sd 15. D=Normal, mean 50, sd 10. E=Normal, mean 50, sd 5.

Timings, which were not started until the rote had reached full size, were made with batches of 1000 arguments. The probability of an argument being found in the rote was measured. With the recursively defined variant the average number of function calls was also monitored. The results are shown graphically in figure 3 and summarized in table 1.

| | Time in millisecs | |
|---|---|---|
| | Iterative definition | Recursive definition |
| No rote | 400 | 400 |
| Rote size = $s$ . $d$ of argument population | 300–350 | 100–150 |
| Rote size = $2 \times s$ . $d$ of argument population | 200–250 | 70–120 |
| Rote size = max size of argument population | 30–70 | 30–60 |

Table 1. Summary of average time taken by memo-ized LOGFACTORIAL function.

The observed savings, particularly those obtained with the recursive definition, are significant, especially as no attempt has been made to optimize the search process by the use of techniques such as binary search or hash coding. The search process in fact is sequential, working from the top to the bottom of the rote. This has the advantage of making the promotional and forgetting schemes easy to implement.

Rather than promoting entries just one place on the rote when they are used the effect of placing them at the top, $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ way down the rote was examined. Also examined was the effect of placing new entries not only at the top but $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ way down the rote. None of these variations led to any significant improvement in performance.

#### Explanation of these results

With the argument distributions used here it should be possible to express the speed-up in terms of probability of the argument being found on the rote.

1. With the iterative definition:

$$T_m = P * \tfrac{1}{2} \,.\, D \,.\, S + (1-P) * (T_f + D \,.\, S) + M$$

where

$T_m$ is the time taken by the memo-ized function,
$T_f$ is the time taken by the unmemo-ized function,
$P$ is the probability of an argument being on the rote,
$D$ is the size of the rote,
$S$ is the time taken to examine an entry on the rote,
$M$ is the basic times taken for entry to and exit from the memo apparatus.

2. With the recursive definition the relationship is in two stages:

(a) The average number of function calls is expressed in terms of the probability of an argument being found on the rote.

$$C_c = \sum_{n=1}^{n=50} n \,.\, P \,.\, (1-P)^{n-1} + 50 \,.\, (1-P)^{50}$$

where $n$ is a particular call number and 50 is the average depth of recursion.
(b) The observed time can be expressed in terms of the average number of function calls.

$$T_m = \tfrac{1}{2} . D . S + (C_O - 1) * (T_{sub} + M + D . S)$$

$C_O$ is the observed number of function calls,
$C_C$ is the calculated number of function calls,
$T_{sub}$ is the time taken to evaluate one sub-call of the recursive function.
Using

$T_f = 400$ ms
$T_{sub} = 8$ ms
$S = 1$ ms
$M = 4$ ms

a very good match with the observed times (figure 3) is obtained.

Extrapolating from these equations it is possible to conclude very generally that memo functions produce significant speed-ups in the evaluation of numerical functions – especially if they are recursively defined. These functions should take a reasonable time to evaluate (200 ms on the particular system used) and should take no more than 100 different argument values. To extend the applicability of memo functions it would be necessary to improve the search technique (so as to reduce the values of $S$) and perhaps also implement the apparatus in machine code (improving the value of $M$).

Van Emden (personal communication) has recently implemented a tree-structured rote and compared the relative access times for arguments on this type of rote with those on a linear rote. His results are shown in table 2. Obviously it now becomes feasible to search larger rotes. At the same time, because the search time is so much reduced, Van Emden estimates that memo-izing functions which only take 50 ms to evaluate will now produce gains.

| Standard deviation of argument population | 25 | 100 |
|---|---|---|
| Probability of an argument being on the rote | 0·69 | 0·58 |
| Size of rote | 64 | 256 |
| Relative access times for | | |
| (a) linear rote | 1·0 | 4·6 |
| (b) tree rote | 0·25 | 0·4 |

Table 2. Relative access times for two types of rote. Arguments are drawn from a normally distributed population.

## MEMO FUNCTIONS AND ROTE LEARNING

A very simple control situation has been devised where the use of memo functions for rote learning has been exploited. The naivety and simplicity of

the learning situation is intentional. It enables the learning and planning procedures to be examined without extraneous complications of environmental influences. If the ideas work here, in this simple situation, they can be used with more confidence in more complex problems. Alternatively, if the ideas are not satisfactory, the results will show the need for caution in the more complex tasks.

### The Knight

The problem situation devised is that of a pseudo-Knight alone on a chess board (figure 4). This Knight may move as it wishes, but must not 'fall' off the edge of the board.

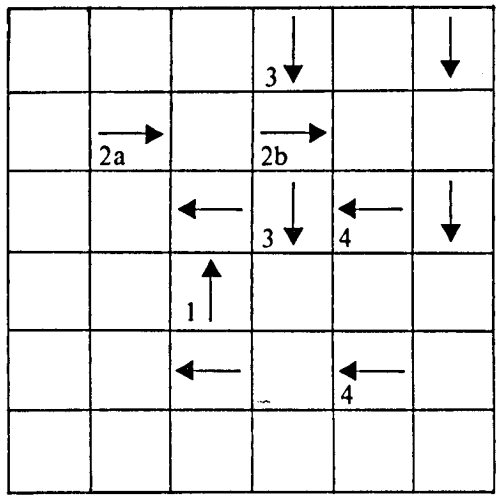



Figure 4. The Knight's problem. A Knight is started in a random position with a randomly selected orientation (e.g. 1). There are two possible moves. In each the Knight is rotated 90° clockwise (e.g., to 2a or 2b). The task is to find a sequence of moves which will cycle (e.g. 1 . 2a . 3 . 4 . 1)

Its position is specified by the board co-ordinates and also by a direction in which it faces. It has two moves open to it. These are to advance two squares and then (a) move left or (b) move right. As a further constraint it will always be rotated 90° clockwise after every move. This turns out to be the simplest arrangement which will allow the Knight to visit every square on the board. It has a further advantage in that it gives the Knight a tendency to go round in circles. This 'cycling' can be used as a measure of the Knight's success at not falling off the edge of the board.

The procedure followed is simple. The Knight is started off at random on the board, or rather from a position from which it is possible to cycle. It uses its memory to look ahead and select a move, which it then takes, updating its memory with the result of this move. Thus it continues until it either cycles or falls off the edge.

Within the Knight there are two functions which are important from the learning and planning point of view.

1. *The predict function*

There is a PREDICT function which acts as a learning function. This is a dummy function whose result is undefined. However, if it is made into a memo function the rote acts as an ideal structure for containing the information, gathered from experience, about the results of taking particular moves from particular positions. Memo functions are written so that they can be updated by assignment in much the same way that an entry is placed in an array (Michie 1967a). Consequently updating the rote is a very simple matter. Obviously the larger the rote of the PREDICT function the greater the look-ahead potential and the better the performance.

While memo functions may not in fact be the best way of organizing the memory structure they do have this advantage of being easy to implement. This is especially true with the generalized version of the apparatus. Using the ROTEUPDATE function it is possible to group together all state-action pairs which lead to the same result. This must be one of the first steps of any automaton which seeks to generalize about its environment. Obviously the Knight in this situation can be treated as a very simple automaton. As such many of the ideas behind its design are due to Popplestone (1967).

2. *The search function*

The second important function is the lookahead, or SEARCH, function which plans ahead from a particular position and decides which move to select. Two different versions of this function were investigated.

(a) *Depth first search.* This is a straightforward procedure. The Knight looks first left and then right from its position and then grows a lookahead tree from each of the two resulting states. There is an evaluation function which is applied to a state to measure its desirability (0 is a fail and 1 is on the board). This function is applied to the states on the lookahead tree and a value for the state at the root of the tree is produced by back-up and maxi-maxing. Because the values at the lower levels of the trees are added to those higher, the resultant value is a measure of how many steps it is possible to take from a particular position. The Knight is attempting to maximize this and so selects the move which leads to the state with the highest value. Ties are broken at random. The lookahead tree is limited in depth, usually set at six.

The apparent disadvantage of this technique is that at every step a whole search tree has to be regrown. It would seem to be a better idea to retain a planning tree from step to step, discarding only outdated portions. Such a facility is inherent in the partial search strategy employed in the Graph Traverser algorithm. Accordingly a Graph Traverser package was written and applied to the planning phase.

(b) *Graph Traverser search.* The Graph Traverser (Doran and Michie 1966,

Michie 1967), is a heuristic algorithm for finding a path across a problem graph whose nodes are a representation of the problem states and whose connecting arcs represent the application of operators to those states (i.e., legal moves). A modified version of a general purpose POP-2 Graph Traverser program (Marsh 1969) has been used. An implementation of this in the form of a library package is described in an Addendum to this paper.

In the Knight's problem the Graph Traverser grows a search tree (initially from a starting state) by applying an operator (move left or move right) to that which, at each time, is the most promising state. The 'promise' of a state is measured by the evaluation function. As with the depth first search process the evaluation function scores 0 for a fail state and 1 for a state which is on the board, adding in the value of the parent state, so as to produce a measure of the distance from the start state. When the tree reaches a specified limit the search halts – yielding as a result the most promising state on the tree. A path to the root is retraced from this state, and the first move on this path selected. Any states which resulted from applying the other operator to the root of the tree are pruned from the tree, so that a tree, relevant to the move about to be made, is retained in store. Assuming that the Knight moves as predicted the tree can continue to be grown up to the specified limit, and then the selection and pruning procedures are repeated.

So as to obtain a direct comparison between this type of search process and depth first search the paths on the search tree were limited to a specified distance (usually six) away from the current root of the tree.

**Rote learning results and a comparison between depth first search and Graph Traverser search**

When the Knight's behaviour is to be exploratory the PREDICT function is a memo-ized dummy function. It can however be set, not as a dummy,

| | Depth first search | | GRAPH TRAVERSER search | | |
|---|---|---|---|---|---|
| Depth of lookahead | Tree size per step | Time (secs) per step | Tree size Per step | No. of nodes per step | Time (secs) per step |
| A. Predict function is not dummy and not memo-ized | | | | | |
| 3 | 12·0 | 0·1 | | | |
| 4 | 20·7 | 0·2 | 21·2 | 11·1 | 1·4 |
| 5 | 33·1 | 0·3 | 32·8 | 16·0 | 3·0 |
| 6 | 51·4 | 0·7 | 56·0 | 26·9 | 8·2 |
| B. Predict function is memo-ized dummy | | | | | |
| 6 | 13·0 | 1·2 | 14·6 | 4·7 | 0·8 |

Table 3. Comparison between depth first search and Graph Traverser search in the Knights problem.

but to contain the full information about the results of each move. Doing this enables a direct comparison to be made between the two types of look-ahead before the PREDICT function is memo-ized. Table 3 shows some results.

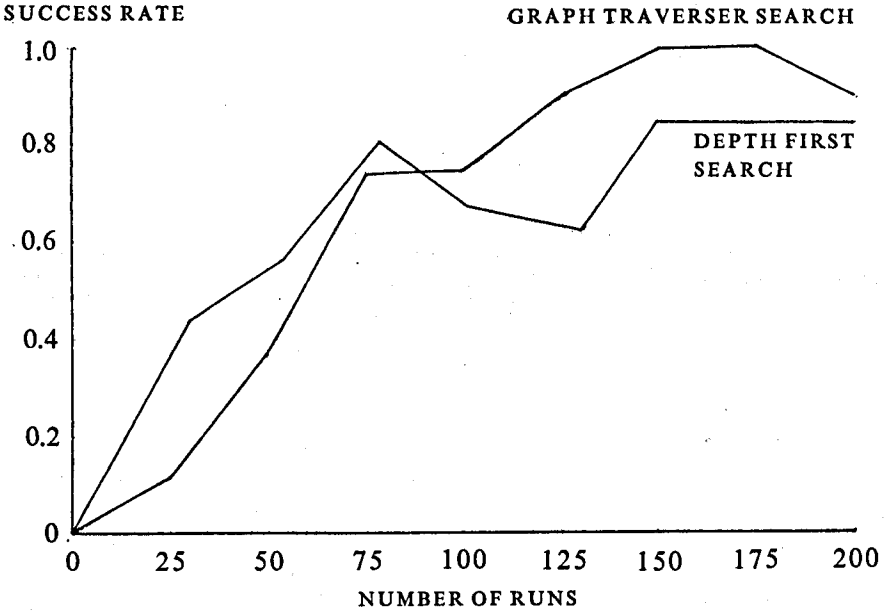



Figure 5. Learning curves for the Knights problem. Only dummy PREDICT function is memo-ized.

Even though the Graph Traverser is on an average looking at less nodes per step it is taking very much longer than the depth first search. This is because the actual process of applying an operator to a state to grow the search tree takes relatively such little time that it is not worth trying to save it.

The situation changes when the PREDICT function is memo-ized dummy, because the rote search takes longer than operator application. In this learning situation the search trees are smaller and less branched with the result that the Graph Traverser search becomes the quicker (see table 3).

The learning curves for the two types of search are shown in figure 5. The data are plotted as the average of a batch of 25 random starts over a series of 200. The performance is measured in terms of the length of a run before failure. If the Knight cycles, this length is regarded as infinite. A performance measure of $(1 - 1/\textit{length of run})$ is used. Under completely random behaviour this has a value of 0.7. The final measure used is therefore:

$$\textit{success rate} = \frac{(1 - 1/\textit{length of run}) - 0{\cdot}7}{1 - 0{\cdot}7}$$

This ranges from 0 for random behaviour to 1 for behaviour which succeeds in cycling every time.

## MEMO-IZING THE SEARCH PROCESS

When Samuel (1959) utilized the stored values associated with each state the gains he obtained were more than just a saving in the growth of a particular branch of a lookahead tree. This is because the stored values had been obtained from complete lookahead. To use these again part way down a lookahead tree will extend its effective depth at that point thereby strengthening the lookahead. The use of memo functions in the Knight's planning routines should, on a very much smaller scale, achieve the same effect.

With depth first search this is very easy to implement because the search process maps from states to values, and furthermore is written recursively. State-value pairs from complete searches will therefore be entered in the rote, but the rote will be searched every time a new state is encountered in the lookahead.

Memo-ization of the Graph Traverser cannot be done in the same way because the search process maps from states to states, producing by implication a path. It does, however, seem feasible to associate a rote with the search process which will contain paths from root states to their best descendant nodes, and then search this rote before applying an operator. If this search is successful the tree can be extended by an extra path and not just by a single state. When entering new paths on the rote the UPEQUIV function can look for overlaps and, if found, the path entries in the rote can be extended. The SEREQUIV function will look not just at the first state in the paths, but right down them to see if there is any portion of those paths that can be utilized. Figure 6 illustrates this process.

The purpose of this memo-ization is to extend the scope of the search without increasing the number of states on the search tree. For this reason, of the states in a path found on the rote, only terminal states can be included in the tree, the other states being referred to by the internal structuring of the tree (i.e., pointers). These states that lie along the path cannot therefore have any operators applied to them. Figure 6 shows how this effectively reduces the size of both the current, but more dramatically the subsequent, search trees.

As a result of this there can be no improvement in behaviour in the Knight's situation. This is because terminal states on the paths from the rote tend to be beyond the depth of lookahead imposed in this problem. Consequently these terminal states will be returned as the best states on a particular search and the Knight will select a move in their direction. Subsequently it will find that it is impossible to grow a further search tree and it will therefore be committed to a path of action which it had previously taken and will have no chance of improving should that path lead to a failure.

Experiments show that there is indeed no improvement in the behaviour of the Knight when the Graph Traverser is memo-ized. Even if there had been this rote mechanism turns out to be very expensive timewise. In fact search time more than doubles with a rote containing only ten paths.

Memo-izing the depth first search process proves more successful. Table 4 shows how the average tree size and the time taken at each step is reduced as the rote increases in size.

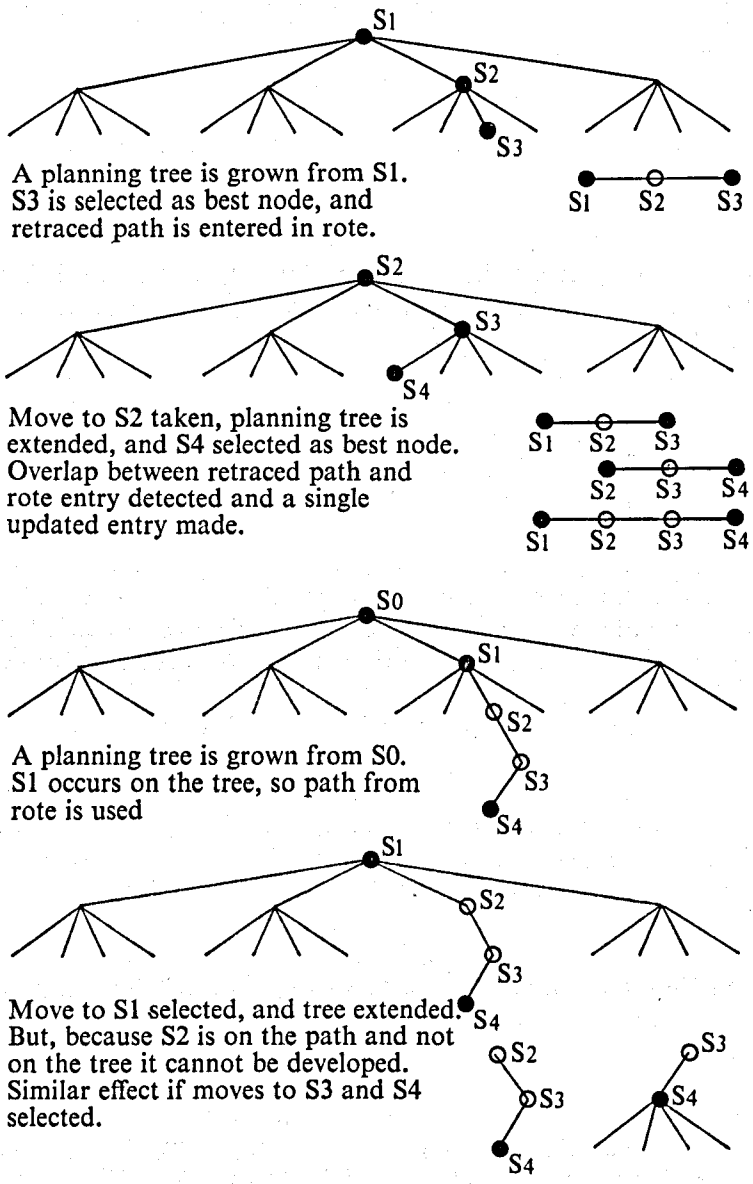



Figure 6. Memo-izing a Graph Traverser search

For these results the PREDICT function was not memo-ized – it had full information about making moves and there was therefore a 100 per cent success rate.

Memo-izing the search process when the PREDICT function is a memo-ized dummy still reduces the average tree size and the time taken at each step. The results are shown in table 5.

| | Depth first search | |
|---|---|---|
| Size of 'search' rote | Tree size per step | Time (secs) per step |
| 0 | 52·4 | 0·8 |
| 5 | 24·6 | 0·8 |
| 20 | 14.2 | 0·9 |
| 40 | 6·8 | 0·6 |
| 60 | 2·6 | 0·3 |

Table 4. Effect of memo-izing the search in the Knight's problem.
*a.* PREDICT function is not dummy and not memo-ized, max. depth of lookahead = 6.

| | Depth first search | |
|---|---|---|
| Size of 'search' rote | Tree size per step | Time (secs) per step |
| 0 | 13·0 | 1·2 |
| 5 | 4·7 | 0·8 |
| 10 | 5·1 | 1·1 |
| 20 | 2·2 | 0·6 |
| 40 | 1·3 | 0·4 |
| 60 | 1·1 | 0·4 |

Table 5. Effect of memo-izing the search in the Knight's problem.
*b.* PREDICT function is memo-ized dummy, max. depth of lookahead = 6.

In these experiments the performance of the Knight is not impaired in any way, remaining approximately the same as when the search process is not memo-ized, even though an improvement would be expected if there is in fact an extension in the effective depth of the lookahead by the application of memo functions. However, further examination of the particular problem shows that a deep lookahead is not crucial to success. In most positions it is enough to avoid those which lead immediately to a failure. This does not invalidate the previous comparisons between different lookahead techniques, provided that the behaviour obtained in each case was the

same. It does, however, mean that to achieve better performance as a result of memo-izing the search process a new situation has to be devised.

### The blind alley situation

Figure 7 shows a very simple problem where the depth of lookahead becomes significant.

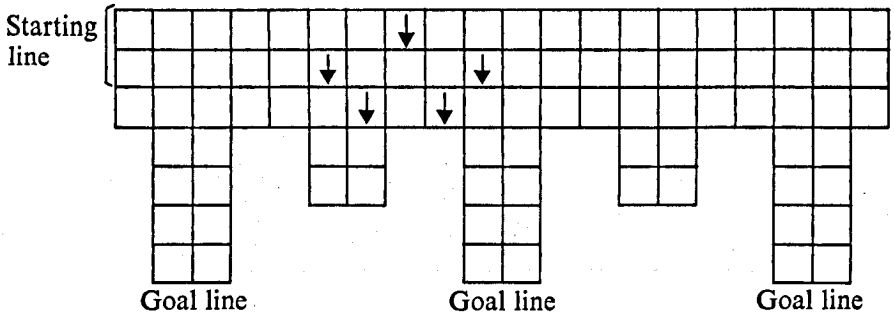



Figure 7. The blind alley situation. A Knight faces downwards, and has four possible moves. It has to pass below the goal line

A 'Knight' faces downwards at all times. It is started at random in the top two rows, and has four moves open to it. The board has two blind alleys from which the Knight cannot return should they be entered. If the lookahead is not deep enough the Knight may not be able to avoid them. When the PREDICT function is not memo-ized dummy and has full information about the possible moves a lookahead to a depth of 3 is sufficient to avoid the blind alleys, while looking ahead only one deep leads to failure 45 per cent of the time. This is shown in table 6. This table also shows that, when the PREDICT

| | Tree size per step | Number of failures Runs 1–50 | Runs 51–100 |
|---|---|---|---|
| A. Vary depth of lookahead | | | |
| 1 | 5·0 | 20 | 25 |
| 2 | 12·8 | 10 | 8 |
| 3 | 24·8 | 0 | 0 |
| B. Set depth=2. Vary size of search rote. | | | |
| Rote size= | | | |
| 0 | 12·8 | 10 | 8 |
| 20 | 4·1 | 9 | 7 |
| 40 | 2·6 | 5 | 2 |
| 60 | 1·7 | 7 | 1 |

Table 6. Depth first search in the blind alley situation. PREDICT function is not dummy and not memo-ized.

function still has full information, and when the depth of lookahead is limited to two, memo-izing the search process results in a decrease in the number of failures due to the expected increase in the effective depth of lookahead. Concomitant with this reduction in the number of failures is a reduction in the size of the search tree, but, because the unmemo-ized PREDICT function is very fast, there is no change in the time taken at each step. This decrease in the number of failures indicates that memo-izing the search process is having the expected effect of extending the lookahead.

When the PREDICT function is a memo-ized dummy, learning curves similar to those in figure 5 are obtained, with the exception that the point of complete success is never reached. Memo-izing the search process in this learning situation produces only a slight reduction in the number of failures. This is because the values initially entered on the rote result from searches where the extent of the lookahead was limited by the small number of moves that the Knight had already taken. Using these limited values later may in fact impair performances whereas a continued search may do better. It will not be until these early results have been dropped from the rote that any improvement will result. If in fact it does occur it is not until at least after 200 random starts (the limit of the experimental runs). Though this delay could be avoided, perhaps by not memo-izing the search process until the rote of the PREDICT function has reached a certain size, or by only entering in the search rote values resulting from a lookahead to a maximum depth, the resultant bag of tricks would involve rather more than the straightforward application of memo functions.

## CONCLUSIONS

Memo functions embody on a small scale many of the principles of rote learning employed by Samuel. Under certain conditions they have been shown to produce successfully significant speed-ups in function evaluation time.

They have been applied to a simple control situation, known as the Knight, where the rote of a memo-ized PREDICT function holds the memory structure of the Knight in a readily accessible form.

In this situation comparisons were made between lookahead by a depth first search and by a Graph Traverser search. The wastefulness of the depth first search, in that a fresh tree has to be grown at every step, can be overcome by the use of the Graph Traverser where the relevant portion of the planning tree is retained from move to move. As long as the process of growing a tree takes longer than the routines required to save the tree from step to step, the Graph Traverser approach is worthwhile.

However, memo-izing the search process can result in the depth first search technique becoming more efficient than the Graph Traverser. Under suitable conditions (such as the blind alley situation) this advantage is greater because the effective depth of the lookahead is extended, and also

because of the impracticality of applying memo functions to the tree-growing routines of the Graph Traverser.

**Acknowledgement**

I am indebted to International Computers Ltd and the Science Research Council for financial support during a two-year secondment to the Department of Machine Intelligence and Perception from ICL.

Throughout the work Professor Michie has been a constant source of ideas and encouragement.

## REFERENCES

Burstall, R.M. & Popplestone, R.J. (1968) POP-2 Reference manual. *POP-2 Papers*. Edinburgh: Edinburgh University Press.

Doran, J.E. & Michie, D. (1966) Experiments with the Graph Traverser program. *Proc. R. Soc. A* **294**, 235–59.

Marsh, D.L. (1969) LIB GRAPH TRAVERSER *Multi-POP Program Library Documentation*. Edinburgh: Department of Machine Intelligence and Perception.

Michie, D. (1967) Strategy building with the Graph Traverser. *Machine Intelligence 1* (eds Collins, N.L. and Michie, D.). Edinburgh: Oliver and Boyd, pp. 137–64.

Michie, D. (1967a) Memo functions: a language feature with rote learning properties. *Research Memorandum MIP-R-29*. Edinburgh: Department of Machine Intelligence and Perception.

Michie, D. (1968) 'Memo' functions and machine learning. *Nature* **218**, 19–22.

Michie, D. & Ross, R. (1970) Experiments with the adaptive Graph Traverser. *Machine Intelligence 5*, pp. 301-18 (eds Michie, D. and Meltzer, B.). Edinburgh: Edinburgh University Press.

Pohl, I. (1970) First results on the effect of error in heuristic search. *Machine Intelligence 5*, pp. 219-36 (eds Michie, D. and Meltzer, B.). Edinburgh: Edinburgh University Press.

Popplestone, R.J. (1967) Memo functions and the POP-2 language. *Research Memorandum MIP-R-30*. Edinburgh: Department of Machine Intelligence and Perception.

Samuel, A.L. (1959) Some studies in machine learning using the game of checkers. *IBM J. Res. and Dev.* **3**, 210–29.

## ADDENDUM

### The Graph Traverser

Formal expositions of an early form of the Graph Traverser algorithm have been given by Doran and Michie (1966), and Pohl (1970) and of a form essentially similar to that used here by Michie and Ross (1970). This is an informal description of some of the Graph Traverser routines as implemented (Marsh 1969) in the POP-2 programming language and used in this work.

The main function provided (GROWTREE) extends a dynamic search tree by applying operators, one at a time, to the most promising states in that graph. The nodes on the graph are POP-2 records containing information about the problem state at that node, the operators that have been applied to that state, the 'promise' or value of that state, and also the parents of that state.

Obviously an initial tree will contain only one node, whose state is the initial problem state. The tree will be extended from this state until a

goal state is found, or until some resignation criterion is reached. The growth of the tree is governed by the parameters of the GROWTREE function.

In the following descriptions this notation is used to describe the arguments and results of functions: ⟨POP-2 function name⟩ ∈ ⟨argument types or names⟩ ⇒ ⟨result types or names⟩.

These parameters are:

1. A Tree.
A list of nodes, ordered so that the best (most 'promising') is at the front of that list. There will either be just one node (the initial node), or a partial search tree, produced by some previous call of GROWTREE.

2. A Predict function.

PREDICT ∈ STATE, OPERATOR ⇒ STATE;

This specifies the effect of applying an operator to a state.

3. An Evaluation function.

EVAL ∈ NODE ⇒ VALUE;

The value of a node is a measure of its distance from a goal state. It is used as a measure of the 'promise' of a state.

4. A Resignation function.

ISLIMIT ∈ TREE ⇒ TRUTHVALUE;

This will return the result TRUE if the search should terminate. It could be because the goal state has been found, or because the tree has exceeded a specified size.

5. A function for deciding whether it is possible to apply any operators to a state.

ISDEVELOPED ∈ NODE ⇒ TRUTHVALUE;

6. A function (BETTERTHAN) for ordering the nodes on the tree, so that those with more 'promise' are at the front.

The GROWTREE function returns as its result the search tree at the time of resignation (ISLIMIT).

The GROWTREE algorithm is

(a) If the resignation criterion is reached (TREE . ISLIMIT) then exit.
(b) Find the next node on the tree, with greatest promise, to which it is possible to apply an operator (testing applicability with ISDEVELOPED).
(c) Apply an operator to that state and produce a new node (PREDICT).
(d) Insert that new node on the tree (ordering with BETTERTHAN).
(e) Go to (a).

Once a search tree has been grown by GROWTREE there is a function provided for retracing a path from a specified node, usually the 'best' node, to the root of the tree. States on that path may be printed out, or operators from that path may be selected and applied to a 'real' world.

If the search ended before a goal was reached the tree can be pruned and GROWTREE called again.

PRUNE ∈ TREE⇒⟨root of old tree⟩, ⟨pruned tree⟩;

This function retraces a path to the root of the tree from the best node on the tree to which it is still possible to apply operators. The root of the tree is left as a function result, and its immediate descendant on the retraced path is marked as a new root. Any node on the tree not retraceable to this new root is then deleted.

### The Graph Traverser Search in the Knight

In the Knight application the SEARCH function is the GROWTREE function partially applied to its parameters.

GROWTREE (%PREDICT, EVAL, ISLIMIT, ISDEVELOPED, BETTERTHAN%)→SEARCH;

This is a mechanism for 'freezing' these parameters into the function SEARCH so that as a consequence it only takes one argument.

SEARCH ∈ TREE⇒TREE;

The parameters are specified as:

1. PREDICT. Already discussed.
2. EVAL. Already discussed.
3. ISLIMIT. Resign only when tree reaches specified size.
4. ISDEVELOPED. A node is considered developed if two operators have been applied to it, or if that node is more than a specified depth from the root node. A fail state and an undefined state are also considered developed.
5. BETTERTHAN. A node with a higher value (derived from EVAL) is considered better. If two nodes have equal value the choice is random.

After SEARCH is called a path is retraced from the best node on the tree. The first operator on that path is selected and taken. Provided the resultant position matches the predicted position on that path, the tree can be pruned and SEARCH entered again, this time with a partial search tree.

### Depth First Search in the Knight

The SEARCH function used in depth first search is much simpler than that used in Graph Traverser Search.

In this case

SEARCH ∈ STATE⇒VALUE;

and it is defined as

```
FUNCTION SEARCH STATE;
IF STATE . ISDEVELOPED THEN EVAL (STATE) EXIT
BETTER (SEARCH (PREDICT (STATE, LEFT)),
SEARCH (PREDICT (STATE, RIGHT)));
END;
```

where PREDICT, EVAL and ISDEVELOPED are the same as those used in the Graph Traverser search, and BETTER selects the 'better' of two values.

# 16

# Experiments with the Adaptive Graph Traverser

Donald Michie and Robert Ross
Department of Machine Intelligence and Perception
University of Edinburgh

**Abstract**

A formal description is given of GT4, a revised and extended version of the Graph Traverser. Methods are described whereby GT4 can improve its performance at run time (a) by automatic optimization of parameters used by the evaluation function and (b) by dynamic re-ordering of operators. Neither method depends upon there being any successful searches in the program's past experience of a given problem. The essential feasibility of both approaches has been validated in experimental tests using sliding block puzzles. Two planned extensions, 'local smoothing' and 'regionalization' are described.

## INTRODUCTION

The Graph Traverser (Doran and Michie 1966), and subsequent work based on it, represents an attempt to adapt game-playing methods, particularly those of Samuel (1959), to automatic problem-solving. The design objective is not the simulation of human problem-solving as a study in psychology, but rather to provide an efficient general-purpose search procedure appropriate to non-numerical problem domains. There is a parallel with the development of direct search techniques for numerical function minimization, for example pattern search (Hooke and Jeeves 1961), simplex (Spendley, Hext and Himsworth 1962, Nelder and Mead 1965). These now form a staple constituent of well-stocked program libraries.

The main topic of the present paper is the proposal that general-purpose search procedures should be able to use accumulating experience of a given problem to improve search efficiency at run time. One of the mechanisms to be described involves optimization of numerical parameters used to control the search, reminiscent of the 'learning by generalization' used in Samuel's (1959) checkers-playing program. In our experiments it was convenient to call the Graph Traverser recursively for this purpose; in the

lower-level calls the program implemented the pattern-search method of Hooke and Jeeves. Thus classical numerical methods can be viewed as special cases of more general search procedures which make no assumptions of continuity or dimensionality.

Some of the formulations of heuristic search which we shall employ were first coined in the GPS work (Newell, Shaw and Simon 1957, Ernst and Newell 1969). The Graph Traverser diverges from GPS in motivation and method; it may be useful to list, as we have done in table 1, some of the contrasts.

| | GPS | Graph Traverser |
|---|---|---|
| A. Method of selecting next state | Current state* | Most promising state on stored lookahead tree, selected by evaluation function |
| B. Method of selecting next operator | Operator-difference table | Re-trace from most promising state |
| C. Action if operator inapplicable | Recursive call to solve the sub-problem: 'find state to which operator *is* applicable' | Try next operator |
| D. Action when memory is full | Search abandoned | Dynamic pruning of lookahead tree |
| E. Adaptive features | — | (a) Automatic optimization of evaluation function (cf. Samuel 1959, 1967)<br>(b) Automatic re-ordering of operators (cf. Quinlan 1969) |

* but lookahead and back-tracking to remembered states can occur when GPS calls itself recursively.

Table 1. A comparison of selected features of GPS and the Graph Traverser. The present paper is concerned with the last item (adaptive features).

A programme of work for an adaptive Graph Traverser was outlined at the first Machine Intelligence Workshop (Michie 1967) in terms which can be related to the scheme of table 1 as follows:

(1) The program must be able to improve its performance before, not after, its first successful search in the problem domain. It must become capable of solving problems for which its initial unimproved form would be inadequate. There is a difference here from the work of Quinlan (1969) which,

although similar in some other respects, depends upon solution of problems for the learning feature to work.

(2) The learning mechanism's ultimate goal is to render the Graph Traverser mode of operation redundant. The chess master, Reti, was once asked 'How many moves do you look ahead?' and is said to have replied 'One: the right one!'. The Graph Traverser represents a particular scheme for managing the lookahead process. Ideally, self-improvement under categories E(a) and E(b) should allow the program continually to reduce the lookahead tree and to narrow the choice among operators at each stage. In the limit the program would be operating on the current state only, having synthesized for itself a complete strategy in the form of a table of states and operators. We are not suggesting attainment of this limit as a practical goal of unaided heuristic search. Methods will be needed, which go outside the present Graph Traverser framework, for automatically abstracting strategically meaningful features so as to set up an 'image space' (e.g., Sandewall 1969) in which higher-level search can proceed, indexed, for example, by a table of state-categories and operator-categories. In the meantime we have the limited aim for an adaptive heuristic search method that it should make effective use of lookahead trees to generate and implement a strategy (a function mapping from states onto operators), and partially condense it into tabular form.

## INFORMAL DESCRIPTION OF THE SEARCH METHOD

The graph of a particular problem is specified to the program implicitly by a list of operators. Each operator will be applicable to some, but not normally all, nodes. When an operator is applied successfully a neighbouring node of the problem graph will be produced. The search for a solution is directed by an *evaluation function* that assigns to each node a numerical value, which is an estimate of its distance from the nearest goal node. At the beginning of a search only the initial node is explicitly known to the program and is capable of being evaluated and *developed.* Development proceeds by applying in succession appropriate operators until a neighbouring node is produced that is estimated to be closer to a goal node. This node is then developed in a similar fashion. At each stage a check is made to avoid adding to the *partial search tree* any node that is already on it. If the most promising node fails to produce a descendant of lower value after all operators have been applied it is labelled as 'fully developed'. The program will always develop the lowest valued node on the partial search tree ignoring any that are fully developed. The search proceeds iteratively in this manner until either a goal is located or the partial search tree reaches a specified size. In the latter case a *partial path* is traced from the lowest-valued developable node to the root of the tree. Then, starting at the root, a specified number (typically one) of nodes on this path are printed out, and that part of the tree is erased, preserving only the part which is dependent from the last node to be printed. If it so happens that the most promising node is also the root then the program will

trace a partial path from the next most promising node and print out the appropriate number of nodes on this path. After *dynamic pruning*, growth of the search tree is resumed. Pruning therefore makes it possible for the search to continue indefinitely. In practice, a *resignation criterion* is specified and when it is met the search is abandoned.

## FORMAL DESCRIPTION

The Graph Traverser's domain includes any problem which can be represented as a graph, $G \equiv (X, \Gamma)$, where

> $X$ is a set of nodes (corresponding to the discrete states of the problem) and
> $\Gamma$: $X \rightarrow 2^X$ is the successor function over the set (corresponding to the 'rule-book' of the problem).

Given a node $s \in X$ (initial state) and a goal-recognizing unary predicate, $P$, it is required to find a sequence of nodes $x_0, x_1, \ldots, x_k$ such that $x_0 = s$, $P(x_k)$ and $x_{i+1} \in \Gamma(x_i)$ for $0 \leqslant i \leqslant k-1$. There may be some additional requirement, for example that $k$ be a minimum.

We now introduce an operator set $\Gamma' = \{\Gamma'_1, \Gamma'_2, \ldots \Gamma'_m\}$, on which we impose an ordering, so that the notation $\Gamma'_i$ denotes the $i$th element in $\Gamma'$. An operator is a (possibly partial) function, $\Gamma'_i$: $X \rightarrow X$, corresponding to a 'move' or 'action' to be performed upon states of the problem. The relation between $\Gamma'$ and $\Gamma$ is: let $X_i = \Gamma(x_i)$ and let $X'_i = \bigcup_{a=1}^{m} \Gamma'_a(x_i)$; then $x_j \in X'_i$ if $x_j \in X_i$. Thus for all $i$, $X_i \subseteq X'_i$. We avoid asserting $X_i = X'_i$ in order to allow for modes of search in which *compound moves* are synthesized and added to $\Gamma'$, supplementing the set of *simple moves* given by the rule book.

We can now represent the path $x_0, x_1, \ldots, x_k$ in terms of a sequence of operator-applications, thus: $\Gamma'_a(x_0)$, $\Gamma'_b(\Gamma'_a(x_0))$, . . ., $\Gamma'_t(\Gamma'_s(\ldots(\Gamma'_b(\Gamma'_a(x_0)))\ldots))$. To allow for the case where $\Gamma'_r(x_i)$ is undefined for some $r$ and $i$ (i.e., some actions not applicable to some states) we extend $X$ to incorporate an additional node representing the value *undefined*. This allows the set of immediate successors of a problem state to include not only all those states which can be generated from it by application of legal moves taken from the rule book, but also a generalized 'error state' which results from the application of any other move. Observe that $\Gamma(\textit{undefined}) = \{\textit{undefined}\}$ and $\Gamma'_i(\textit{undefined}) = \textit{undefined}$ for all $i$.

$\Gamma$ is a function which, with $X$, can be used to generate all paths within the problem graph. The operators in $\Gamma'$ on the other hand are used in conjunction with a search algorithm to generate a search tree. In the original version of the Graph Traverser, which we might here call GT1, $\Gamma$ was used as the move-generator of the search algorithm, so that when a node was developed all its immediate successors were produced. 'Partial development' was implemented by Doran (1968) in ALGOL versions which we shall collectively call GT2,

and by Marsh (1969) in the POP-2 library program GT3. We call the version described in the present paper GT4.

There is a function *strategy*: $X \rightarrow \Gamma'$ which can be applied iteratively to generate a path, one node at a time after each successive call of *strategy*. To specify it so as to give the action of GT4 we consider a tree, $T$, which can be represented as a set of 4-tuples of the form

$t = \langle stateof(t), j, parentof(t), i \rangle$ where $stateof(t) \in X$ and *parentof*: $T \rightarrow T$ is defined so that for all $t$ in $T$
$\Gamma'_i(stateof(parentof\ (t))) = stateof(t)$ except when $t = t_0$, the root node, in which case $parentof(t) = undefined$. Note that $i$ is used to preserve the index number of the operator used to generate $stateof(t)$ from the state component of the predecessor node. In like manner, $j$ is 1 plus the index number of the most recent operator to have been applied to $stateof(t)$.

Growth of the tree proceeds by successive enlargements $T_0, T_1, T_2, \ldots$ where $T_0 = \{undefined, t_0\}, T_1 = \{undefined, t_0, t_1\}, T_2 = \{undefined, t_0, t_1, t_2\}$ etc. Observe that $t_0$ is constructed from $x_i$, $0 \leqslant i < k$, where $x_i$ represents some member of the path $x_0, x_1, \ldots x_k$ and is the argument of the current call of *strategy*. *undefined* is the parent of the root node. It is also the immediate successor obtained from any node in $T$ by applying an operator corresponding to an action inapplicable to that state. Tree growth is guided by

(1) an evaluation function $f$: $X \rightarrow R^+$ (the set of non-negative reals) which estimates the distance of any $x \in X$ along the minimal path to the nearest $x$ satisfying $P$,

(2) an operator-selection function $c$: $T \rightarrow \Gamma'$ which uses the $j$-component of the node to which it is applied to ensure that the new operator selected is not a member of the sequence of operators $\Gamma'_1, \Gamma'_2, \ldots, \Gamma'_j$ which have already been applied to the state component of this node during the current call of *strategy*.

Application of *strategy* to a state, $x$, causes the following chain of events. Recall that $m$ is the size of the operator set $\Gamma'$.

1. Create sets $S = \{\langle x, 1, undefined, undefined \rangle\}$, $S' = \{\langle undefined, undefined, undefined, undefined \rangle\}$; note that $S \cup S' = T$ at every stage
2. Select $t_{min} \in S$ such that $f(stateof(t_{min}))$ is a minimum
3. If $P(stateof(t_{min}))$ or if size of $S \cup S' =$ limit then *retrace* $(t_{min})$ and exit
4. Apply $c$ to $t_{min}$ to obtain $\Gamma'_r$, $j \leqslant r \leqslant m$
   (in the study to be described $r = j$ always)
5. Assign $j+1$ to $j$ in $t_{min}$
6. If there is no $t'$ in $S \cup S'$ such that $\Gamma'_r(stateof(t_{min})) = stateof(t')$ then create a new node $\langle \Gamma'_r(stateof(t_{min})), 1, t_{min}, r \rangle$ and place it in $S$
7. If $j > m$ then move $t_{min}$ from $S$ into $S'$
8. Go to step 2.

The retrace function is evaluated by repeated use of *parentof*. When *undefined* is finally produced, *retrace* gives as its result the member of $\Gamma'$ which corresponds to the first arc of the retraced path. In the operator-reordering mode to be described later it is at this point that promotion and demotion of operators occur. We arrange that when the argument of *retrace* is the root, i.e., $t_{min} = t_0$, $t_{min}$ is transferred into $S'$ and control returns to step 2.

To summarize the action of the search algorithm as a whole, if the following is a POP-2 function (globals defined as previously):

```
FUNCTION TRANSFORM STATE; VARS OPERATOR;
LOOP: PRINT(STATE);
  IF P(STATE) THEN EXIT;
  STRATEGY(STATE)—> OPERATOR;
  OPERATOR(STATE)—> STATE;
  GOTO LOOP
END;
```

then the call TRANSFORM(X0); will cause the sequence $x_0, x_1, \ldots x_k$ to be generated and printed out provided that a solution path exists and is found.

The above is a satisfactory description of an idealized version of the Graph Traverser. It would, however, be seriously inefficient if implemented in the form described, and the following modifications are needed to bring it into line with the actual program.

(1) Once a goal node is located, during step 3 above, the complete path is immediately retraced and printed out and the program halts. Provided that the condition 'if $P(a)$ and not $P(b)$ then $f(a) < f(b)$' is satisfied then the final result is unaffected.

(2) The procedure as described re-grows the complete lookahead tree with every fresh call of *strategy*. Most of this represents needless duplication of work. To eliminate this, step 1 above is omitted, and $S$ and $S'$ (which together comprise $T$) are held as globals and hence the lookahead tree is preserved between successive calls.

In order to get rid of the part of the tree which is *not* re-grown under the previous form of the algorithm it is necessary to introduce a pruning routine into the TRANSFORM function which deletes the root node of the lookahead tree and all side-branches dependent from it. Under standard conditions it can be arranged that the end result is the same under either form of the algorithm, although this is not the case if certain further efficiency-promoting changes are made.

We shall not pursue implementation details, since they are irrelevant to the main topic. It will, on the other hand, prepare the ground and help fix ideas if we state at this point that the 'learning' features which we will describe operate solely by allowing the program at run time to modify

(a) the elements of a global list of numerical parameters, thus modifying the effects of applying $f$,

(b) the ordering of a global list of integers used to index $\Gamma'$, thus modifying the effects of applying *c*.

Implementation of (a) is through recursive call of GT4, while implementation of (b) is through a side-effect of *retrace* immediately before exit from *strategy*.

## 'LEARNING' EXPERIMENTS WITH GT4

Program-modification of the effects of *f* and *c* is restricted to making changes in global structures used by the respective evaluation procedures: no modification of corresponding procedure bodies is involved. The structure used to evaluate *f* is a parameter list analogous to the list of coefficients used in Samuel's program to evaluate his 'scoring polynomial'. In the case of *c* the relevant structure is a permutation on the indices by which *c* orders its selection of operators from $\Gamma'$. Two separate series of experiments were conducted, one to examine the feasibility of automatic optimization of the evaluation process, and the other to test the effect of automatic re-ordering of operators. For these experiments GT4 was implemented as a POP-2 program and run on an ICL 4130 computer.

### Parameter optimization: method

The evaluation function, *f*, is defined as an estimator of the minimum distance from a node of the problem graph to the nearest state satisfying *P*. More generally we may define a distance estimator $d: X \times X \rightarrow R^+$ which estimates the minimum distance between any two nodes of *G*. A given distance estimator over *G* may be transformed into an evaluation function for a particular problem associated with *G* by requiring $f(x) = d(x, x_k)$ where $x_k$ is chosen from the goal set so as to minimize the true distance from *x*, which we denote $\delta(x, x_k)$. Since in practice $\delta$ is unknown there is no general way of obtaining *f* from *d*, except in the special case where the goal set contains only a single node, $x^*$ say. In this case, in the terminology of POP-2 programming, *f* may be produced from *d* by partially applying *d* to $x^*$, i.e., D(%XSTAR%)—>F. This condition obtained in all the experimental work to be reported, and the program did in fact construct *f* from *d* on entry in the manner described. We can usefully list at this point the data objects which the user has to supply to this version of GT4 in order to specify a problem to it and set it to run in self-optimizing mode.

1. $\Gamma'$ in the form of a POP-2 list of functions of one variable,
2. $x^*$ in the form of a POP-2 data-structure of appropriate type,
3. *d* in the form of a POP-2 function,
4. a list of parameters used by the numerical optimizer.

1 and 2 comprise the specification of the problem, whereas in principle items 3 and 4 could be supplied by default settings. By substituting different combinations of values in the global parameter list used by *d* different estimators can be produced whose utility the program must assess. The basis of this assessment, suggested by Doran (1967), is:

(1) that the utility of a given $d$ can be measured by the closeness with which $d(x_i, x_j)$ approximates $\delta(x_i, x_j)$ over $X \times X$; actually a weaker measure is sufficient, namely that the *rank order* of the distances estimated by $d$ should approximate the rank order of the true distances.
(2) that in the absence of direct knowledge of $\delta$ an estimate of this rank correlation can be formed from comparison of estimated with actual distances over the partial search tree, $T$; for example distances from terminal nodes to the root node. $T$ corresponds to a connected subgraph, and hence in some sense to a 'local sample', of $G$. The success of such an approach clearly depends upon the manner in which this local sample has been formed, and hence upon how good an estimator $d$ is in the first place. It also depends upon $d$ possessing a certain homogeneity over $G$, or at least over sufficiently large connected sub-graphs, with respect to the features to which the entries in the global parameter list correspond. This is the same as saying that rank correlations of $d$ with $\delta$ computed over different local samples of $G$ should tend to agree.

The entries in the parameter list can thus be regarded as independent variables of a 'goodness of match' function which it is desired to maximize: specifically this function is the rank correlation coefficient computed over $T$ as outlined above. GT4's search for an optimal combination of settings is carried out recursively after a specified number of searches, the program acting as a direct search numerical optimizer, in a fashion which will be described in detail in the next section.

The optimization technique investigated was proposed by Hooke and Jeeves (1961) and is known as 'pattern search'. While experimenting with the method we produced a modification (Michie and Ross 1969) which resulted in significant improvements in search efficiency as measured by the number of function evaluations. When, however, comparisons were made on a strict real-time basis it was found that such gains could only be maintained for functions involving unusually heavy computational work to evaluate: for the four test functions used in our experiments the unmodified algorithm was faster. It was further found that the algorithm of Powell (1964), which is conspicuously sparing in the number of function evaluations, fared no better than pattern search on the real-time criterion. On the basis of these and subsequent experiments we therefore adopted pattern search as the numerical optimizer to be used in parameter optimization. Pattern search also possessed the advantage that it could be readily represented in a form suitable for execution by the Graph Traverser itself.

## DESCRIPTION OF PATTERN SEARCH

The search proceeds from an initial exploration phase through an alternation of pattern phases and further exploration phases, the latter being followed by an interposed rescue phase where appropriate.

### Exploration phase

An initial base point $\mathbf{b}_1$, is defined by the starting values for $x_1$, $x_2$ (for simplicity we shall envisage a function of only two variables) and the function is evaluated at this point. $x_1$ is then incremented by an amount $\Delta$ and the function's value at the resulting point is compared with that at $\mathbf{b}_1$. If no improvement is obtained another exploratory move is defined by decreasing $x_1$ by $2\Delta$. If this move also fails $x_1$ is reset to its value at $\mathbf{b}_1$. Perturbation of $x_2$ is carried out in a similar manner about the point resulting from the previous exploration.

If the exploratory phase results in an improvement in the value of the function a new base point, $\mathbf{b}_2$, is set and the pattern phase is entered. Failure triggers a decrease in step size. We will now describe the details of this after first explaining the role of the pattern move.

### Pattern phase

As soon as $\mathbf{b}_1$ and $\mathbf{b}_2$, or more generally $\mathbf{b}_n$ and $\mathbf{b}_{n+1}$, exist, a pattern move can be specified as the geometric production of the line joining $\mathbf{b}_n$ and $\mathbf{b}_{n+1}$ to point whose distance from $\mathbf{b}_{n+1}$ is equal to the distance from $\mathbf{b}_{n+1}$ to $\mathbf{b}_n$. This point is a 'candidate' to become a new base-point, $\mathbf{b}_{n+2}$. After each pattern move the exploratory phase is entered and is aimed at revising the pattern move. If there is an improvement over $\mathbf{b}_{n+1}$ in the value of the function either at the candidate point or after the exploratory moves have been performed from this point then a new base point is set, defining a new pattern move. Otherwise the rescue phase is entered.

### Rescue phase

Failure, as defined above, results in a return to the current base point, $\mathbf{b}_{n+1}$, where the exploratory phase is again entered in an attempt to define a fresh pattern move. Further failure causes $\Delta$ to be decayed by the factor $\rho$ after which the exploratory process is again carried out. Subsequent failure results in $\Delta$ being reduced still further. When $\Delta$ falls below a limiting value, $\delta$, the search is abandoned.

## APPLICATION OF THE GRAPH TRAVERSER TO NUMERICAL FUNCTION OPTIMIZATION

The Graph Traverser may be employed as a numerical optimizer if we let problem states correspond to $n$-tuples whose components are the variables of the function to be optimized. This function is used as the evaluation function (compare the use of the cost function of the Travelling Salesman problem as an evaluation function in solution by the Graph Traverser (Doran 1968)). The choice and ordering of the operators will depend on the particular optimization technique that is being implemented. In the case of pattern search it will consist of two operators which are applied to base points and which produce, when successful, new base points. They are:

(1) make a pattern move from the current base point and then carry out an exploratory search

(2) carry out an exploratory search from the current base point.

To complete the adaptation of the Graph Traverser we must inhibit backtracking, a search being terminated when both operators have been applied unsuccessfully to the current base point, i.e., when a Shen Lin search (Lin 1965) has been completed. At this point the step length is reduced and a fresh search is initiated from the current base point.

The similarity between the Graph Traverser and pattern search can be extended to the exploratory phase. Thus we have the Graph Traverser as one of its own operators. To effect this extension we must first define another list of operators consisting of $2n$ exploratory moves, a positive and a negative one forming a pair which is associated with each of the $n$ variables of the function being optimized, i.e.,

$$[+\Delta_1, -\Delta_1, +\Delta_2, -\Delta_2, \ldots +\Delta_n, -\Delta_n].$$

Each subscript identifies the independent variable to which the associated exploratory move is applied. When a new best-valued node is generated the usual procedure (see earlier formal description) is to choose, as an operator to apply to it, the next in order on the list after the last one applied to it. To complete the transformation we must prevent this initialization and arrange that the first operator to be applied to the new best node is the successor of the one that produced it. The Graph Traverser is now adapted for pattern search function optimization.

## EXPERIMENTAL SCHEME

As explained, the rank correlation coefficient, $r$, provides an approximate assessment of merit which can be made independently of any successful searches. By substituting different combinations of parameter settings we compute different values of $r$ over $T$, finding those settings which maximize $r$. In the experiments the recursive optimization routine was entered after the first partial search of each problem and on every 20th succeeding partial search.

Optimization consisted of three stages.

(1) The formation of a list of pairs, each pair consisting of the depth in the tree of a terminal node (its *tree distance*) together with its *estimated distance* from the root.

(2) The tree distances were then rank correlated with the estimated distances and the resulting coefficient maximized by the Graph Traverser operating in *pattern search* mode. The optimized values of the parameters formed a *current estimate* of the optimum.

(3) The current estimate was then incorporated in a *cumulative mean estimate*, obtained by averaging all past estimated optima and the original values of the parameters. The mean estimate was then used by the evaluation function on subsequent searches.

## PARAMETER OPTIMIZATION: RESULTS

Preliminary trials were made with the Eight-puzzle (*see* Schofield 1967), in which 8 sliding blocks must be re-arranged in a bounded $3 \times 3$ array so as to form a specified goal configuration,

conventionally $\begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$,

where the zero stands for the empty square. The operator set consisted of all movements of a block made by sliding it unit distance into the space. The evaluation function was the $P+wS$ expression used by Doran and Michie, whose results indicated that the optimum for the adjustable parameter $w$ lies on a rather flat surface in the region $\frac{3}{2} \leqslant w \leqslant 9$. The size of the partial search tree was 200. Four replicate runs, each with 25 randomly generated puzzles, were done, with $w$ initialized to zero. The step length, $\Delta$, was 0·1. The final values for $w$ were 0·9, 2·9, 2·1, 1·6 respectively; 98 out of the 100 searches terminated successfully, as compared with uniform failure when the optimizing call was suppressed so that $w$ remained fixed at 0.

With this preliminary encouragement, the Fifteen-puzzle was substituted with a search space of 16!/2 as compared with 9!/2. As evaluation function we used Doran and Michie's $\sum_{i=1}^{15} h_i^a p_i^b + cR$, and initial parameter settings $a=1$, $b=1$, $c=50$. Step lengths were 0·1, 0·2 and 4. Other conditions were as before, with the exception that the number of tree-distance/estimated-distance pairs used to compute $r$ was limited to 50 (the number for an average tree in the Eight-puzzle runs was about 75).

Table 2 shows the results of the first run. The values to which $a$, $b$ and $c$ settled down are well within the ranges judged optimal from independent evidence. A more critical test was set up, in which GT4 was run on either 0, 1, or 2 Fifteen-puzzles in optimizing mode, and the values of $a$, $b$, and $c$ were then frozen. Performance was then tested on a new set of 8 Fifteen-puzzles, with results shown in table 3. Evidently very few optimizations have proved sufficient to confer as much adaptive improvement as can be got within this limited framework.

A criticism is that the starting values are arbitrary, and might be said to embody external knowledge of the problem; for example, they are all positive. A natural 'default setting' for initializing each parameter in the *total* absence of externally acquired knowledge would perhaps be zero, and we are now studying the effects of imposing this condition. In the meantime we can summarize by saying that given a very little external knowledge, quite inadequate in itself for solving the problem, the program can very rapidly improve upon it.

| puzzle | number of optimizations | average parameter settings | | |
|---|---|---|---|---|
| | | a | b | c |
| 1 | 0 | 1·000 | 1·000 | 50·00 |
| (unsolved) | 1 | 1·025 | 2·100 | 42·00 |
| | 2 | 0·725 | 2·367 | 40·67 |
| | 3 | 0·575 | 2·292 | 32·67 |
| 2 | 4 | 0·435 | 2·532 | 46·27 |
| (unsolved) | 5 | 0·368 | 2·365 | 38·93 |
| | 6 | 0·326 | 2·179 | 40·65 |
| | 7 | 0·263 | 2·279 | 47·40 |
| 3 | 8 | 0·257 | 2·335 | 42·07 |
| (unsolved) | 9 | 0·222 | 2·255 | 42·07 |
| | 10 | 0·204 | 2·746 | 36·61 |
| | 11 | 0·229 | 2·779 | 36·28 |
| 4 | 12 | 0·222 | 2·733 | 34·44 |
| (solved) | 13 | 0·218 | 2·711 | 34·72 |

Table 2. Fifteen-puzzle: optimization of the parameters of the Doran–Michie evaluation function using a random sample of 5 puzzles. The 5th puzzle was solved before the first partial search had been completed. Lookahead tree of 200 nodes.

### OPERATOR-SELECTION: METHODS

In step 4 of the *strategy* function $c$ is applied to $t_{min}$ to obtain $\Gamma'_r$.

The action of $c$ with which we have experimented is restricted to choosing the $j$th operator in $\Gamma'$ so that $r=j$. If some operators are more frequently useful than others when searching a given problem graph, we would like the preferred operators to come high on the list. Ideally we would like to associate different orderings with different *categories* of node (different 'images' in Sandewall's (1969) notation), arriving at these orderings automatically during the search process, and perhaps discarding, in course of time, all but a few favourites occupying the top places in each such ordering. We have not aimed so high, but have restricted ourselves to investigating whether

| training regime | number of optimizations | parameter settings a | b | c | number of puzzles solved | total number of states generated |
|---|---|---|---|---|---|---|
| no training | 0 | 1·000 | 1·000 | 50·00 | 0 | 4,000 |
| puzzle 1 (unsolved) | 2 | 0·725 | 2·367 | 40·67 | 3 | 3,833 |
| puzzle 2 (unsolved) | 3 | 0·592 | 0·950 | 50·00 | 0 | 4,000 |
| puzzle 1 then puzzle 2 (both unsolved) | 6 | 0·326 | 2·179 | 40·65 | 6 | 2,480 |
| puzzle 2 then puzzle 1 (both unsolved) | 5 | 0·563 | 2·574 | 42·13 | 5 | 3,226 |

Table 3. Fifteen-puzzle: performance on 8 randomly generated puzzles using parameter settings that were obtained after varying degrees of training. Lookahead tree of 200 nodes.

dynamic re-ordering can be feasible and profitable even for a single global ordering of operators. A positive answer would suggest *a fortiori* that self-improvement on a more impressive scale should be attainable for problem-spaces which have been subdivided into state-categories.

The method we have adopted is to promote an operator whenever it is selected by *strategy* and to demote it in the ordering whenever it *could* have been selected (in the sense that it was actually applied during the given call of strategy) but was not selected. In our experiments a demotion was always by one place, and a promotion by one less than the number of immediate descendants of the root node. This scheme is essentially the 'path popularity ratio' proposal of Michie (1967) modified along lines suggested by work of Longuet-Higgins and Ortony (1968). The 'path popularity ratio' is a measure of the frequency with which an operator directs the lookahead along a line which is subsequently adopted, as opposed to directing it away from the main path. Table 4 shows this statistic tabulated for the six symmetry classes into which 48 operators for the Eight-puzzle can be grouped. These

operators comprise all the macro-moves which can be formed from concatenations of the elementary moves, subject to the conditions:
(1) the problem is re-formulated so that $X$ comprises only the 8!/2 states of the board in which the centre square is empty; it follows that every operator is the concatenation of an even number of moves, at least 4 in number. The identity operator is disallowed.
(2) no operator is formed by the concatenation of more than 8 moves.

| category of operator | average frequency on final path / frequency off final path (path popularity ratio) | | | rank position of average path popularity ratio | | |
|---|---|---|---|---|---|---|
| | sample A | sample B | average | sample A | sample B | average |
|  | 0·119 | 0·101 | 0·110 | 1 | 2 | 2 |
|  | 0·075 | 0·084 | 0·080 | 3 | 3 | 3 |
|  | 0·108 | 0·143 | 0·126 | 2 | 1 | 1 |
|  | 0·036 | 0·034 | 0·035 | 6 | 6 | 6 |
|  | 0·070 | 0·077 | 0·074 | 4 | 4 | 4 |
|  | 0·055 | 0·046 | 0·051 | 5 | 5 | 5 |

Table 4. Eight-puzzle: path popularity statistics obtained with random development using the 48 macro-moves. Each sample consisted of 50 randomly generated puzzles. Lookahead tree of 50 nodes.

It should be added that for the purpose of the experiment summarized in table 4, the function $c$ was redefined to give a random selection of the next operator from $\Gamma'$.

Table 4 shows that marked differences do indeed exist between the different categories of operator on the criterion of path popularity. This was sufficient encouragement to run a test of the promotional scheme previously outlined.

In terms of the formal description, *retrace* is extended so as to reduce the index of the selected operator by *size* ($X'_{root}$) − 1 and to increment by one the indices of all other operators leading from the root node.

Table 5 shows the results obtained using the same two samples of randomly constructed puzzles as those of table 4. The initial ordering of the 48 operators was random in each case. The average final ordering, shown in table 5, agrees exactly with that of the path popularity ratios.

| category of operator | rank position after promotion | | |
|---|---|---|---|
| | sample A | sample B | overall average |
|  | 2 | 1 | 2 |
|  | 3 | 3 | 3 |
|  | 1 | 2 | 1 |
|  | 5 | 6 | 6 |
|  | 4 | 4 | 4 |
|  | 6 | 5 | 5 |

Table 5. Eight-puzzle: results of promotional experiment. Each sample consisted of 50 randomly generated puzzles. Lookahead tree of 50 nodes.

At this point the operator ordering was frozen and split into a top 24 and a bottom 24. Duplicate re-runs were then done using the two reduced operator sets with the promotional mechanism inhibited. The results, shown in table 6, demonstrate that the program has sorted the 48 operators into a 'good' and a 'bad' subset; use of the two sets is associated with a marked difference in performance, whether assessed on the proportion of problems solved or on the computational cost per search.

| selection procedure | sample | percentage of puzzles solved | average number of states generated / puzzle |
|---|---|---|---|
| top 24 operators on list after promotion | A | 100% | 62·4 |
| | B | 100% | 72·9 |
| | average | 100·0% | 67·7 |
| bottom 24 operators on list after promotion | A | 24% | 173·3 |
| | B | 30% | 166·4 |
| | average | 27·0% | 169·9 |

Table 6. Eight-puzzle: results of a promotional scheme for operator selection. Lookahead tree of 50 nodes.

### DYNAMIC SHRINKING

This result provoked the idea that the program might improve its own performance and economize its effort from the start, discarding an operator which was already at the bottom of the list if it became a candidate for further demotion. This policy of dynamic shrinking of $\Gamma'$ was given a preliminary trial, with a bound set on the shrinking process so that the size of $\Gamma'$ could not fall below 24. The results were satisfactory, and as far as they went indicated convergence to steady state conditions similar to those of table 6.

### DISCUSSION

Our results establish the feasibility of automatic optimization procedures in heuristic search with respect both to state-evaluation and preference-ordering of operators. Improvement of performance is not dependent, under either heading, on having problems sufficiently tractable to be within reach of the program in its non-adapted state. Optimization proceeds through modifying

the action of $f$ and $c$ respectively without modifying their structure, i.e., through trial-and-error modifications of global parameters.

Substantial further gains in performance should accrue from two extensions which we are now pursuing:

1. *Local smoothing.* Much of the program's *time* is consumed in the application of large numbers of operators (typically the whole set) to a small number of unproductive nodes, while most of its *progress* is achieved through a larger number of nodes, on each of which only a little development work is expended. The 'obstructive' nodes are those to which $f$ has assigned a misleadingly low value; in other words $d(x, x^*) \ll \delta(x, x^*)$. The program ought, but in its present form does not, gradually re-value such a node in the light of accumulating evidence from its high-valued immediate descendants. A simple approach, reminiscent of the use in genetics of progeny-testing to improve the assessment of an individual's genotypic value, would be to use $f'$ rather than $f$ to guide the search, where

$$f'(x) = \frac{f(x) + w\sum_{i=1}^{j-1} f(\Gamma_i'(x))}{1 + w(j-1)}$$

and $w$ is a weighting coefficient. Preliminary results are positive.

2. *Regionalization.* We do not have any systematic method for partitioning problem spaces in general into regions for the purpose of associating independent promotional processes with these regions. Such partitioning, even if done crudely, can be surprisingly effective. Samuel's (1967) 'signature tables'; and Michie and Chambers' (1968) 'boxes' are relevant cases in point. In the experience of human solvers the Eight-puzzle exhibits marked interaction between different features of problem states and the relative utilities of different operators. It should therefore provide a good testing ground for regionalized promotional systems, in which the features used for partitioning into regions are in the first instance supplied to the program from outside. The possibility to which we attach most importance arises from the use of dynamic shrinking in the independent regional promotional processes. This could result in the action of *strategy* being gradually re-structured during search until only a small specialized operator set was associated with each region, forming the basis of small and economical lookahead trees. A self-mediated transformation of this kind would constitute a step towards the compression of lookahead-type strategies into tabular form, a process which seems to play a role in the acquisition of skill by human solvers.

**Acknowledgements**

This work was done as part of a programme of research into machine simulation of learning, cognition and perception supported by the Science Research Council. One of us (R.R.) is in receipt of an S.R.C. Research Studentship which is here gratefully acknowledged.

## REFERENCES

Doran, J.E. Michie, D. (1966) Experiments with the Graph Traverser program. *Proc. R. Soc. A*, **294**, 235–59.

Doran, J. (1967) An approach to automatic problem-solving. *Machine Intelligence 1*, pp. 105–23 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Doran, J. (1968) New developments of the Graph Traverser. *Machine Intelligence 2*, pp. 119–35 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.

Ernst, G.W. & Newell, A. (1969) *GPS: A Case Study in Generality and Problem Solving*. New York and London: Academic Press.

Hooke, R. & Jeeves, T.A. (1961) 'Direct Search' solution of numerical and statistical problems. *J. Ass. comput. Mach.*, **8**, 212–29.

Lin, S. (1965) Computer solutions of the Travelling Salesman problem. *Bell System Tech. J.*, **44**, 2245–69.

Longuet-Higgins, H.C. & Ortony, A. (1968) The adaptive memorization of sequences. *Machine Intelligence 3*, pp. 311–22 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Marsh, D.L. (1969) LIB GRAPHTRAVERSER. *Multi-POP Program Library Documentation*. Edinburgh: Department of Machine Intelligence and Perception, University of Edinburgh.

Michie, D. (1967) Strategy-building with the Graph Traverser. *Machine Intelligence 1*, pp. 135–52 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Michie, D. & Chambers, R.A. (1968) BOXES: an experiment in adaptive control. *Machine Intelligence 2*, pp. 137–52 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.

Michie, D. & Ross, R. (1969) A comparison of Powell's general purpose function optimizing algorithm with that of Hooke and Jeeves, using a real-time criterion. *Research Memorandum MIP–R–36*. Edinburgh: Department of Machine Intelligence and Perception, University of Edinburgh.

Nelder, J.A. & Mead, R. (1965) A simplex method for function minimization. *Computer Journal*, **7**, 308–13.

Newell, A., Shaw, J.C. & Simon, H.A. (1957) Preliminary description of general problem solving program – I (GPS-1), *CIP Working Paper No. 7*. Pittsburgh: Carnegie Institute of Technology.

Powell, M.J.D. (1964) An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, **7**, 155–62.

Quinlan, J.R. (1969) A task-independent experience-gathering scheme for a problem solver. *Proc. International Joint Conference on Artificial Intelligence*, pp. 193–7 (eds Walker, D.E. & Norton, L.M.).

Samuel, A.L. (1959) Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, **3**, 211–29.

Samuel, A.L. (1967) Some studies in machine learning using the game of checkers, 2 – recent progress. *IBM J. Res. Dev.*, **11**, 601–17.

Sandewall, E.J. (1969) A planning problem solver based on look-ahead in stochastic game trees. *J. Ass. comput. Mach.*, **16**, 364–82.

Schofield, P.D.A. (1967) Complete solution of the 'Eight-puzzle'. *Machine Intelligence 1*, pp. 125–33 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Spendley, W., Hext, G.R. & Himsworth, F.R. (1962) Sequential application of simplex designs in optimization and evolutionary operation. *Technometrics*, **4**, 441–61.

# MAN – MACHINE INTERACTION

# 17

# An Interactive Theorem-Proving Program

John Allen and David Luckham
Computer Science Department
Stanford University

**Abstract**

We present an outline of the principal features of an on-line interactive theorem-proving program, and a brief account of the results of some experiments with it. This program has been used to obtain proofs of new mathematical results recently announced without proof in the *Notices of the American Mathematical Society*.

## 1. INTRODUCTION

The construction of the theorem-proving program we are going to discuss here is by no means completed. Rather, the program exists in a state of continual revision and extension, and that part of it which is running and yielding results at the moment is intended to form the nucleus of a much larger and more extensive automatic deduction system in the future. The program is being developed with a number of different questions and applications in mind.

First of all, some of the present generation of deduction programs are already capable of proving the sort of theorem of an elementary mathematical theory that appears in the textbooks either as a basic theorem or standard exercise (sometimes as a 'starred' exercise), and might reasonably be classified as 'somewhat' tricky. We have in mind the theorems of algebra and number theory obtained by programs discussed by Wos, Robinson and Carson (1965), Wos *et al.* (1967), and Luckham (1968). In addition, it is reported by Guard *et al.* (1969) that an open problem in modular lattice theory was solved with the aid of an on-line program (and it is interesting to note that, with reference to the initial set of axioms and hypotheses, this seems not to be the most difficult theorem that has been proved by a program so far). We are thus motivated to ask if, by adding a flexible interactive facility to a good deduction program, it is possible to construct a system that would be useful in investigating some fairly basic mathematics. This could take the form of searching for interesting consequences of unusual sets of

axioms, or for alternative proofs of known results, or of checking the steps of informal mathematical proofs for correctness. The interactive facility is to allow the user to monitor and direct the progress of a proof search at run time.

So far, we have used the program described below (sections 2 and 3) in interactive mode to prove some mathematical results announced in a recent issue of the *Notices of the American Mathematical Society* (Chinthayamma 1969). These are results concerning questions of dependence in the axiomatization of Ternary Boolean Algebra (henceforth called TBA) and in degree of difficulty appear quite similar to trigonometric identities. Interestingly, certain of the more 'important looking' deductions that turned up in the course of a proof search and that we chose to use in proceeding with the search, were (as we later found out) standard theorems of TBA published by Grau (1947).

There then follows a number of questions bearing upon the problem of finding out how the existing logical rules of inference and proof search strategies are best put to good use. For example, the program under discussion here uses the Resolution Principle (Robinson 1965a) as the fundamental rule of inference for that part of it dealing with pure first-order logic. There are many refinements of this principle aimed at improving the efficiency of the proof search and, for the most part, they do not work uniformly well, but tend to help or hinder the search depending on the problem involved. The same thing is true of the equality rule (Paramodulation (Wos and Robinson 1968)) which the program uses for problems involving the identity relation, and doubtless we shall find that it also applies to our extended system for multi-sorted logic. We need to gain a good deal more information about the relative merits of the various search strategies than we have at present. Also, the use of these two rules of inference together poses some additional questions. In some problems a significantly large set of deductions is generated by both rules, so that methods for reducing this duplication of effort would definitely be useful. We also need to know how different formalizations of a problem affect the efficiency with which proofs are obtained; sometimes a formalization within applied logic with identity is easier for the program to deal with than a pure first-order logic formulation, but this is not always true. There are basic programming questions. We do not have any theoretical study to tell us the most efficient way to implement the basic operations at the lowest level of the program (e.g., the tests for unification, alphabetic variants and subsumptions) so that even this is, at present, a matter for experimentation. And there is the problem of allowing for the incorporation of new strategies and inference rules into the program as they are developed. This rather chaotic state of knowledge forces upon us a number of basic decisions about the construction of the program and the way it is used. Our program places nearly all efficiency strategies under user control. At the start, when the program is first called, the user can choose which

strategies he wishes to apply to a particular problem, and he may alter his choice at any time during the search for a proof. This facility allows the user to take advantage of his own experience with the class of problem he is working on, and it also gives us an easy way to compare the performances of various combinations of strategies. Further decisions are to write the program in LISP, and to base its construction on a very simple fundamental flow diagram in which the subroutines for different operations are kept independent of each other, and operations of different types (e.g., choice strategies, inference rules, editing strategies; see figure 1) are programmed in strictly separate boxes of the diagram. While both of these decisions involve sacrificing some computational speed and efficiency, they make it relatively easy to introduce new operations, change the implementation of existing ones, and to extend the options available to the user, all of which one is almost certainly going to want to do.

Finally, the possibility of applying the deduction program to areas other than mathematical theorem-proving, for example information retrieval and projects in artificial intelligence requiring a certain deductive capability, should be mentioned. Here, it is usually necessary for the program to be able to reconstruct detailed information about any proof it generates, and we have allowed for this, so that, in particular, the question-answering techniques outlined by Green and Raphael (1968) and Luckham and Nilsson (1969) can be added easily.

The discussion below is organized into three sections and from time to time it assumes the reader has a slight acquaintance with the sort of terminology that appears in Robinson (1965a) and Luckham (1967, 1969). We shall deal only with the program for first-order logic with identity. In section 2 we describe the basic flow structure of the program and give a brief survey of what is known about the search strategies it uses. Section 3 contains a description of the on-line facility and the way we tend to use it at present, and some practical comments about the various strategies. Section 4 contains an analysis of some proofs of theorems in group theory and TBA.

## 2. BASIC STRATEGIES

Originally, the problem of running out of memory space was the crucial stumbling block of all the early theorem-provers, and led to concentration upon the development of space-saving strategies. This, together with the increase in the size of computer memories, has placed us in a situation where the computation time required to find a proof is now an equally important problem, especially if one has on-line interactive applications in mind. Indeed, with some of the more sophisticated strategies currently available, it is possible for the program to spend most of its time saving space! We have tended therefore to experiment with space-saving strategies which may not be the most restrictive, but have relatively simple and fast implementations,

and stand a good chance of saving computation time as well (see example 5 in Luckham (1969)).

The fundamental flow diagram of the program is given in figure 1. Each cycle or loop of its computation may be considered as beginning with a CHOICE operation when two or more statements (or clauses) from the list of clauses already in memory are chosen to make deductions from next. These clauses are given to one or both of the rules of inference, Resolution (Robinson 1965a) and the equality rule – Paramodulation (Wos and Robinson 1968) – depending on the way the user has things set up. Deductions following from the rules are passed on to separate editing operations associated with each of the rules, and then (if any remain) to a common editing operation. If a proof has been found, this is recovered by PROOF; otherwise the remaining new deductions are added to the lists by BOOK-KEEP, and the cycle is repeated. The computation may be interrupted any time a Book-Keep operation has just been completed, by the UPDATE routine which provides the standard interactive facilities (section 3).

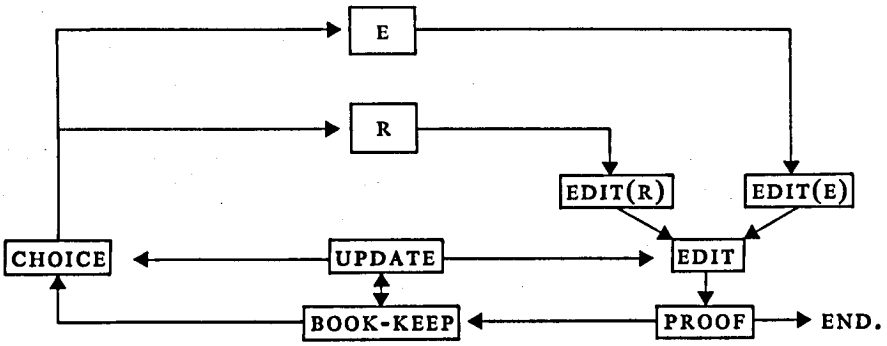



Figure 1

The strategies fall naturally into two classes, *choice* strategies and *editing* strategies, since they all operate either at the choice stage or the edit stage of the computation. At the moment, the editing strategies are, with one exception, the standard logical ones (see section 3). The principal choice strategies are refinements of the resolution principle in the sense of Luckham (1969) and they operate in the following way. Each refinement has associated with it a refining condition, $P(A, B)$, on pairs of clauses $A$, $B$ (more generally, $P$ operates on finite sets of clauses). A refinement allows only those resolvents $C$ of Clauses $A$ and $B$ satisfying $P$ to be generated. Thus,* if we let $R(A, B)$ denote the set of resolvents of $A$ and $B$, if $R^n(S)$ denotes the set of all resolvents of level $i \leqslant n$ that can be generated starting from the initial set $S$ of

* For further notation and terminology see Robinson (1965a) or Luckham (1968, 1969).

clauses, and if $\tilde{R}^n(S)$ denotes the subset of $R^n(S)$ that will be allowed by a refinement, we have,

$\tilde{R}^0(S) = S,$

$\tilde{R}^{n+1}(S) = \{C : C \in R(A, B) \;\&\; A, B \in \tilde{R}^n(S) \;\&\; P(A, B)\} \cup \tilde{R}^n(S).$ (1)

Trivially, $R^n(S)$ corresponds to the case where $P$ is always true. The program includes the following refinements.

**$\tilde{R}_1$: Resolution relative to a model**

Let $M$ be an interpretation or model over the domain of literals in the Herbrand expansion of $S$, $H(S)$ (*see* Luckham, 1969).

If each instance of clause $A$ contains a literal occurring in $M$, we say $M$ satisfies $A$ (notation, $M \models A$). Then $\tilde{R}_1$ is defined by (1) with the condition,

$$P_1(A, B) =_{df} M \not\models A \vee M \not\models B.$$

Under this refinement, only resolvents of pairs of clauses, at least one of which is false of the model $M$ are generated. The program allows any arbitrary LISP definition of a model to be given, but so far we have used only very simple models. The conditions, $P$, corresponding to some particularly simple (but nevertheless useful) models are programmed specially, e.g., the model consisting of all positive atoms, which yields the refinement called $P_1$-deduction in Robinson (1965b). The Set of Support (Wos, Robinson and Carson 1965) is a special case of $\tilde{R}_1$ (*see* Luckham 1969).

**$\tilde{R}_2$: Resolution with merging**

Andrews (1968) introduced the notion of a merge instance of a resolvent, $C$: if $C$ is a resolvent of $A$ and $B$, say $C = (A - \mathscr{L})\theta \cup (B - \mathscr{M})\theta$, and $\rho$ is a substitution (possibly empty) such that $(A - \mathscr{L})\theta\rho \cap (B - \mathscr{M})\theta\rho \neq \varnothing$, then $C\rho$ is a merge instance of $C$, (or simply, a *merge*). Resolution with merging, $\tilde{R}_2$ is the refinement obtained from (1) by substituting the condition,

$$P_2(A, B) =_{df} A \in S \vee B \in S \vee A \text{ is a merge } \vee B \text{ is a merge.}$$

Thus $\tilde{R}_2$ omits resolvents of pairs of non-merge resolvents, and generates only those resolvents that follow from at least one axiom or one merge. This refinement only restricts the deductions at level 2 and beyond, and does not cut down the number of level 1 deductions, so it is important to use it in conjunction with something that restricts level 1, such as set of support. Also, the property of being a merge is a property of the deduction tree of a clause; a clause may occur first as a merge and later on as a non-merge, for example. This leads to some complications in implementing $\tilde{R}_2$ if one wants to take full advantage of the refinement. What we have done in this program is to implement a computationally slick but less restrictive refinement which we refer to simply as 'merging'.

**$\tilde{R}_3$: Ancestry filter form**

This refinement, originally introduced independently by Loveland (1969) and Luckham (1969) is obtained from (1) by the condition.

$$P_3(A, B) =_{df} A \in S \vee B \in S \vee A \in Tr(B) \vee B \in Tr(A),$$

where $Tr(A)$ denotes the deduction tree of $A$ (cf. Andrews 1968, or Luckham 1969). Essentially, under this refinement, when it comes to choosing which clauses to resolve a clause $A$ against, the ancestry tree of $A$ is used to filter out the necessary clauses from the full list of all clauses in memory; $A$ is resolved only with those clauses $B$ that are axioms or occur in $Tr(A)$. The resulting proof trees generated in this way have a particularly simple structure (see figure 3) and are said to be in *Ancestry Filter Form* (AFF). The implementation of $\tilde{R}_3$ is very easy since it simply uses the mechanism for recovering proofs, which is already available in the program.

Each of these refinements has been proved to be logically complete. In fact, slightly stronger results are contained in the completeness proofs that have been given. Andrews (1968) gives a proof of the completeness of $\tilde{R}_2$ in conjunction with the set of support; the completeness proof for $\tilde{R}_1$ given in Luckham (1969) contains additional facts about the editing strategies; and proofs of the completeness of $\tilde{R}_3$ in conjunction with the set of support are given in Loveland (1969) and Luckham (1969). From a practical point of view, it is necessary to know as much as possible about the conditions under which the logical completeness of a refinement is maintained when it is used in conjunction with the standard editing strategies or with other refinements, for it is this knowledge that gives us some lines of possible further action when a proof search terminates without finding a proof. Essentially, the problem here is that one usually starts a proof search with a highly restrictive and logically incomplete combination of strategies. If this is unsuccessful, some of the parameters causing incompleteness are then relaxed and, of course, one has to know which these are. Some of the extra details that have proved useful may be summarized as follows.

Let us denote the strategy obtained by running the refinements $\tilde{R}_i$ and $\tilde{R}_j$ in conjunction, by $\tilde{R}_i \cap \tilde{R}_j$. It is proved in Luckham (1969) that both of the combinations $\tilde{R}_1 \cap \tilde{R}_2$ and $\tilde{R}_1 \cap \tilde{R}_3$ are *logically incomplete*. In other words, if one tries to extend the completeness results for $\tilde{R}_2$ or $\tilde{R}_3$ in conjunction with the set of support to the more restrictive refinement, $\tilde{R}_1$, one meets with failure. This is especially annoying since $\tilde{R}_1 \cap \tilde{R}_3$ has turned out to be quite useful (see section 4 and also example 5 in Luckham (1969)), and we do not have as yet any characterization of classes of problem for which this combination is complete. On the positive side, it has recently been proved by Kieburtz and Luckham (1969) that $\tilde{R}_2 \cap \tilde{R}_3$ is complete, and indeed a more restrictive strategy is also complete.

The interaction of the refinements with the editing strategies also needs careful scrutiny. For example, Loveland (1969) shows that if the initial set of hypotheses is not minimally inconsistent and the support set is chosen appropriately badly, then the conjunction of $\tilde{R}_3$ and set of support will generate a proof only if tautologies are admitted. It is also now well known that the various refinements will in general exclude the simplest proof trees from consideration, and admit proofs that are longer or contain longer

clauses or more complex functional terms. In this respect $\tilde{R}_1$ is well behaved. Corollaries of the proof of theorem 1 (Luckham 1969) are that $\tilde{R}_1$ remains complete with the elimination of tautologies, and that it does not increase the complexity of terms in the proof tree. The proof is also easily extended to show that $\tilde{R}_1$ remains complete when any clause which is subsumed by another is eliminated. Regarding $\tilde{R}_3$, it is shown by Kieburtz and Luckham (1969) that if the initial set of hypotheses satisfies some simple conditions, this refinement remains complete when both tautologies and clauses subsumed by an ancestor are eliminated. However, $\tilde{R}_3$ may increase the depth of nesting of function symbols in the terms occurring in the proof; that is, in order to find an $\tilde{R}_3$-proof from hypotheses $S$, the program may have to use terms more complex than those in the smallest set $K$ such that $K(S)$ is inconsistent.

The refinements $\tilde{R}_1$ and $\tilde{R}_3$ are also usefully employed in the system for logic with identity to control the number of deductions generated by the equality rule. Here, some completeness results are known, but there are still many open questions.

## 3. THE INTERACTIVE FACILITIES

In this section, without going into fine details, we will attempt to give the reader an idea of how the program appears to a user and the sort of facilities that may be called upon in the course of a proof search. The program is written in LISP and currently runs on the PDP-10 time-sharing system at the Stanford Artificial Intelligence Project. (With very little effort the program can be altered so as to be compatible with a LISP System available on IBM 360-series Machines.) The basic input–output device for on-line use is an Information International Inc. display scope and keyboard (henceforth called a console). Teletypes are also available, but these tend to be far too slow for the volume of information that this program puts out; they can be used when the program is run in a sort of batch-processing mode as a background job to the system. Other peripheral equipment usually required during a run includes a line printer for printed output, and the disk which is used as a permanent storage device for the program and problems, and as temporary storage for output and checkpoint situations when the user wishes to store uncompleted computations in order to experiment with other strategies and directions of proof.

When the user first sits down at the console to start a proof, the initial dialogue with the program assigns values to a sequence of program variables. This determines which strategies the program will begin with; the settings of these variables may be changed at any time during a proof search thus changing the strategies. The sequence of variables appearing on the console is usually the following.

1. *Set of support* (Wos, Robinson and Carson 1965). The user specifies which axioms are to be supported. To reject a strategy, the user types, 'NIL'.

2. *Unit Preference* (Wos, Robinson and Carson 1965). The user specifies a bound on the level or depth to which the preference strategy is allowed to look ahead.

3. $\tilde{R}_3$*: Ancestry Filter Form* (Luckham 1969). If AFF is not selected, the program uses a level saturation scheme to generate resolvents and paramodulants, in which the order of generation is determined by the order in which clauses appear on the clause lists.

4. $\tilde{R}_1$*: Model – relative deduction* (Luckham 1969). If the user chooses $\tilde{R}_1$, he is asked to define the model $M$.

5. $\tilde{R}_2$*: Merging* (Andrews 1968). As previously mentioned, the merging strategy is less restrictive than Andrew's 'Resolution with Merging', but is easier to program and its execution is much faster.

6. *Ordering.* The user can ask the program to order the predicate letters. This imposes an ordering on the literals. The resolvents generated from two clauses may then be restricted to those in which only the literal highest in the ordering is eliminated from one of the clauses.*

7. *Equality Rule* (*Paramodulation* (Wos and Robinson) 1969). If a proof involving the equality rule is to be attempted, the user is asked to volunteer some further information:

(a) Which predicate letter is to be interpreted as representing the equality relation;

(b) A bound on the depth of nesting of function symbols to which the matching test involved in computing paramodulants is to be applied;

(c) A list of unit clauses, each consisting of a positive equality atom (called the demodulation list), to be used in the operation of *demodulation* (Wos *et al.* 1967). All equality atoms, $A_i=B_i$, are ordered so that $A_i$ is at least as complex as $B_i$. The demodulation list, $\{A_i=B_i\}$ is then used as follows. If a clause $C$ contains a (well-formed) subexpression which is an alphabetic variant of some $A_i$ in this list, that subexpression is replaced by the corresponding variant of $B_i$. The procedure is iterated until no further variants of $A$'s are found in the modified $C$. At present, this operation is applied to paramodulants, and it is the final descendant (or modification) of $C$ that is retained (or more accurately, handed on to the other editing strategies). This user-controlled demodulation list has been a useful tool in reducing the number of trivial paramodulants.

The following editing strategies are also specified by the user at this time.

8. A bound on the *depth of nesting of function symbols* occurring in the terms of a clause. Any clause in which this bound is exceeded is rejected.

9. A bound on the *length* (number of literals) of a clause.

10. *Subsumptions.* The number of subsumption tests can be restricted in three ways. The test can be restricted to clauses of length less than a chosen bound, or so that only the newly generated clauses can be discarded (we call

* This is a variation of an idea of J.C.Reynolds.

this 'forward subsumptions') or so that it applies only to clauses of equal length (alphabetic variants). The implementation of the subsumption test is now sufficiently fast that our usual practice is not to restrict the test at all (except when AFF is employed; see section 2).

11. *Trivial Deductions.* The length-two clauses are treated specially. Let $\mathscr{L}$ be a list of unit clauses chosen by the user (e.g., a subset of the unit axioms is a common choice). Let $C$ be a two-clause of the form $\{\neg l_1, l_2\}$. If there is a substitution $\sigma_1$ such that $\{l_1, \sigma_1\} \in \mathscr{L}$ and $\{l_2 \sigma_1 \sigma_2\} \in \mathscr{L}$ for some further $\sigma_2$, then $C$ is rejected. We call this strategy, trivial deduction elimination. Note that the elimination of tautologies is a special case of this strategy if $\mathscr{L}$ is the set of all atoms in the Herbrand Universe. Since trivial deduction elimination is logically incomplete, we allow the user to choose the list $\mathscr{L}$. The strategy is an attempt to eliminate some of those clauses which, although not eliminable on purely logical grounds, are unlikely to contribute to a proof because the deductions following from them are already known. Obviously, we need more sophisticated (but fast) strategies to do this.

The value assigned to the final variable determines the amount of information the program gives the user about the deductions it makes; normally, the new resolvents and paramodulants are displayed on the scope as they occur, but this feature can be turned off. When the initial dialogue is completed and the strategies to be used have been chosen, the axioms and hypotheses are typed in from the console or read from a file on the disk, and the proof search begins. At any time, if a key is struck, the search will be interrupted and the program will wait for inquiries from the user. The current on-line command vocabulary allows the user the following options. (a) He can search the clause list, displaying each member either one at a time or in rapid succession, or he can display any particular clause if he knows its position in the list. (b) The ancestry tree (or proof) of any member of the clause list can be displayed. (c) The resolvents or paramodulants of any two chosen clauses can be computed and displayed. (d) Clauses can be deleted from or added to the clause list. (e) Finally, the user may enter a LISP READ–EVAL loop. In this state the settings of the strategy variables can be examined or redefined, the model (if $\tilde{R}_1$ is being used) can be changed, and in fact any arbitrary LISP computation (including one using the prover itself) can be executed.

Naturally, the user will attempt to find the most powerful combination of strategies for his problem. As a result of our own experience, we can make the following comments. The two most effective of our refinement strategies are $\tilde{R}_1$ and $\tilde{R}_3$. For almost all problems there seems to be a choice of model (and a very simple model) such that $\tilde{R}_1$ is of some significant help in getting a proof (*see also* Luckham 1968). $\tilde{R}_3$ has yielded some striking results. Although requiring the proof tree to satisfy the AFF condition almost always increases the ordinal depth of the tree by a significant factor (50 per cent is not uncommon), it has been the very simple examples with relatively short

proofs (level 5 or 6) that show unfavourable statistics for $\tilde{R}_3$. The AFF condition imposes a strong restriction on the growth of the number of deductions generated as a function of level. In many examples this seems to be linear. For more difficult problems it pays to search the extra levels with the AFF restriction. Even in cases where it turns out that more deductions are *retained* under the AFF condition, it often happens that far fewer deductions are *generated*, so that the deeper AFF proof-tree is obtained faster than the shorter trees because much less time is spent on editing computations. Also, many natural human proofs satisfy AFF; a point at which a resolvent of non-axiom clauses $A$ and $B$ is computed, and $A \in Tr(B)$, corresponds intuitively to an appeal to the lemma $A$. The combined strategy, $\tilde{R}_1 \cap \tilde{R}_3$, although incomplete, is very effective. If $\tilde{R}_3$ is used without $\tilde{R}_1$, it is important to add the set of support strategy. Merging, $\tilde{R}_2$, does not seem to be nearly as effective as the other two refinements, and it is almost always used in conjunction with at least one of them. The unit preference strategy appears to do more harm than good on the more difficult problems, usually because it leads to too many irrelevant clauses at the early levels. It is therefore used mainly to test for the end of a proof when the user is getting impatient. There are two 'end condition' strategies, which allow the user to search for unit proofs (unit preference) and vine-form proofs (i.e., proof trees in AFF and such that each deduction follows from an axiom). One way of using these strategies is to save the current state of the proof search, and then to introduce either strategy in an attempt to find a quick proof; if this is not successful, the user can continue with the original search. We hope to add some decision procedures for use in this way. Finally, the setting of the depth and length bounds (8 and 9 above) is usually crucial, since these essentially limit the subspace of the Herbrand expansion that will be searched for a proof. The user tends to start with these set as low as seems reasonable without excluding all proofs.

In practice we usually start a new problem with the $\tilde{R}_1 \cap \tilde{R}_2 \cap \tilde{R}_3$ combination, the model being chosen so as to generate a small number of level 1 deductions. As the search progresses, we try to keep track of any interesting deductions that may be relevant to the problem, and print their proofs on the line printer. These may be added as additional hypotheses when new searches are initiated. If a proof is found, the program displays it on the scope and stops. The search may fail to terminate successfully for three reasons. Too many clauses may be generated so that memory space is used up. This is unusual, but if it happens one generally tightens the bounds and tries again. The search may run out of time and the end condition strategies fail to produce a proof. If the maximal level searched is much larger than expected, this may mean that an incomplete combination of strategies is being used, so a different, possibly less restrictive, combination is tried. If, on the other hand, most of the time is spent in editing computations, one must use a more restrictive combination of strategies and bounds. Finally the program may stop

because it cannot make any more deductions. This case is treated as a sure sign of an incomplete strategy, and some of the bounds are relaxed or strategies turned off.

## 4. EXAMPLES OF PROOFS

In 1947 A.A.Grau (1947) presented a study of a ternary operation in Boolean Algebra, where 'ternary operation' was taken to mean a function of three variables defined on a set of elements and having values in the set. Grau defined the following algebraic system, $K$, which he called Ternary Boolean Algebra (TBA).

$K$ is a system consisting of a set $S$ and two operations under which the system is closed, one ternary, $(x\,y\,z)$, and the other unary, $x'$, satisfying the axioms,

$$(x\,y\,(u\,v\,w)) = ((x\,y\,u)\,v\,(x\,y\,w)) \quad (1)$$
$$(y\,x\,x) = x \quad (2)$$
$$(x\,y\,y') = x \quad (3)$$
$$(x\,x\,y) = x \quad (4)$$
$$(y'\,y\,x) = x. \quad (5)$$

$K$ is thus a homogeneous theory (all elements have equal significance) and has a realization within Boolean Algebra if the ternary operation, $(x\,y\,z)$, is interpreted as $(x \cap y) \cup (y \cap z) \cup (z \cap x)$. The axioms are therefore consistent. Furthermore, let $p$ be a fixed element of $S$, and define $x \cap y = df.\ (x\,p\,y)$, and $x \cup y =_{df.} (x\,p'\,y)$. Then the system $\langle S, \cap, \cup, ' \rangle$ forms a Boolean Algebra with $p$ as its null element and $p'$ as its universe element.

A recent abstract in the *Notices of the American Mathematical Society* (Chinthayamma 1969) announces without proof that Grau's axioms 1–3 are sufficient to define a TBA, and that they are also independent, and some new sets of axioms are given. The program has been used to establish all of the dependence results announced by Chinthayamma between the various axiomatizations. Let us consider the dependence results concerning Grau's axioms.* Establishing axiom 4 from 1–3 is straightforward, and it takes the program only a few seconds to display a proof,

$$\begin{aligned}(x\,x\,y) &= (x\,x\,(y\,x\,x')), \text{ axiom 3}\\ &= ((x\,x\,y)\,x\,(x\,x\,x')), \text{ axiom 1}\\ &= ((x\,x\,y)\,x\,x), \text{ axiom 3}\\ &= x, \text{ axiom 2.}\end{aligned}$$

(The proof is actually displayed in a formal notation–see figure 2.)
The dependence of axiom 5 on 1–4 is a bit more problematic and was first obtained with a certain amount of interaction. While the theorem-prover was chewing on the axioms with the strategies set so that it was actually attempting a proof by contradiction, the user was also trying the problem. He found that he could in fact prove the result on the assumption, $y\,x\,y' = x$,

* At this point the reader might care to try these problems before reading on.

*Axioms.* (Note that the disjunction connective is omitted from the clauses. Interpret P(X, Y, Z, W) as (X Y Z)=W, and C(X) as X′.)

1. P(x8,x2,x9,x7) ¬P(x4,x5,x3,x9) ¬P(x4,x5,x1,x8) ¬P(x4,x5,x6,x7) ¬P(x1,x2,x3,x6)
2. P(x4,x5,x6,x7) ¬P(x8,x2,x9,x7) ¬P(x4,x5,x3,x9) ¬P(x4,x5,x1,x8) ¬P(x1,x2,x3,x6)
3. P(x2,x1,x1,x1,)
4. P(x1,x2,C(x2),x1)
5. P(x1,x1,x2,x1)
6. ¬P(C(B),B,A,A) (Negation of Grau Axiom 5)

PROOF:

SUPPORT=NIL
UNIT-PREFERENCE=NIL
MERGE=T
ORDER=NIL
ANCESTRY=T
DEPTH-BOUND=1
LENGTH-BOUND=4
MODEL=T
NIL
(P)
PARAMODULATE=NIL

NIL 1 2
1. P(C(x1),x1,x15,x15) 3 4
2. ¬P(C(B),B,A,A) AXIOM 6
3. P(C(x13),x11,x2,x2) ¬P(C(x13),x11,x13,x13) 5 6
4. P(x2,x1,x1,x1,) AXIOM 3
5. P(x8,x11,x1,x1) ¬P(x1,C(x2),x10,x8) ¬P(x10,x11,x2,x2) 7 8
6. P(x2,x1,x1,x1) AXIOM 3
7. P(x8,x11,x1,x7) ¬P(x1,C(x2),x10,x8) ¬P(x1,C(x2),x6,x7) ¬P(x10,x11,x2,x6) 9 10
8. P(x1,C(x2),x2,x1) 11 12
9. P(x1,C(x2),x2,x1) 13 14
10. P(x8,x2,x9,x7) ¬P(x4,x5,x3,x9) ¬P(x4,x5,x1,x8) ¬P(x4,x5,x6,x7) ¬P(x1,x2,x3,x6) AXIOM 1
11. P(x1,C(x2),x2,x7) ¬P(x1,x2,C(x2),x7) 15 16
12. P(x1,x2,C(x2),x1) AXIOM 4
13. P(x1,C,(x2),x2,x7) ¬P(x1,x2,C(x2),x7) 17 18
14. P(x1,x2,C(x2),x1) AXIOM 4
15. P(x8,x10,x1,x7) ¬P(x2,x1,x10,x8) ¬P(x2,x1,x10,x7) 19 20
16. P(x1,x2,C(x2),x1) AXIOM 4
17. P(x8,x10,x1,x7) ¬P(x2,x1,x10,x8) ¬P(x2,x1,x10,x7) 21 22
18. P(x1,x2,C(x2),x1) AXIOM 4
19. P(x8,x1,x9,x7) ¬P(x4,x5,x2,x9) ¬P(x4,x5,x1,x8) ¬P(x4,x5,x1,x7) 23 24
20. P(x2,x1,x1,x1,) AXIOM 3
21. P(x8,x1,x9,x7) ¬P(x4,x5,x2,x9) ¬P(x4,x5,x1,x8) ¬P(x4,x5,x1,x7) 25 26
22. P(x2,x1,x1,x1) AXIOM 3
23. P(x1,x1,x2,x1) AXIOM 5
24. P(x8,x2,x9,x7) ¬P(x4,x5,x3,x9) ¬P(x4,x5,x1,x8) ¬P(x4,x5,x6,x7) ¬P(x1,x2,x3,x6) AXIOM 1
25. P(x1,x1,x2,x1) AXIOM 5
26. P(x8,x2,x9,x7) ¬P(x4,x5,x3,x9) ¬(Px4,x5,x1,x8) ¬P(x4,x5,x6,x7) ¬P(x1,x2,x3,x6) AXIOM 1

QED

Figure 2

and could prove this if $x\,y'\,y=x$ were true. A quick check of the clause list found both $y\,x\,y'\neq x$ and $x\,y'\,y\neq x$ derived from the negation of axiom 5 (in other words, it had come to the same conclusions). The program was then given the two assumptions as well as axioms 1–4 and found a proof using only one of the assumptions, $x\,y'\,y=x$. This at least gave the user an alternative 'sub-goal'. However, an examination of this proof showed length and depth bounds of 4 and 1 respectively to be sufficient. Re-setting these parameters and attempting the original problem yielded a proof in 8 minutes; a translation into natural notation is the following.

| | |
|---|---|
| $x=(x\,y\,y')$ | axiom 3, |
| $=(x\,y\,(y'\,y'\,y))$ | axiom 4, |
| $=((x\,y\,y')\,y'\,(x\,y\,y))$ | axiom 1, |
| $=(x\,y'\,y)$, | axioms 3 and 2, (lemma) |
| $=(x\,y'\,(y'\,y\,y))$ | axiom 2, |
| $=((x\,y'\,y')\,y\,(x\,y'\,y))$ | axiom 1, |
| $=(y'\,y\,x)$ | axiom 2 and lemma. |

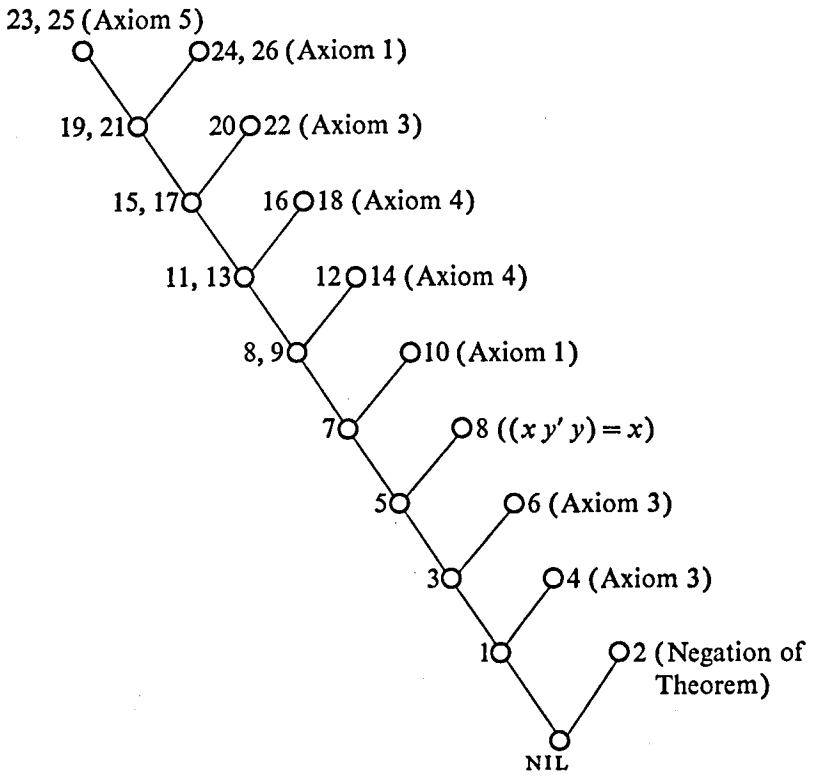



Figure 3. AFF proof-tree of proof in figure 2

These appear to us to be exactly the sort of problem mathematicians would much prefer to have machines to do. After all, it seems to have taken twenty years for these dependencies to become common knowledge, which is certainly a monument to the reluctance of people to negotiate such puzzles.

The formal proof of the second problem is given in figure 2, and its AFF proof tree is drawn in figure 3 (notice that the proof recovery outputs duplicate copies of isomorphic sub-trees; in figure 2, $Tr(8)=Tr(9)$).

Figure 4 contains some statistics comparing $\tilde{R}_1$ and $\tilde{R}_3$ for proofs of the lemma $x\,y'\,y=x$, from Grau's axioms 1–4. Here all strategies not mentioned were held constant at the settings indicated in figure 2; in each case when $\tilde{R}_1$ was used, the model was the set of all negative literals, and the set of support, when used, was equivalent to using this model at level 1. In all of these TBA problems, the Ancestry Filter proved to be a very powerful strategy.

| | *Clauses* | | | |
|---|---|---|---|---|
| *Choice Strategies* | *Retained* | *Generated* | *Time* | *Proof Level* |
| $\tilde{R}_1 \cap \tilde{R}_2 \cap \tilde{R}_3$ | 153 | 711 | 174 sec. | 5 |
| $\tilde{R}_2 \cap \tilde{R}_3 \cap$ (Set of Support) | 185 | 2244 | 280 sec. | 4 |
| (No Support) $\tilde{R}_2 \cap \tilde{R}_3$ | 185 | 2632 | 369 sec. | 4 |
| $\tilde{R}_1 \cap \tilde{R}_2$ | 236 | 17,289 | 30 min. | 4 (No Proof, run stopped) |

Figure 4

Finally, figure 5 is an example of a proof using both rules of inference. The problem is the following:

> Let $K$ be a system consisting of a set $S$ and a binary operation defined on $S$. If $K$ is closed and associative and has an element $e$ such that $e^2=e$, and every element $S$ has a left inverse with respect to $e$ and at most one right inverse with respect to $e$, then $K$ is a group.*

The resolvents in the proof are marked, '$R$', and the paramodulants are marked, '$E$', or '$E$, $D$' if they result from demodulation as well. We have used elementary algebra problems of this type (e.g., those considered in Wos, Robinson and Carson (1967) and Guard *et al.* (1969)) to compare the performances of our strategies on formulations in pure logic and in the first order theory of equality. Initial results suggest that the equality formulation proofs are shorter and more concise, but take longer to generate because of the amount of editing necessary on the paramodulants.

* This problem was suggested by Dr. L. Wos.

*Axioms.* (Interpret R(X,Y) as X=Y, F(X,Y) as the binary operation, and G(X) as the left inverse of X with respect to E).

1. R(F(F(X1,X2),X3),F(X1,F(X2,X3)))
2. R(F(E,E),E)
3. R(F(G(X1),X1)E)
4. R(X1,X1)
5. R(X2,X3) ¬R(F(X1,X3),E) ¬R(F(X1,X2),E)
6. ¬R(F(E,A),A)

SUPPORT=NIL
UNIT-PREFERENCE=NIL
MERGE=T
ORDER=NIL
ANCESTRY=T
DEPTH-BOUND=2
LENGTH-BOUND=2
MODEL=2
NIL
(R)
PARMODULATE=T
EQUAL-SYMBOL=R
PAR-DEPTH-BOUND=2

PROOF:

NIL 1 2
1. R(F(E,X2),X2) 3,4 E
2. ¬R(F(E,A),A)AXIOM 6
3. R(F(E,X2),F(X2,E)) 5,6 R
4. R(X1,F(X1,E)) 7,8 R
5. R(F(G(X2),F(E,X2)),E) 9,10 E,D(AXIOM 1)
6. R(X2,F(X1,E)) ¬R(F(G(X1),X2),E) 11,12 R
7. R(X2,F(X1,E)) ¬R(F(G(X1),X2),E) 13,14 R
8. R(F(G(X1),X1),E) AXIOM 3
9. R(X1,F(X1,E)) 15,16 R
10. R(F(G(X1),X1),E) AXIOM 3
11. R(F(G(X1),F(X1,E)),E) 17,18 E
12. R(X2,X3) ¬R(F(X1,X3),E) ¬(F(X1,X2),E) AXIOM 5
13. R(F(G(X1),F(X1,E)),E) 19,20 E
14. R(X2,X3) ¬R(F(X1,X3),E) ¬R(F(X1,X2),E) AXIOM 5
15. R(X2,F(X1,E)) ¬R(F(G(X1),X2),E) 21,22 R
16. R(F(G(X1),X1),E) AXIOM 3
17. R(F(G(X1),F(X1,X3)),F(E,X3)) 23,24 E
18. R(F(E,E),E) AXIOM 2
19. R(F(G(X1),F(X1,X3)),F(E,X3)) 25,26 E
20. R(F(E,E),E) AXIOM 2
21. R(F(G(X1),F(X1,E)),E) 27,28 E
22. R(X2,X3) ¬R(F(X1,X3),E) ¬R(F(X1,X2),E) AXIOM 5
23. R(F(G(X1),X1),E) AXIOM 3
24. R(F(F(X1,X2),X3),F(X1,F(X2,X3))) AXIOM 1
25. R(F(G(X1),X1),E) AXIOM 3
26. R(F(F(X1,X2),X3),F(X1,F(X2,X3))) AXIOM 1
27. R(F(G(X1),F(X1,X3)),F(E,X3)) 29,30 E
28. R(F(E,E),E) AXIOM 2
29. R(F(G(X1),X1),E) AXIOM 3
30. R(F(F(X1,X2),X3),F(X1,F(X2,X3))) AXIOM 1

QED

Figure 5

**Acknowledgement**

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Department of Defense (SD–183).

## REFERENCES

Andrews, P.B. (1968) Resolution with merging. *J. Ass. comput. Mach.*, **15**, 367–81.

Chinthayamma (1969) Sets of independent axioms for a ternary Boolean algebra. *Notices of Amer. Math. Soc.*. **16**, 69T–A69, 654.

Grau, A.A. (1947) Ternary Boolean Algebra. *Bull. Amer. Math. Soc.*, **53**, 567–72.

Green, C. & Raphael, B. (1968) The use of theorem-proving techniques in question–answering, *Proc. 23rd National Conference ACM*. Washington, D.C.: Thompson Book Co.

Guard, J.R., Oglesby, F.C., Bennett, J.H. & Settle, L.G. (1969) Semi-automated mathematics. *J. Ass. comput. Mach.*, **16**, 49–62.

Kieburtz, R. & Luckham, D. (1969) *Compatibility of Refinements of the Resolution Principle* (in press).

Loveland, D. (1969) A linear format for resolution. *Proceedings IRIA Symposium on Automatic Demonstration.* Versailles, France, December 1968. Springer-Verlag (in press).

Luckham, D. (1967) The resolution principle in theorem-proving. *Machine Intelligence 1*, (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Luckham, D. (1968) Some tree-paring strategies for theorem-proving. *Machine Intelligence 3*, pp. 95–112 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Luckham, D. (1969) Refinement theorems in resolution theory. *Proceedings IRIA Symposium on Automatic Demonstration.* Versailles, France, December 1968. Springer-Verlag (in press).

Luckham, D. & Nilsson, N. (1969) *On extracting information from resolution proof trees.* Stanford Artificial Intelligence Project Memo (in 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. *International Journal of computer Mathematics*, **1**, 227–34.

Wos, L., Robinson, G., & Carson, D. (1965) Efficiency and completeness of the set of support strategy in theorem-proving. *J. Ass. comput. Mach.*, **12**, 536–41.

Wos, L., Robinson, G., Carson, D. & Shalla, L. (1967) The concept of demodulation in theorem-proving. *J. Ass. comput. Mach.*, **14**, 698–704.

Wos, L. & Robinson, G. (1969) Paramodulation and set of support. *Proceedings IRIA Symposium on Automatic Demonstration.* Versailles, France, December, 1968, Springer-Verlag (in press).

# 18

# A Symbol Manipulation System

F. V. McBride,
D. J. T. Morrison and
R. M. Pengelly
Department of Computer Science
The Queen's University of Belfast

## 1. INTRODUCTION

This paper describes an online symbol manipulation system based upon extensions to the LISP 1.5 programming language. The aim of the project is to develop a general purpose symbol manipulation system incorporating features to enable the user to modify and extend the manipulative capabilities of the basic processor. These requirements can be met only by the use of a low-level language (LISP 1.5 was a convenient choice in this instance) in which the user can apply the primitive facilities provided, to the construction of the particular capability required. Since the provision of an 'arbitrarily complex definitional capability' within the language seems to necessarily imply a high degree of interpretive execution (cf. the FAMOUS system of Fenichel (1966)) the system has been based upon an existing LISP 1.5 interpreter.

The system has been implemented by adding a matching algorithm to the standard LISP interpreter and by introducing a slightly altered syntax into the LISP 1.5 language. The advantage of this approach is that the new facilities provide the user with a more 'natural' language for describing his manipulative processes while retaining the full processing power of standard LISP. Two versions of the matching system are described, one in section 2 and the other in section 3, and their operation is illustrated with examples.

The extended LISP system has been embedded within an interactive environment. This enables the user to guide the manipulative processes being performed and use the definitional facilities to extend or modify the system, if necessary. The fundamental importance of the interactive capability arises from the fact that no program can currently match the power of its human user to recognize 'significant' or 'useful' items of information. The system environment is described in section 4.

## 2. INTRODUCTION OF A MATCHING ALGORITHM INTO LISP

### 2.1. Introduction

The evaluation process of standard LISP is associated with the functional form:

$$G(e) = (p_1(e) \rightarrow s_1(e); p_2(e) \rightarrow s_2(e); \ldots p_n(e) \rightarrow s_n(e))$$

where the $p_i (i=1 \text{ to } n)$ are predicates and the $s_i (i=1 \text{ to } n)$ are general expressions. The value of such a function is the value of the substitute $s_i$ corresponding to the leftmost true $p_i$ encountered by the interpreter (if such a $p_i$ exists).

In the design of many symbol manipulation algorithms, the construction of predicates may be divided into two distinct phases: first the isolation of specific structures and second the application of constraints to the constituent elements of these structures. In standard LISP these distinct operations must be represented by either a single complex predicate or a chain of simple predicates. The description of these processes is more natural if a suitable form-matching system is introduced to isolate structures, along with associated predicates to describe the elemental constraints. The resulting evaluation process might then be represented by a functional of the form:

$$\begin{aligned} G(e) = (&f_1 \textit{ matches } e \wedge p_1(e) \rightarrow s_1(e); \\ &f_2 \textit{ matches } e \wedge p_2(e) \rightarrow s_2(e); \\ &\cdot \\ &\cdot \\ &\cdot \\ &f_n \textit{ matches } e \wedge p_n(e) \rightarrow s_n(e)) \end{aligned}$$

where the $f_i (i=1 \text{ to } n)$ are dummy forms. The value of such a function is the value of the substitute $s_i$ corresponding to the leftmost successful match (between the form $f_i$ and the argument expression $e$) with an associated true predicate.

### 2.2. A matching algorithm

The matching algorithm introduced, which is similar to one used by Fenichel (1966), is as follows. If a form $f$ is to match an expression $e$, then:

(a) if $f$ is a number, the name of a defined function or the name of a constant, then $e$ must be identical to $f$.

(b) any atomic form, apart from those described in (a), matches any expression:

e.g., $f=x$ matches $e=x, y, 4, x+y \log 4$ and all others.

(c) if $f$ is a quotation of another form $g$, then the expression $e$ must be identical to $g$:

e.g., $f={}^{u}x$ matches only $e=x$.

(*d*) if $f$ consists of a list of elements $f_1, f_2, \ldots, f_n$, then

(i) $e$ must consist of a list of elements $e_1, e_2, \ldots, e_n$,

(ii) for $i=1, 2, \ldots, n, f_i$ must match $e_i$, and

(iii) if $g$ is a name which occurs more than once in $f$, then the corresponding subexpressions in $e$ must be identical:

e.g., $f=(u\ v\ w)$ matches $e=(x\ y\ z)$
but $f=(u\ u)$ does *not* match $e=(x\ y)$;
however $f=(u\ v\ w)$ matches $e=(x\ x\ x)$.

In general a user will require numbers, defined functions, and constants to match only themselves; this is covered by (a), which is equivalent to implicit quotation of these atom types. The 'automatic' matching process described in (b) for free atoms is directly analogous to the pairing operation between the LAMBDA variable list and the argument list in the evaluation of EXPRS in standard LISP.

(d) implies that a left-to-right scan is employed; this precludes the necessity of compounding *car*s and *cdr*s to isolate elements of argument expression which is a cumbersome feature of standard LISP. (d) also expresses the fact that variable names in the form are unique, a feature which is essential in order to prevent ambiguity arising during the match.

### 2.3. Rules

To facilitate the use of the matching algorithm in the LISP interpreter, a new type of function has been introduced, characterized by the indicator RULE on the property list of an atom. Its *S*-expression definition has the form:

(DARG(D1 D2 . . . DN)
A1(ASSERTION 1)
A2(ASSERTION 2)
.
.
.
AM(ASSERTION M))

where the atom DARG serves a similar function for RULES as LAMBDA does for EXPRS. (D1 D2 . . . DN) is a list of dummy arguments; A1, A2, . . ., AM are called assertion labels and are present primarily to ease the burden of amendment (see section 4.2).

The format of assertion is

(form $f$, substitute $s$, predicate $p$)

that is, a three-element list. (This may be compared with the format of a 'single rule' as described by Fenichel (1966).) The predicate has been positioned at the end of the list so that a two-element list can be taken to denote an implicitly true predicate.

The matching process consists of comparing the argument expression with the form of each assertion taken in sequence, according to the matching

algorithm described in section 2.2. If a successful form-match is obtained, the form elements are bound to the corresponding elements in the argument expression and the bindings appended to an association list (*a*-list).

e.g., a form (+AB) matches an expression (+(* 4 C)(↑ T 2)) and produces bindings (A.(* 4 C)) and (B.(↑ T 2)).

Then, if the associated predicate evaluated with respect to the newly-created *a*-list is true, the value of the RULE is the corresponding substitute evaluated with respect to the *a*-list. Either of these evaluations may involve recursive re-entry into the matching system.

As far as the evaluation functions of the LISP interpreter are concerned, RULES are treated in a similar manner to EXPRS with the atom DARG serving the same purpose as LAMBDA does for EXPRS. However, whereas EXPRS are evaluated with respect to an *a*-list created by pairing the list of dummy variables and the argument list, the corresponding *a*-list for RULES is created by the matching process described above. This pairing process in the evaluation of EXPRS may be regarded as an automatic matching procedure. If the form of *each* assertion of a RULE is made identical to the list of dummy variables of an EXPR with corresponding substitutes and predicates, it is evident that this automatic matching process is reproduced in a RULE: this provides a simple method of transcribing an EXPR into a RULE.

### 2.4. Illustration of the matching system

As an illustration of the descriptive capability and the operation of the system, consider the construction and operation of a simple differentiator, which incorporates the following relations:

$$\frac{dn}{dx}=0 \text{ when } n \text{ is a number} \tag{1}$$

$$\frac{dx}{dx}=1 \tag{2}$$

$$\frac{d}{dx}(u+v)=\frac{du}{dx}+\frac{dv}{dx} \tag{3}$$

$$\frac{d}{dx}(u*v)=u*\frac{dv}{dx}+v*\frac{du}{dx} \tag{4}$$

$$\frac{d}{dx}(-u)=-\frac{du}{dx} \tag{5}$$

These relations may be expressed as the assertions of a RULE D. The transcription process is very straightforward and yields:

D1: (N X)→0 *when* (NUMBERP N)
D2: (X X)→1
D3: ((+ U V)X)→(+(D U X)(D V X))
D4: ((*U V)X)→(+(* U(D V X))(* V(D U X)))
D5: ((− U)X)→(−(D U X))

provided that +, * and − are defined functions; this ensures that the atoms +, * and − are regarded as quoted by the matching algorithm. Suppose that +, * and − are 'dummy' RULES defined with single assertions:

LAST: (A B)→(LIST ("+) A B),
LAST: (A B)→(LIST ("*) A B)
and LAST: (A)→(LIST ("−)A)

respectively. These definitions are used to evaluate the substitutes in D.

In the following the evaluation of D((+(−(*7Z)) 3)Z) (that is, (d/dz) (−7z+3)) is demonstrated by presenting the argument expression at each entry to D and the value at each exit. Level numbers in parentheses serve to link corresponding entries and exits to D; the label present opposite an argument expression denotes the assertion in D where the expression has been matched.

```
ARGUMENTS(1): ((+(−(* 7 Z))3)Z)        D3
ARGUMENTS(2): ((−(* 7 Z))Z)            D5
ARGUMENTS(3): ((* 7 Z)Z)               D4
ARGUMENTS(4): (Z Z)                    D2
VALUE(4):     1
ARGUMENTS(4): (7 Z)                    D1
VALUE(4):     0
VALUE(3):     (+(* 7 1)(* Z 0))
VALUE(2):     (−(+(* 7 1)(* Z 0)))
ARGUMENTS(2): (3 Z)                    D1
VALUE(2):     0
VALUE(1):     (+(−(+(* 7 1)(* Z 0)))0)
```

Since the RULES + and * are applied in the evaluation of the substitutes in D, the construction of a primitive simplifier to evaluate subexpressions such as (* Z 0) in the above result merely involves the addition of the assertions:

AP1: (A 0)→A

to + and

AS1: (A 0)→0
AS2: (A 1)→A

to *. Then if D((+(−(* 7 Z))3)Z) is evaluated the result is (−7).

## 3. AN EXTENDED MATCHING SYSTEM

### 3.1. Introduction

The matching system described in section 2 employs a left-to-right, one-to-one, ordered matching algorithm. In certain contexts, predominantly those involving arithmetic operators, this system requires the definition of a rather cumbersome set of assertions for solving a simple matching problem.

For example, consider a function LINEAR (X E) which is to determine if

E is linear with respect to X. (For this illustration the only arithmetic operators involved in E are + and *). In terms of the matching system described in section 2, the forms and predicates required may be written as

A1: (X X)
A2: (X(* A X)) *when* (FREE A X)
A3: (X(* X A)) *when* (FREE A X)
A4: (X(+ X B)) *when* (FREE B X)
A5: (X(+ B X)) *when* (FREE B X)
A6: (X(+(* A X)B)) *when* (FREE A X) ∧ (FREE B X)
A7: (X(+B(* A X))) *when* (FREE A X) ∧ (FREE B X)
A8: (X(+(* X A)B)) *when* (FREE A X( ∧ (FREE B X)
A9: (X(+B(* X A))) *when* (FREE A X) ∧ (FREE B X)

It is apparent that any one of the last four forms would be sufficient to describe the solution, providing a mechanism is available to inform the system of the commutivity of the operators + and * and the equivalence of A, (+ A 0) and (* A 1). This can be achieved by the introduction of a mechanism to reconstitute the argument of a RULE in an equivalent form. It is not possible to provide the required information in 'assertional' form associated with the RULES + and * because such information is processed only during the evaluation of the RULES + and *. Furthermore the evaluation of substitutes of assertions of the form (+ A B)→(+ B A) causes the RULE + to be re-entered and may therefore lead to infinite recursion.

### 3.2. Transformations

In the present attempt to make the matching system more sophisticated, a new type of entity (which will be referred to as a transformation) has been introduced into RULES, which now have the s-expression form:

(DARG (D1 D2 . . . DN)
(A1 (ASSERTION 1)
.
.
.
AMA (ASSERTION MA))
(T1 (TRANSFORMATION 1)
.
.
.
(TMT (TRANSFORMATION MT)))

where a transformation has the same format as an assertion (see section 2.3).

The general strategy which the matching system employs for processing 'transformational' information is as follows. As before, the left-to-right matching algorithm described in section 2.2 is used; if the match fails, however, an attempt is made to reconstitute the argument subexpression at the

level of failure, by applying the transformations of the RULE governing the corresponding subexpression of the form. If a 'suitable' reconstitution is obtained, the left-to-right scan is recommenced; however, if neither a 'suitable' reconstruction nor a successful match after reconstitution is possible, the transformation process is attempted at the next higher structural level. This procedure is followed until the top level is reached and if a successful match is not possible at this level the next assertion is considered. (A reconstitution is regarded as 'suitable' if there is a left-to-right match between the form of the transformation being applied and the relevant argument subexpression, and the associated predicate is true.)

The necessity to determine the subexpression level at which a match failure occurs requires that the assertional predicates be evaluated before a variable binding is appended to the *a*-list. For the matching system described in section 2, the timing of predicate evaluation is inconsequential because both the matching process and the predicate evaluation are pass or fail tests and, if either fail, the evaluation process continues with the next assertion. However for this extended matching system the timing is critical since the point of failure determines at which subexpression level reconstitution attempts are to commence.

### 3.3. An illustration of the extended matching system

The function LINEAR (X E) introduced in section 3.1 may be written in the new RULE format with a single assertion:

(X(+(* A X)B)) *when* (FREE A X)∧(FREE B X)

providing the RULES + and * have transformations:

TP1: (+ A B)≡(+ B A)
TP2: A≡(+ A 0)
TP3: A≡(+ 0 A)

and TS1: (* A B)≡(* B A)
TS2: A≡(* A 1)
TS3: A≡(* 1 A) respectively.

In the evaluation of LINEAR (Z Z) the sequence of matching operations is as follows.

Attempt to match:

(I) $f$=(X(+(* A X)B)); $e$=(Z Z).

Attempt to match the first elements of (I):

(II) $f$=X; $e$=Z.

Successful match: (X . Z) appended to *a*-list.

Attempt to match the second elements of (I):

(III) $f$=(+(* A X)B); $e$=Z.

Unsuccessful match: reconstitute $e$ by applying +'s transformations. TP1 is not applicable since (+ A B) does not match Z; however, TP2 reconstitutes Z as (+ Z 0).

Attempt to match:

(IV) $f=(+(*\ \text{A}\ \text{X})\text{B})$; $e=(+\ \text{Z}\ 0)$.

Attempt to match the first elements of (IV):

(V) $f=+$; $e=+$.

Successful match.

Attempt to match the second elements of (IV):

(VI) $f=(*\ \text{A}\ \text{X})$; $e=\text{Z}$.

Unsuccessful match: reconstitute $e$ by applying *'s transformations. The first suitable transformation is TS2 which reconstitutes Z as (* Z 1).

Attempt to match:

(VII) $f=(*\ \text{A}\ \text{X})$; $e=(*\ \text{Z}\ 1)$.

Attempt to match the first elements of (VII):

(VIII) $f=*$; $e=*$.

Successful match.

Attempt to match the second elements of (VII):

(IX) $f=\text{A}$; $e=\text{Z}$.

Unsuccessful match since the predicate associated with A is false: reconstitute $e$ in (VI) by applying TS3, the next ('suitable') transformation of *; hence Z becomes (* 1 Z).

Attempt to match:

(X) $f=(*\ \text{A}\ \text{X})$; $e=(*\ 1\ \text{Z})$.

Successful match since the predicate associated with A is true and X is already bound to Z: (A . 1) is appended to the $a$-list.

Attempt to match the second elements of (IV):

(XI) $f=\text{B}$; $e=0$.

Successful match since the predicate associated with B is true: (B . 0) appended to the $a$-list.

Therefore match (I) is successful and the $a$-list formed is

((B . 0)(A . 1)(X . Z)).

A further example of a function LINEAR (X E) in the extended RULE format is presented in Appendix A.

## 4. THE SYSTEM ENVIRONMENT

### 4.1. Introduction

All the defining and processing facilities of the standard LISP interpreter are available to the user in addition to the RULE processing facilities described in sections 2 and 3. The basic action of the LISP interpreter is to read a unit of information (i.e., either a single s-expression or a pair of s-expressions terminated by a $) to execute it and to return for further input. No attempt has been made at this stage to design and implement a more sophisticated input/output language.

The remainder of section 4 is devoted to a brief description of additional

functions which have been introduced to make the system more useful, particularly in an interactive environment. The operation of some of these functions is illustrated in Appendix B.

### 4.2. RULE definition and amendment

RULEs may be defined by the function DEFRULES, which is the exact analogue of the EXPR definition function DEFINE. Both RULE and EXPR definitions may be saved by the user on backing-store. Amendment of RULEs by the addition or deletion of an assertion or a transformation is also possible (see 2.4); assertion and transformation labels are used to specify the location of the insertion or deletion in the RULE body.

### 4.3. Variable definition

Facilities have been introduced for the definition of both local and global variables. Local variable bindings persist for the duration of the execution cycle in which they are defined whereas global bindings persist throughout a program run or until they are destroyed by user intervention. Global variable definitions can be saved on backing-store. Examples of variable definition are given in Appendix B.

### 4.4. Interactive facilities

An essential element in the system is the degree of control which the user may exercise over RULE evaluation by the introduction of programmed halts. For example, these facilities allow the user to input substitutes or predicates in the form of s-expressions during RULE evaluation, or to suspend the evaluation of a RULE while any number of other executions (such as defining or editing) are performed. In addition facilities have introduced into the interpreter evaluating functions to cope with problems arising from the absence (in core) of function definitions and the mismatching which may occur because of the consequent non-recognition of functions by the matching system; these are illustrated in Appendix B.

## REFERENCE

Fenichel, R.R. (1966) An online system for algebraic manipulation. Report MAC-TR-35, M.I.T.

## APPENDIX A

The RULE LINEAR (X E) described in section 3.3 has been extended to deal with association and distribution; a special feature of this RULE is that evaluation of predicates may involve recursive re-entry into the RULE. In the following the RULES +, * and LINEAR are defined and the procedure involved in testing the linearity of $ax+b+cx+d$ with respect to $x$ is illustrated by presenting the argument expression for LINEAR at each entry and the value of LINEAR at each exit (see section 2.4).

```
        DEFRULES((
        (+(DARG(A B)
        (AP1((A B)(LIST("+)A B))
        (TP1((+ A B)(("+)B A)))))
        (*(DARG(A B)
        (AS1((A B)(LIST("*)A B))
        (TS1((* A B)(("*)B A))
        TS2(A(("*)1 A))
        TS3(A(("*)A 1)))))
        (LINEAR(DARG(X E)
        (L1((X E) F ((E.(FREE E X))))
        L2((X(* A X)) T ((A.(FREE A X))))
        L3((X(+ A B)) T ((A.(LINEAR X A))
              (B.(OR(FREE B X)(LINEAR X B)))))
        L4((X(* A B)) T ((A.(LINEAR X A))
              (B.(FREE B X)))))
        NIL))))$
(+ * LINEAR)

                        LINEAR(X(+(* A X)(+ B(+(* C X)D)))) $
        ARGUMENTS(1): (X(+(* A X)(+ B(+(* C X)D)))) L3
        ARGUMENTS(2): (X(* A X)) L2
        VALUE(2):     T
        ARGUMENTS(2): (X(+ B(+(* C X)D))) L3
        ARGUMENTS(3): (X B) L1
        VALUE(3):     F
        ARGUMENTS(3): (X(+ (* C X)D)) L3
        ARGUMENTS(4): (X(* C X)) L1
        VALUE(4):     T
        VALUE(3):     T
        VALUE(2):     T
        VALUE(1):     T.
```

## APPENDIX B

The following provides an illustration of the operation of some of the system functions described in section 4. Comments have been inserted where necessary to aid understanding. Note that the disk-file LDF holds RULE definitions of +, −, *, /, ↑ and D before the run commences; the exact form of these definitions is unimportant for the purposes of this illustration. Lines which are indented and terminated by a $ are input by the user.

```
        GETFILE(LDF)$
        RESTORE((+ - */↑)LDF)$
```

Input RULE definitions from file LDF.

```
        IDENT(Y(+(/(↑ X 2)F)F))$
```

Bind second argument of IDENT to first.

```
        IDENTQ(D1Y(D Y X))$
```

Bind evaluated second argument of IDENTQ to first.

```
  NO DEFINITION FOR D
        RESTORE((D)LDF)$
        RESTART S
  (+(/(-(* F(* 2 X))(*(↑ X 2)(D F X)))(↑ F 2))
  (D F X))
        REMOVE((↑))$
```

Remove definition of RULE ↑.

```
        IDENTQ(D1Y(D Y X))$
  IS THE FOLLOWING A VALID BINDING ?
  (↑ . +)
        NO$
  NO DEFINITION FOR ↑
        RESTORE((↑)LDF)$
        RESTART S
  (+(/(-(* F(* 2 X))(*(↑ X 2)(D F X)))(↑ F 2))
  (D F X)).
```

# COGNITIVE PROCESSES: METHODS AND MODELS

# 19

# Associative Memory Models

D. J. Willshaw and
H. C. Longuet-Higgins
Department of Machine Intelligence and Perception
University of Edinburgh

## 1. INTRODUCTION

In a recent article in *Nature* written in collaboration with O.P. Buneman (Willshaw, Buneman and Longuet-Higgins, 1969) we described two quasi-holographic devices for the associative storage and retrieval of complex patterns. These devices, the correlograph and the associative net, serve many of the same purposes as the holograph, but are simpler in conception and lend themselves more easily to computer simulation. Our interest in them is twofold: first, it is quite possible that the principles on which they work are employed in the central nervous system; and secondly, they may well find application in computing technology, especially when parallel computation techniques become generally available.

## 2. THE OPTICAL CORRELOGRAPH

The correlograph was originally envisaged as an optical analogue device, illustrated in figures 1 and 2. It is designed for storing the correlation C between a pair of patterns A and B in such a way that A can be retrieved from B and C, or B retrieved from A and C. A and B are patterns of pinholes through black cards. When A is illuminated from the left, rays pass through the holes in A and B and are guided by a lens L on to a screen at C. (The distances AB and LC are both set equal to the focal length of L.) A 'correlogram' is then made by taking a black card and making a pinhole through it at every point at which a ray strikes the viewing screen. This correlogram is mounted at C and illuminated from the right, so that rays now pass through the holes in C and the holes in B. A pattern of bright spots now appears at A, and the brightest of these spots coincide with the pinholes of the original pattern A. A can therefore be reconstructed by sifting out the brightest spots with a threshold detector. (Also, though this need not concern us, B can be reconstructed by turning C upside down, placing A where B was before, illuminating from the

right and mounting the threshold detector where A was before. The reader may care to satisfy himself of the truth of this assertion.)

A further use of the correlograph is for detecting the presence of a particular configuration of pinholes in the pattern B. In this application the pinhole

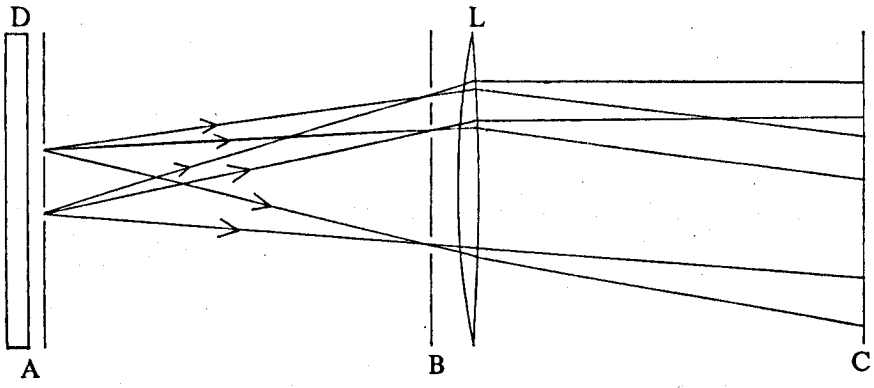



Figure 1. Constructing a correlogram. D is a diffuse light source, L a lens and C the plane of the correlogram of A with B

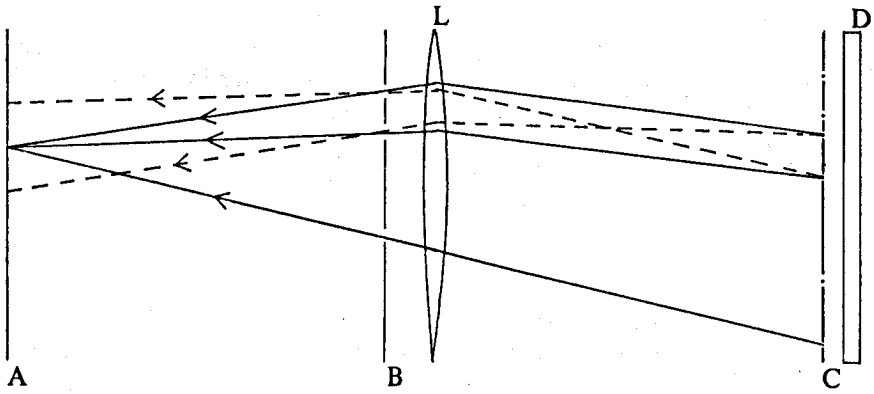



Figure 2. Reconstructing a pattern. Full lines are paths traversed in Figure 1. Broken lines are paths *not* traversed in Figure 1

pattern on A is a copy of the configuration to be looked for, and an exceptionally bright spot will appear on C at every point which lies opposite to an occurrence of A in B. Furthermore – and this is something that cannot be done holographically – if a scaled-down version of A appears somewhere in B, this fact can be detected by moving the viewing screen towards the lens L, when an exceptionally bright spot will appear on the screen at an appropriate distance from L. The correlograph can therefore be used as a pattern-recognition device possessing both displacement invariance and size invariance.

But the main interest of the device is its ability to store simultaneously the correlations between several different pairs of patterns, in such a way that either member of any pair can be used for reconstructing the other. Let $C_1$ be the correlogram between $A_1$ and $B_1$, let $C_2$ be the correlogram between $A_2$ and $B_2$, and so on. Then (up to a certain limit of saturation, which we shall discuss below) one can achieve this multiple performance by constructing a joint correlogram C which is the union of the various pinhole sets $C_1$, $C_2$, etc. For the optical correlograph the exact theory of this application is difficult to formulate precisely, because to obtain a high information storage density the pinholes must be small; but if they are too small diffraction effects seriously complicate the situation, which can no longer be described by geometrical optics. A further complication is presented by edge effects in an optical realization of the correlograph. We shall therefore not discuss this realization further, but proceed at once to a consideration of the digital correlograph, in which both these complications are avoided.

## 3. THE DIGITAL CORRELOGRAPH

The digital correlograph is an abstract system in which the cards A and B of the optical correlograph are regarded as discrete spaces each comprising $N$ points. An A-pattern or a B-pattern is a choice of $M$ points from one of these spaces. The product space of A and B is mapped on to a third space C, which also comprises $N$ points, in the following systematic manner: the point pair $(a_i, b_j)$ is mapped on to the point $c_k$ if and only if $j-i=k$ or $k-N$. (Each of the three subscripts runs from 1 to $N$.) There is a converse mapping of point-pairs from C and B on to points of A: $(c_k, b_j)$ is mapped on to $a_i$ if and only if $j-k=i$ or $i-N$. This converse mapping is employed in the reconstruction of an A-pattern from the correlogram and the paired B-pattern. Another converse mapping allows the retrieval of a B-pattern from its A-pattern: the point pair $(c_k, a_i)$ is mapped on to $b_j$ if and only if $i+k=j$ or $j-N$. (The + sign in the last equation may serve as a hint to the reader who has not yet solved the exercise offered in the preceding section.)

Now let us consider briefly the storage of several pairs of patterns. Each pair maps onto $M^2$ of the $N$ points of C, but these $M^2$ points may not all be distinct. After the association of $R$ pairs of patterns, $RM^2$ (not necessarily distinct) points on C will have been added to the correlogram. Let $pN$ be the number of distinct points of C which belong to the multiple correlogram. Then if the recorded patterns are random and uncorrelated we may safely assume that

$$1-p=\exp(-RM^2/N). \tag{1}$$

In the reconstruction of an A-pattern from the correlogram and the associated B-pattern we employ the converse mapping $(c_k, b_j)\rightarrow a_i$, for all points $c_k$ belonging to the correlogram and all points $b_j$ of the B-pattern. A particular point belonging to the original A-pattern will be activated $M$ times; but the

chance that a point not belonging to the A-pattern will be activated $M$ times is only $p^M$. There are $N$ points altogether in the space A; so we can be fairly sure of not obtaining any spurious points at a threshold of $M$ if

$$Np^M<1. \tag{2}$$

Regarding the latter equation as an equality defining the point of saturation, we may rewrite (1) and (2) as

$$RM=-(N/M)\log_e(1-p) \tag{3}$$

and

$$\log_e N=-M\log_e p \tag{4}$$

respectively. From these two equations it is an easy matter to determine the density at which information is stored in C when the digital correlograph is working near saturation. The information content of a single A-pattern is, in natural units rather than bits,

$$\log_e \binom{N}{M};$$

so when $R$ such patterns can be retrieved with accuracy the stored information amounts to

$$I_e=R\log_e\binom{N}{M}=RM\log_e N \text{ approximately}$$

(provided $M/N$ is small, as we shall soon verify). So by (3) and (4)

$$I_e=N\log_e p\log_e(1-p). \tag{5}$$

This expression is obviously maximal when $p=\frac{1}{2}$; so the maximum number of natural units which can be reliably retrieved is $N(\log_e 2)^2$ natural units. Noting that 1 bit$=\log_e 2$ natural units, and that $\log_e 2=0{\cdot}69$, we conclude that we can store information in the space C, regarded as a set of binary registers, to a density of 0·69 bits per register — nearly 70 per cent as densely as information is stored in a random access store!

To complete this discussion we note that when the system is being stretched to its limit, so that $p=\frac{1}{2}$, equation (4) implies that $M=\log_2 N$, so that $M/N$ is rather small, as assumed in approximating $I$; and that the number of pairs of patterns which have been associated is given by $R=N\log_e 2/(\log_2 N)^2$.

## 4. A COMPUTER SIMULATION

We have tested our theoretical results by computation, taking $N=256$ and $M=\log_2 N=8$. The theoretical value for $R$ at saturation is $4\times 0{\cdot}693=2{\cdot}77$, so we should be able to store two pairs of patterns with accurate retrieval, and three pairs with only slightly inaccurate retrieval. Such is indeed the case. With one pair of random 8-point patterns the correlogram is found to comprise 59 points, and retrieval is perfect. When a second pair is loaded the number of points of C involved rises to 101, and either member of each pair can still be retrieved without error from the other. When a third pair is loaded, the correlogram comprises 136 of the 256 points of C. Patterns $A_1$

and $B_1$ still retrieve one another without error, but inaccuracies arise in the mutual retrieval of $A_2$ and $B_2$, and of $A_3$ and $B_3$. In the reconstructions of $A_2$, $B_2$, $A_3$ and $B_3$ the numbers of spurious points appearing are 1, 1, 2 and 0 respectively; so we are beginning to witness breakdown, and this becomes catastrophic if any more pairs are loaded into the system.

With random 5-point patterns the system works less efficiently; it can only be loaded to a $p$ value of about 0·3 without the appearance of many spurious points in the retrieved pattern:

| Number of stored pairs | Number of points in correlogram | Divided by 256 | Mean number of spurious points |
|---|---|---|---|
| 1 | 25 | 0·1 | 0 |
| 2 | 49 | 0·2 | 0 |
| 3 | 69 | | 0 |
| 4 | 85 | 0·3 | 0·25 |
| 5 | 104 | 0·4 | 1·4 |
| 6 | 121 | | 3·3 |
| 7 | 131 | 0·5 | 7 |

## 5. THE ASSOCIATIVE NET

The associative net is logically similar to the digital correlograph, and performs much the same function, but it sacrifices the displacement invariance of the correlograph in return for a much greater absolute storage capacity. As shown in figure 3 the information is stored in the associative net in a set of on/off switches which are situated at the $N_A N_B$ intersections between a set of $N_A$ A-lines and a set of $N_B$ B-lines. A pair of patterns is stored associatively by sending pulses down $M_A$ chosen A-lines and at the same time sending pulses down $M_B$ chosen B-lines, and turning on switch $c_{ij}$ if the lines $a_i$ and $b_j$ both carry pulses. A second pair of patterns is stored by repeating this process and turning on further switches by the same rule; if such a switch is already on, it is simply left on (but see later). Retrieval of an A-pattern from a B-pattern is effected by sending $M_B$ pulses down the appropriate B-lines, and these are transmitted to certain A-lines through the switches which happen to be on. Each A-line is fitted with a threshold detector, and if the line receives as many as $M_B$ pulses through its switches it discharges a pulse. The A-lines which discharge pulses will include those of the required A-pattern; one will try to arrange that no other lines also discharge.

In the absence of damage, or inaccuracy in the cue pattern, the theory goes very much as for the digital correlograph. The corresponding relationships are:

$$RM_A = -(N_A N_B / M_B) \log_e (1-p),$$

$$\log_e N_A = -M_B \log_e p,$$

and
$$I = R \log_e \binom{N_A}{M_A} = RM_A \log_e N_A = N_A N_B \log_e p \,.\, \log_e (1-p).$$

Again we find that the maximum information density is about 0·69 bits per switch for a large net, and we conclude that in order to attain this maximum density we must have $M_B = \log_2 N_A$.

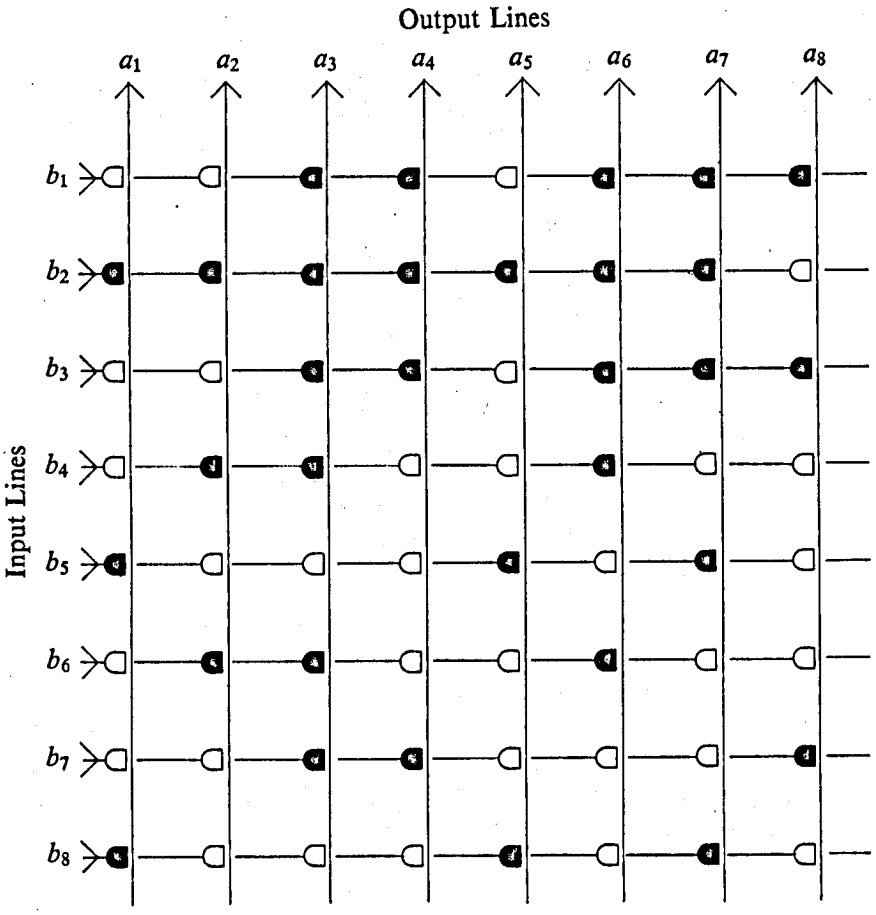



Figure 3. Associative net

The critical value of $R$, the number of stored patterns, is then

$$R = (N_A N_B / M_A M_B) \log_e 2.$$

If the system is being used both for retrieving As from Bs and vice versa, optimal efficiency requires also that $M_A = \log_2 N_B$ and $R$ takes the alternative form $R = N_A N_B \log_e 2 / (\log_2 N_A \,.\, \log_2 N_B)$.

For example, with $N_A = 10^6 = N_B$ (figures which might not be unreasonable for a piece of cerebral cortex) we should be able to associate approximately $1{\cdot}5 \times 10^9$ pairs of 20-impulse patterns before saturating the net.

## 6. THE EFFECTS OF DAMAGE AND INACCURACY

Since the correlograph and the associative net store information in a distributed manner, one might expect them to be able to perform well in the face of damage or inaccurately presented cues. We will therefore imagine that initially the associative net was loaded so that a fraction $p$ of the switches were turned on, but that some disaster then occurred which caused a fraction $1-q$ of these switches to be turned off, at random, so that only a fraction $pq$ remains on.

Consider now what happens when a stored B-pattern is fed into the B-lines. An A-line which ought to put out a pulse will only receive $qM_B$ pulses on the average, rather than $M_B$, so if this line is to fire the threshold must be lowered to $tM_B$, say, where $t$ is sufficiently smaller than $q$. But $t$ must not be too small, or else the A-lines which ought to be silent will emit pulses; these A-lines will receive $pqM_B$ pulses on the average, so $t$ must be safely larger than $pq$. Straightforward statistical analysis shows, in fact, that the evoked A-pattern will be accurate only if the following two inequalities hold:

$$(1/M_B)\log_e M_A < t\log_e(t/q) + (1-t)\log_e((1-t)/(1-q)) \quad (6)$$

$$(1/M_B)\log_e N_A < t\log_e(t/pq) + (1-t)\log_e((1-t)/(1-pq)) \quad (7)$$

The former inequality must be satisfied if all the A-lines which ought to fire are to do so; the latter if all the others are to remain silent.

We have made some calculations on the performance of the damaged associative net for which $N_A = N_B = N = 10^6$, and $M_A = M_B = M$. We began by choosing a value for $q$ in the range 0·4 to 1·0, and used equation (6), regarded as an equality, to find a functional relation between $M$ and $t$. Next we used (7), also regarded as an equality, to find a functional relation between $p$ and $M$, which is equivalent to a relation between $p$ and $t$. For given $t$, $p$ and $q$ the fractional capacity of the net, in bits per switch, can be calculated as follows:

$$I_2 = R\log_2\binom{N}{M} = RM\log_2(N/M) \quad (8)$$

gives the amount of stored information, in bits, to a rather better approximation than that used in sections 3 and 5. But

$$R = -(N^2/M^2)\log_e(1-p). \quad (9)$$

Therefore the mean number of bits per switch is, by (6), (7), (8) and (9),

$$I_2/N^2 = \log_2(1-p)(t\log_e p + (1-t)\log_e(1-pq)/(1-q)). \quad (10)$$

For given $q$ and $t$ this expression has a maximum when $p$ is between 0 and 0·5, and this maximum is easily calculated. So there are definite values of

$t$, $M$ and $p$ which optimize the performance of the system for given $q$, and these are given in the following table:

| | | | | | | |
|---|---|---|---|---|---|---|
| $q=$ | 0·4 | 0·5 | 0·75 | 0·9 | 0·95 | 1·00 |
| $t=$ | 0·261 | 0·340 | 0·557 | 0·731 | 0·811 | 1 |
| $p=$ | 0·224 | 0·24 | 0·257 | 0·30 | 0·337 | 0·5 |
| $M=$ | 111 | 84 | 43 | 31 | 27 | 20 |
| $R=$ | $2{\cdot}1\times10^{7}$ | $3{\cdot}9\times10^{7}$ | $1{\cdot}6\times10^{8}$ | $3{\cdot}7\times10^{8}$ | $5{\cdot}6\times10^{8}$ | $1{\cdot}7\times10^{9}$ |
| $I/N^2=$ | 0·03 | 0·044 | 0·10 | 0·18 | 0·23 | 0·54 |

The fact that the last figure, 0·54, is rather less than $\log_e 2$, is due to the finiteness of the net. The corresponding figures for the damaged correlograph would be exactly the same except for the $R$ values, which would all be smaller by a factor of $10^6$.

The main conclusion to be drawn from these figures is that although damage reduces seriously the stored information density, the associative net and the correlograph can still be made to work reliably under such adverse conditions by increasing the number of pulses in the input patterns and being careful not to load too many pairs of patterns into the system. A further point which should be noted is that precisely the same calculations apply to a situation in which there is no damage, but the input cues are incomplete. The absence of a pulse in an input line which ought to carry one is logically equivalent to a malfunctioning of a switch which ought to be on but is in fact off. So the above table applies to the case in which $q$ represents not damage to the switches but the degree of completeness of the input pattern – the number of input pulses being not $M$ but only $qM$. So again, if the value of $M$ is made large enough, and not too many pairs are loaded into the system, we can obtain accurate retrieval of A-patterns from incomplete B-patterns.

## 7. LEARNING WITH FORGETTING

So far, in considering the digital correlograph and the associative net, we have assumed that every switch which is turned on remains on unless it is turned off by accidental damage. We have found that in the absence of damage there is a fairly sharp limit to the number of associations that can be stored; if this number is exceeded the performance of the system degenerates rapidly. But is it not possible to store an indefinite number of associations, if one is prepared to forget old ones gradually as new ones are recorded? We now consider this problem.

Let us assume, then, that an associative net has been loaded with a number of pairs of patterns, and that $pN^2$ of its $N^2$ switches are now on. A new pair of patterns is to be stored, but the value of $p$ is not to increase. The new pair of patterns will call for the turning on of $M^2$ switches, but $pM^2$ of these will already be on. If each of the others is turned on with probability $s$, then the

total number of switches turned on in recording the new pair will be $s(1-p)M^2$. The total number of switches which are already on is $pN^2$, so to maintain $p$ at its previous value we must turn off each of these with probability $r$ given by

$$rpN^2=s(1-p)M^2.$$

After a new association has been recorded, then, each of the relevant switches will be on with a probability $p+s(1-p)$, but this probability will fall steadily towards $p$ as other associations are recorded. Suppose that at any stage it has a probability $p+z$ of still being on. Then after one more recording this chance becomes $p+z'$, where

$$p+z'=(1-r)(p+z)+(sM^2/N^2)(1-p-z).$$

Using the fact that $r=s(1-p)M^2/pN^2$ we deduce straightforwardly that

$$z'=z(1-sM^2/pN^2).$$

Remembering that $M^2/N^2$ is small in general, we rewrite this as

$$z'=z\exp(-sM^2/pN^2),$$

and deduce that after the recording of $n$ other associations the value of $z$ will have dropped by a factor $\exp(-nsM^2/pN^2)$. An alternative form for this factor may be derived in terms of $R$, the effective number of stored patterns, namely

$$R=-(N^2/M^2)\log_e(1-p)=pN^2/M^2 \text{ approximately.}$$

So if the original association is overlaid with $n$ other associations, all of the same strength $s$, the final value of $z$ will be given by

$$z^{(n)}=z^{(o)}\exp(-ns/R).$$

By an obvious generalization, if the subsequent associations have strengths $s_1, s_2, \ldots, s_n$, then the exponential factor on the left becomes

$$\exp-(s_1+s_2+\ldots+s_n)/R.$$

In particular, if each recording is of full strength, the memory length is just $R$, which is the effective number of associations in store.

In order to make effective use of an associative net which forgets earlier messages slowly as it learns new ones, it will of course be essential for the threshold on each output line to be less than the $M$ value of any input signal which is to survive being overlaid by a substantial number of other signals. In fact, the effect on a recorded signal of overlaying it with others is much the same as the effect of damage, discussed in section 6. There are two ways of increasing the survival time of a particular association: either recording it several times in succession – 'overlearning' it – which has the effect of raising its effective strength to near unity; or increasing the number $M$ of pulses in the input pattern, so that it takes a longer time for the number of effective pulses to fall below the preset threshold. The mathematical analysis of this situation would, however, take us beyond the scope of this paper.

## REFERENCE

Willshaw, D.J., Buneman, O.P. and Longuet-Higgins, H.C. (1969) *Nature*, **222**, 960.

# 20

# Hierarchical Decomposition of Complexity

M. H. van Emden
Mathematical Centre
Amsterdam

**Summary**

Classification methods for quantitative data have received more attention than those for qualitative data. Excess-entropy, which may be interpreted as a measure of complexity, enables us to formulate existing methods for normally distributed data in such a way as to be applicable also to qualitative data.

After an introductory section 1, section 2 defines excess-entropy and provides some information-theoretical background. It then treats the qualitative case by methods analogous to principal components and clustering respectively. The first of these is much like the existing technique of Association Analysis.

Section 3 is concerned with the multivariable normal distribution. The well-known method of principal components is given a simple interpretation in terms of entropy. Furthermore, excess-entropy is shown to be identical with the expected value of the log likelihood-ratio between the hypotheses that sets of variables are dependent and independent respectively.

In section 4 it is shown that in Markov chains excess-entropy provides a measure of clustering. In order to be able to do so, an operation on a Markov chain has to be defined: that of 'fusing' two states.

## 1. THE ANALYSIS OF COMPLEX SYSTEMS

We may think of a *system* as a set of variables influencing each other. *Complexity* may arise in two ways: the presence of a large number of variables and the fact that most of these influence many others.

There exist situations where the simultaneous treatment of all variables presents a computational problem that is too large by any standard. Yet in such a situation it is sometimes possible to decompose the whole system into a few subsystems with relatively weak interactions between them. At this

level we have a system of manageable complexity where the subsystems are treated as 'black boxes'.

In their turn, each of these subsystems may be subjected to the same treatment, and so on. This process is just a particular case of the well-known principle: divide and rule; or, as we shall encounter it as a recurrent theme: *hierarchical decomposition of complexity.*

In this study we want to see what can be done by viewing the interaction between subsystems as *information transfer.* The decomposition of complexity then corresponds to the decomposition of the total amount of information transfer.

### 1.1. Example: A set of linear algebraic equations

Let $Ax=b$ (1)

represent a set of $n$ linear algebraic equations in $n$ unknowns. $A$ is an $n \times n$-matrix and $x^T=(\xi_1, \ldots, \xi_n)$, $b^T=(\beta_1, \ldots, \beta_n)$ are $n$-component vectors.

Consider a partition $\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ of $A$, where $A_{11}$ is a $k \times k$-matrix. With $(x_1^T, x_2^T)$, $(b_1^T, b_2^T)$ as the corresponding partitions in $x^T$ and $b^T$, we can write (1) as:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \tag{2}$$

Suppose that variables $\xi_1, \ldots, \xi_k$ are but 'weakly' represented in the last $n-k$ equations, i.e., the elements of $A_{21}$ are small compared to those of $A_{11}$ or $A_{22}$. In such a situation it may be advantageous to use the following iterative scheme for solving (1):

Start with:

$$x_2^0 := b_2/A_{22} \tag{3}$$

(In this notation we denote the solution vector of: $A_{22}x_2^0=b_2$);
For $k=0, 1, 2, \ldots$ do:

$$x_1^{k+1} := (b_1 - A_{12}x_2^k)/A_{11}; \tag{4}$$

$$x_2^{k+1} := (b_2 - A_{21}x_1^{k+1})/A_{22}; \tag{5}$$

The sequence of: $\begin{pmatrix} x_1^1 \\ x_2^1 \end{pmatrix}, \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix}, \ldots$ is regarded as a sequence of approximations to the solution of (1).

If $A_{21}$ consists of zeros only, the solution is obtained after (3) and a single execution of (4). It seems reasonable to suppose that this scheme converges faster when the elements of $A_{21}$ are smaller. In general, the solution is not obtained after a finite number of steps because a change in $x_2$ is transmitted to $x_1$ via $A_{12}$ in step (4), and then the change in $x_1$ is transmitted to $x_2$ via $A_{21}$ in (5), and so on. Viewed in this way, $A_{12}$ and $A_{21}$ represent the interactions between the subsystems $A_{11}$ and $A_{22}$.

It is desirable to find a quantitative description of this interaction. In a similar situation (see next example) an information transfer may be defined between subsystems.

In sections 3.4 and 4.2 we exhibit special systems of linear equations where we can express the interaction between subsystems as a quantity of information in the usual interpretation of this concept (see section 2.2).

### 1.2. Example: A model of the design process

Let us consider the following abstraction of a complicated design problem: a designer has to construct a *form* which has to satisfy a large number of conditions.

For example, we might think of the design of a human settlement where the number of conditions may run in the hundreds, and many of which are conflicting. Here again complexity may arise in two ways: the number of variables is large and there occur many interactions between them. In general there may not exist a form which satisfies all conditions to the required extent, and so the designer should aim at maximizing goodness-of-fit with respect to all conditions simultaneously. The designer cannot keep in mind all of the conditions at once; suppose he finds an iterative design process by first concentrating on some subset $A_{11}$ of the conditions, finding a provisional form that maximizes goodness-of-fit locally and then proceeding with another subset $A_{22}$. Interaction between condition $i$ and condition $j$ arises in the following way: In adapting the form to condition $i$, it may be modified in such a way that goodness-of-fit with respect to condition $j$ decreases.

We see that the iterative design process sketched above is analogous to the iterative method of solving a system of equations. Suppose that conditions are partitioned into subsets $A_{11}$ and $A_{22}$, then the designer first ignores $A_{22}$ and then $A_{11}$. When there is no interaction between the two, he is done. In general he finds on returning to $A_{11}$, that, while concentrating on $A_{22}$, he has undone some of the good properties his provisional form had with respect to $A_{11}$ and he will start a following cycle of the iterative process. Even when there is some interaction between $A_{11}$ and $A_{22}$, this process may succeed in yielding a satisfactory form after an acceptable number of cycles.

Alexander (1967) has studied the problem of finding subsets in the set of all conditions in such a way that the amount of interaction between subsets is small compared to interaction within subsets. He quantified 'interaction' by regarding it as information transfer. To this end he constructed a model consisting of a set of random variables corresponding to conditions. He was then able to define information transfer as a difference between entropies. He reports the existence of computer programs for the hierarchical decomposition of the set of conditions, where their interactions are specified pair by pair.

## 2. ENTROPY AND OBJECT-PREDICATE TABLES

### 2.1. The object-predicate table

Suppose we have a certain set of *objects* and each of these may be described by stating whether it does or does not have any of a fixed (same for all objects) set of *predicates*. In this way each object is identified with a certain subset of the predicates; when two objects have an identical subset there is, in this context, no way to tell them apart.

This situation may be represented by an *object–predicate table*: a rectangular array of noughts and crosses. The $j$th cell of the $i$th row of this array shows whether the $i$th object does (when it contains a cross) or does not (when it contains a nought) possess the $j$th predicate.

| objects ↓ $i$ \ $j$ → predicates | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | × | 0 | 0 | 0 |
| 2 | 0 | × | 0 | 0 | × | × |
| 3 | 0 | × | × | × | × | 0 |
| 4 | × | 0 | 0 | 0 | × | 0 |

Figure 1. An object–predicate table

The object–predicate table is a rather general scheme for exhibiting relations between objects, either via (common) predicates or, directly, by identifying the $i$th predicate with the $i$th object. Nought or cross then indicates whether the one object is dependent on the other. An example of a system would then be a set of objects related as specified by their object–predicate table.

### 2.2. The entropy functional

In order to provide a conceptual framework and a terminology for what follows, we will first review some important properties of the entropy functional $H$ (Khinchin 1957).

Suppose there are two sets of descriptions of events

$$A=\{a_1, \ldots, a_m\} \text{ and } B=\{b_1, \ldots, b_n\}.$$

A probability is assigned to each description:

$$\Pr\{a_k\}=p_k\geqslant 0, \quad \sum_k p_k=1, \quad k=1, \ldots, m \text{ and}$$

$$\Pr\{b_l\}=q_l\geqslant 0, \quad \sum_l q_l=1, \quad l=1, \ldots, n.$$

Thus $a_k$ and $b_l$ are sets of events having an identical description, i.e., in this context events belonging to the same set cannot be distinguished. In the sequel we will therefore denote such a set of events as 'event'. The entropy functional $H$ associated with $\{p_1, \ldots, p_m\}$ is defined as:

$$H(A)=-\sum_k p_k \log p_k. \tag{1}$$

$H$ may be interpreted as a measure of uncertainty with respect to the outcome of an experiment $A$, which is the event $a_k$ with probability $p_k$. The following two properties of $H$ justify such an interpretation: $H$ is never negative and vanishes if $p_k=1$ for some $k=1, \ldots, m$. In this case $a_k$ is certain to occur; the uncertainty vanishes. The other property is that $H$ attains its maximum when all events are equally probable, which corresponds to the situation of maximum uncertainty. This property follows from the well-known inequality:

$$f(\sum_k \lambda_k x_k) \geqslant \sum_k \lambda_k f(x_k) \tag{2}$$

where $\lambda_k \geqslant 0$, $\sum_k \lambda_k = 1$ and $f$ is a continuous and convex function of $x$. We now choose $\lambda_k = 1/m$, $f(x) = x \log x$ and $x = p$.

$$f\left(\sum_k \lambda_k x_k\right) = \frac{1}{m} \log \frac{1}{m} \leqslant \sum_k \frac{1}{m} p_k \log p_k \Rightarrow$$

$$H(A) = -\sum_k p_k \log p_k \leqslant \log m.$$

Let us consider also the Cartesian product set $A \times B$ of the two sets. On $A \times B$ a two-dimensional array of probabilities is defined as: $\Pr\{a_k \text{ and } b_l\} = r_{kl}$. The associated conditional probabilities are: $\Pr\{b_l | a_k\} = q_{kl} = r_{kl}/p_k$. (the probability that $b_l$ will occur under condition that $a_k$ has occurred).

The two sets are said to be independent when $r_{kl} = p_k q_l$. In that case $q_{kl} = q_l$, which means that the probability of the occurrence of $b_l$ is independent of which $a_k$, $k=1, \ldots, m$, has occurred. For the entropy of the product scheme we have:

$$H(A \times B) = -\sum_{kl} r_{kl} \log r_{kl}.$$

In the case of independence this reduces to:

$$H(A \times B) = -\sum_{kl} p_k q_l (\log p_k + \log q_l).$$

$$= -\sum_l q_l \sum_k p_k \log p_k - \sum_k p_k \sum_l q_l \log q_l$$

$$H(A \times B) = H(A) + H(B). \tag{3}$$

In case $A$ and $B$ are dependent, this relation generalizes to:

$$H(A \times B) = -\sum_{kl} r_{kl} \log r_{kl} = -\sum_{kl} p_k q_{kl} \log p_k q_{kl}$$

$$= -\sum_{kl} p_k q_{kl} \log p_k - \sum_{kl} p_k q_{kl} \log q_{kl}$$

$$= H(A) + \sum_k p_k H_k(B)$$

$H_k(B)$ is regarded as the outcome of a random variable: The entropy of the conditional scheme $\{q_{k1}, \ldots, q_{kn}\}$ under condition that $a_k$ has occurred. The second term is then the mathematical expectation of $H(B)$ in the scheme $A$, which we shall designate by $H_A(B)$:

$$H(A \times B) = H(A) + H_A(B) \text{ and similarly}$$

$$H(A \times B) = H_B(A) + H(B). \tag{4}$$

$H_A(B)$ never exceeds $H(B)$. This is a consequence of the inequality (2) where this time we take $\lambda=p$ and $f(x)=x\log x$.

$$-\sum_k p_k q_{kl}\log q_{kl} \leqslant -\left(\sum_k p_k q_{kl}\right)\log\left(\sum_k p_k q_{kl}\right).$$

Summing both sides over $l$ gives:

$$H_A(B)\leqslant H(B). \tag{5}$$

From (3) and (4) we find that equality is attained in the case of independence.

If we view the entropy functional as a measure of uncertainty this may be interpreted as the fact that prior knowledge of the outcome of $A$ never increases the uncertainty in the outcome of $B$.

The inequality (5) is an important one: In a study on the interactions of nucleons, S. Watanabe introduced in 1939 a measure of dependence between random variables based on a difference between entropies. In a later paper (Watanabe 1960) this idea is elaborated.

From (4) we find that:

$$\begin{aligned} H(A)+H(B)-H(A\times B) &= H(A)-H_B(A)\\ &= H(B)-H_A(B)\\ &= C(A,B) \text{ by definition.} \end{aligned} \tag{6}$$

The quantity $C$ defined in this way is never negative according to (5) and it vanishes only when $A$ and $B$ are independent. Watanabe (1960) proposed to use $C$ as a measure of dependence between $A$ and $B$. In this report the quantity $C$ will be called the *excess-entropy*.

### 2.3. Entropy in object–predicate tables

2.3.1 *Entropy and excess-entropy in partitions*

A $k$-partition of a set $S$ is a set of $k$ mutually disjoint subsets (called 'cells') whose union is $S$. Suppose the $i$th cell has $n_i$ elements and $\sum_i n_i=n$. We may associate with a $k$-partition the set $\{n_1/n,\ldots,n_k/n\}$ of non-negative numbers whose sum is 1.

This analogy to the discrete probability scheme caused Rescigno and Maccacaro (1961) to define the entropy of a partition as:

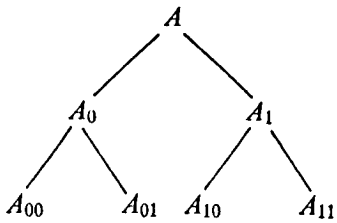
$$H=-\sum_i \frac{n_i}{n}\log\frac{n_i}{n}=\log n-\frac{1}{n}\sum_i n_i\log n_i. \tag{1}$$

To every pair of partitions there corresponds a product partition (which is again a partition): if a partition is defined on $S$, so also it is on every subset of $S$ and therefore also on each of the cells of the other partition. Accordingly, there corresponds an excess-entropy to every pair of partitions $A$ and $B$:

$$C(A,B)=H(A)+H(B)-H(A\times B).$$

Let us consider partitions of the set $A$ generated by subjecting every cell to a 2-partition. One of the subcells is denoted by putting a 0, the other by

putting a 1 behind the name of the cell. Starting from the trivial partition $\{A\}$ of $A$ we get successively:




(2)

Now let the elements of $A$ be partitions. We are going to study the entropy-relations between the product partitions of the partitions of a subset of $A$: $H$ and $C$ will denote entropy and excess-entropy again, with indices to indicate to which subset of $A$ they apply. We find for the excess-entropy between the two product partitions $\Pi A_0$ and $\Pi A_1$:

$$C(0, 1) = H_0 + H_1 - H.$$

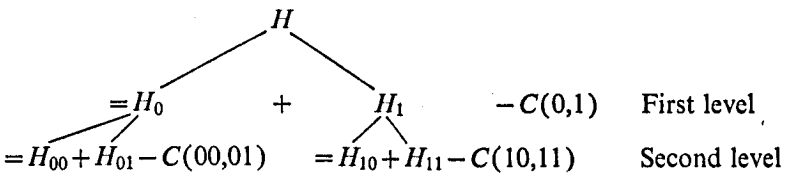
It will be found useful to extend this definition to apply to more than 2 partitions, for instance the 4-partition of the lowest level of (2):

$$C(00, 01, 10, 11) \overset{\text{def}}{=} H_{00} + H_{01} + H_{10} + H_{11} - H.$$

This 4-way excess-entropy can be expressed in 2-way entropies as follows:

$$\begin{aligned} C(00, 01, 10, 11) &= H_{00} + H_{01} - H_0 + H_{10} + H_{11} - H_1 + H_0 + H_1 - H \\ &= C(00, 01) + C(10, 11) + C(0, 1). \end{aligned}$$

This can be represented in a hierarchical diagram:




Thus the multi-way excess-entropy of a certain level may be hierarchically decomposed into 2-way excess-entropies of all levels not below it. We regard this as a method in compliance with the principle of dealing with systems by hierarchical decomposition of complexity. The analogous procedure for a set of random variables has been described by Watanabe (1960).

2.3.2 *Data compression in an object-predicate table*

Let us now study the object–predicate table as directly as possible from the point of view of the information provided by the predicates about the objects. This may be illustrated by a guessing game: One person takes an object in mind and has to answer yes or no to another person's questions about it in the form: Does it have predicate $p_i$? The answers to questions concerning a subset of the predicates define a partition in the set of objects. Following the classical definition of Shannon, a suitable definition for the information

provided by a set of predicates is the entropy of their product partition as defined in the previous section. The set of $n$ predicates defines a partition of $2^n$ cells and the maximum entropy of such a partition is $n$ bits. When the actual entropy is less than this, we say there is *redundancy* in the set of predicates.

When we realize that there exists an object–predicate table with $n$ predicates and $2^n$ objects where every cell of the partition contains exactly one object and which therefore does not contain any redundancy, it is apparent that in tables with moderately large and roughly equal numbers (between, say, 10 and 1,000) of objects and predicates, enormous amounts of redundancy are usual.

Thus we are led to the problems of *data compression* (see Tou 1967, Watanabe 1965):

1. For given $k<n$ find a subset $\{p_{i_1}, \ldots, p_{i_k}\}$ of the predicates such that $H(p_1, \ldots, p_n) - H(p_{i_1}, \ldots, p_{i_k})$ is a minimum.

2. For $k=1, 2, \ldots, n$ find the $k$ such that the data compression achieved in 1. is, in some respect, optimum.

It will be interesting to encounter, in a later section, an analogous problem for an $n$-dimensional normal probability distribution.

2.3.3 *Hierarchical decomposition of excess-entropy*

*Association analysis.* In plant ecological studies data may be obtained in the following way. In the geographical area to be treated, a number of plots, called *quadrats*, are staked off and of each of these it is noted which species of plants are present. Williams and Lambert (1959) (the quotations below are from this paper) have proposed *Association Analysis* as a method for sorting quadrats into groups.

Data of this origin may be presented as an object–predicate table where it is immaterial whether species (quadrats) are identified with the objects (predicates). 'The basic problem is to subdivide the population so that all associations disappear . . .'. Here 'association' is to be used in its 'statistical sense'. It seems desirable to give a more precise interpretation of 'association'.

In 2.3.1 we have defined the excess-entropy of a set of partitions. A predicate effects a 2-partition in the set of objects (the objects that do and those that do not have the predicate); a set of predicates therefore corresponds to a set of partitions in the objects.

Likewise, an object effects a 2-partition in the set of predicates (those it does and those it does not have) and, by the previous sentence, this object corresponds to two sets of partitions in the set of objects. To these two sets of partitions there corresponds an excess-entropy and this we may call the *entropy loading* of that object.

Now the set of all predicates together define a product partition in the set of objects and this has a *collective* entropy. Every predicate on its own

defines a 2-partition and the *individual* entropy of this partition. The difference between the sum of individual entropies and their collective entropy is the (multi-way) excess-entropy defined in 2.3.1. Its hierarchical decomposition may be used to analyse the structure of the interrelations existing in the set.

Let us identify objects as species and predicates as quadrats. The purpose of the rest of this section is to show that the excess-entropy of a set of predicates has the properties that Williams and Lambert (1959) expect the undefined concept of association to have.

(a) Williams and Lambert (1959) argue that 'positive' as well as 'negative' associations are to be taken into account. From this we may infer that, roughly speaking, if two species are positively (negatively) associated, then the presence of the one makes occurrence of the other more (less) likely. Therefore association without sign is something that is expected to give a positive contribution in both cases. This is just what excess-entropy does: it is never negative and, independently of the sign of the interaction, indicates whether species influence each other more or less strongly.

(b) Apparently, association should not only be defined between a pair of species, but somehow all associations present in a set of species should be pooled. This is the reason why we have used the extension from a pair to an arbitrarily large set of partitions.

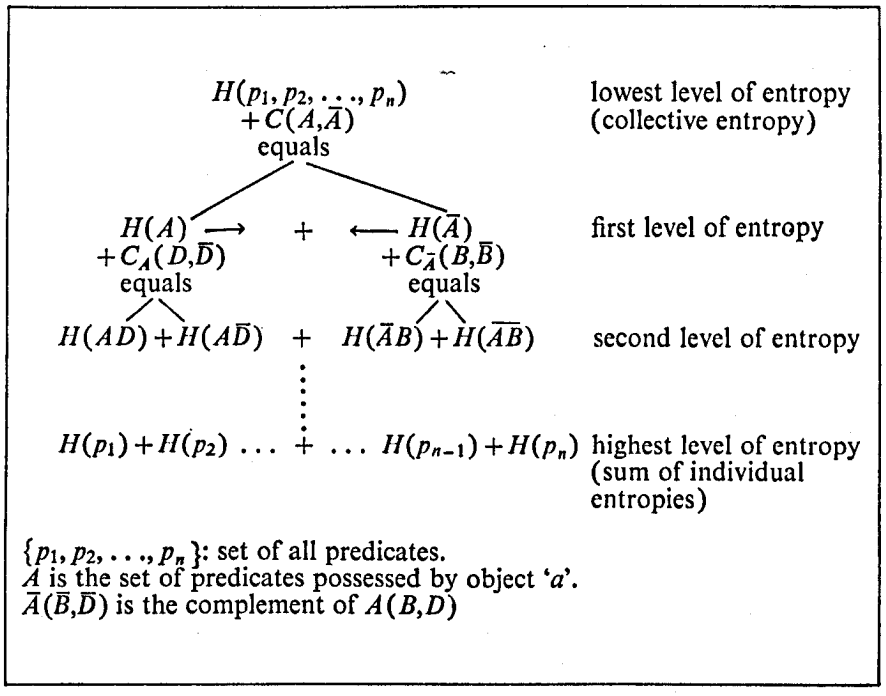



Figure 2. Diagram showing hierarchical decomposition of excess-entropy

(c) The objective of association analysis is to subdivide the set of quadrats so that all associations disappear. This is done by taking a particular species and partitioning the set of quadrats into those in which it did and those in which it did not occur.

The species is chosen so that the pooled association in the subsets is as small as possible. Stating it in terms of excess-entropy in the object predicate table, we can say that we must find the object with the highest entropy loading and repeat the process on each of the cells of predicates.

If we therefore interpret association as excess-entropy, we find that association analysis is the hierarchical decomposition of the total excess-entropy in such a way that the largest component of excess-entropy is produced first.

When studying a 'system', the basic phenomenon that one is interested in, is the way in which a whole (the system) is different from the set of its components. The phenomenon may be referred to as 'dependence', or as 'interaction', or as 'synergy'. Excess-entropy is a quantity closely related to this phenomenon: it is the difference between the sum of the entropies taken separately and the entropy of the predicates together. Complexity has something to do with it: excess-entropy would be zero in the case where there is no interaction at all between predicates and the system of all predicates is trivially simple. When excess-entropy is greater than zero, there is interaction between the components, and in examples 1.1 and 1.2 we would regard it as evidence of complexity. Thus it may be fruitful to see what can be done by regarding total excess-entropy as a measure of complexity. This principle has been stated before by Mowshowitz (1967) who applied it to the study of the complexity of graphs.

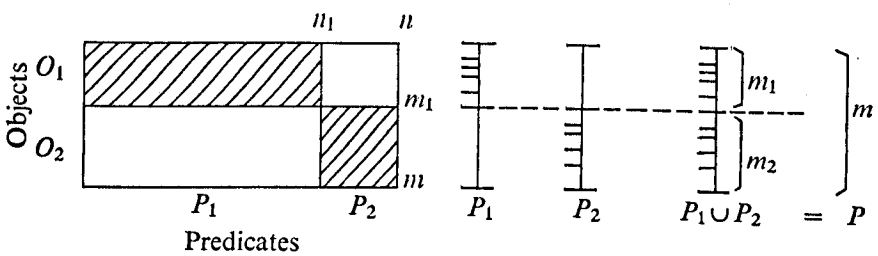
In decomposing total excess-entropy, various strategies may be followed. If we split on the object with lowest entropy loading, we may detect clustering (see 2.3.4, 3.4 and 4.2), while, if we do the opposite, it is a naïve step in data compression.

### 2.3.4 *Clustering*

In the first section we mentioned the possibility of a system of many variables consisting of a few subsets of variables with interactions between subsets weak relative to those within subsets.

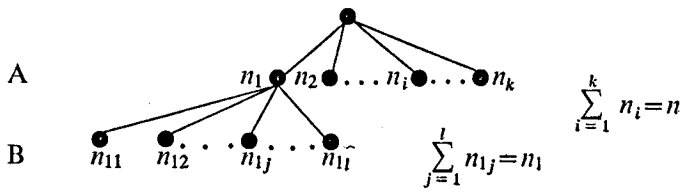
In such a case it is possible to study one aspect of the whole system by regarding a simple system consisting of these subsets as 'black boxes'.

It is necessary of course to describe in a more specific fashion the interactions between variables. To a certain extent this is possible in the object–predicate table. We will define the situation in which the table is considered to be 'completely decomposed' into two subsets of objects and predicates and we will show that in that case the excess-entropy between the subsets is minimal. This gives a more flexible way of describing a table, which is of practical importance because a table 'almost' decomposed is more likely to occur in practice than one completely decomposed.




Object-predicate table (without actual entries) with examples (at right) of partitions in the set of objects induced by subsets $P_1$ and $P_2$ of the set $P$ of all predicates.

If there are no crosses outside the shaded area, that is, when none of the predicates of $P_1$ is possessed by any of the objects in $O_2$ and vice versa, we say that the table is completely decomposed. The partition $P$ is the product partition of $P_1$ and $P_2$. This product is of a peculiar kind: $P_1$ subdivides only one cell of $P_2$ and vice versa. Let us consider a related special form of product: that of hierarchical subdivision. Suppose that we have a partition $A$ and that partition $B$ acts only on one cell of $A$; without loss of generality we may suppose this one to be the first.




$$H_A = \log n - \frac{1}{n}\sum_{i=1}^{k} n_i \log n_i;\quad H_B = \log n_1 - \frac{1}{n_1}\sum_{j=1}^{l} n_{1j} \log n_{1j};$$

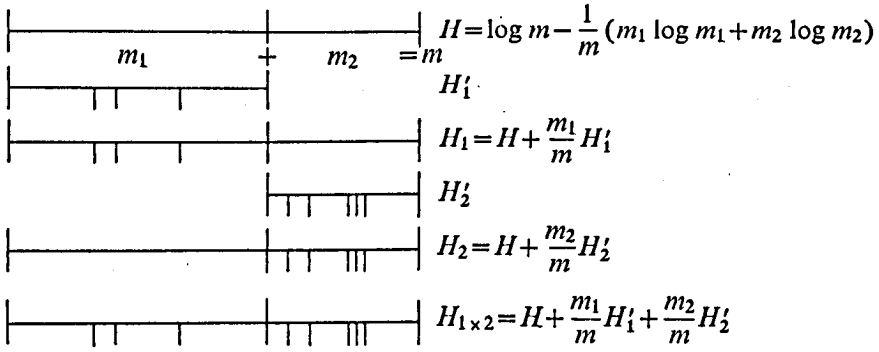
$$H_{A\times B} = \log n - \frac{1}{n}\left\{\sum_{i=2}^{k} n_i \log n_i + \sum_{j=1}^{l} n_{1j} \log n_{1j}\right\}$$

$$= \log n - \frac{1}{n}\left\{\sum_{i=1}^{k} n_i \log n_i - \left(n_1 \log n_1 - \sum_{j=1}^{l} n_{1j} \log n_{1j}\right)\right\}$$

$$H_{A\times B} = H_A + \frac{n_1}{n} H_B.$$

We use this formula to find the excess-entropy that exists between sets of partitions $P_1$ and $P_2$ completely decomposing the table. It seems convenient to derive the entropies of the partitions $P_1$ and $P_2$ by hierarchical subdivision of the partition $\{m_1, m_2\}$ that they have in common.

Any additional subdivision in the left half of $H_1$ or in the right half of $H_2$ leaves the excess-entropy $C(P_1, P_2)$ unchanged at $H$. Any additional subdivision in the right half of $H_1$ or in the left half of $H_2$ makes the excess-entropy $C(P_1, P_2)$ greater than $H$. Therefore the excess-entropy of an




For the excess-entropy between sets of predicates $P_1$ and $P_2$ we find:
$C(P_1, P_2)=H_1+H_2-H_{1\times 2}=H$

object–predicate table completely decomposed with respect to two mutually disjoint subsets of predicates $P_1$ and $P_2$ of $m_1$ and $m_2$ elements respectively is minimal and equal to $H=\log m-\frac{1}{m}(m_1 \log m_1+m_2 \log m_2)$.

## 3. ENTROPY AND THE NORMAL PROBABILITY DISTRIBUTION

### 3.1. Variance and entropy

Let $A$ be a positive definite symmetric matrix with proper values $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_n$ and corresponding proper vectors $x_1, x_2, \ldots, x_n$.

*Theorem* (Bellman 1960, p. 117)

If $A$ is positive definite,

$$\frac{(2\pi)^{k/2}}{\sqrt{|A|_k}}=\max_{R_n}\int_{R_k} e^{-\frac{1}{2}(z,Az)}\,dV_k, \tag{1}$$

where $|A|_k=\prod_{i=n-k+1}^{n}\lambda_i$, the product of the $k$ smallest proper values and $dV_k$ is the $k$-dimensional element of volume in $R_k$. The maximum is achieved for the $R_k$ spanned by the proper vectors corresponding to $k$ smallest proper values.

Taking $k=n$ and noting that $|A|_n=|A|$, the determinant of $A$, we obtain the well-known equality:

$$\frac{(2\pi)^{\frac{1}{2}n}}{\sqrt{|A|}}=\int_{R_n} e^{-\frac{1}{2}(z,Az)}\,dV_n, \tag{2}$$

which allows us to define as the $n$-dimensional normal probability density:

$$f(x)=\frac{|A|^{\frac{1}{2}}}{(2\pi)^{\frac{1}{2}n}}e^{-\frac{1}{2}(x,Ax)} \tag{3}$$

with $x$ some $n$-dimensional vector and $A$ a positive definite symmetric matrix. $V=A^{-1}$ is the covariance matrix of the distribution.

The determinant of $V$ is called the *generalized variance*; hereafter we will refer to it as the *variance*. The motivation of this treatment of the normal distribution is the fact that here, too, we may define the entropy functional. The practical use of the normal distribution is limited by the fact that in many situations the assumption that the data arise from a normal distribution is difficult to justify. The object–predicate table is of wider applicability. In both cases the entropy functional may be defined and this allows us to formulate analogous problems.

For the entropy of the normal distribution we find:

$$H=-\int_{R_n} f(x)\ln f(x)\,dV_n$$

$$H=-\int_{R_n}\frac{|A|^{\frac{1}{2}}}{(2\pi)^{n/2}}e^{-\frac{1}{2}(x,Ax)}\left\{-\frac{n}{2}\ln(2\pi)+\tfrac{1}{2}\ln|A|-\tfrac{1}{2}(x,Ax)\right\}dV_n$$

$$=-\tfrac{1}{2}\ln|A|+\frac{n}{2}\ln(2\pi)+\frac{|A|^{\frac{1}{2}}}{(2\pi)^{\frac{1}{2}n}}\int_{R_n}\tfrac{1}{2}(x,Ax)\,e^{-\frac{1}{2}(x,Ax)}\,dV_n$$

$$=\frac{n}{2}\ln(2\pi e)+\tfrac{1}{2}\ln|V|.$$

If we express entropy in bits, we get the usual formula:

$$H=\frac{n}{2}\log(2\pi e)+\tfrac{1}{2}\log|V|$$

where the logarithms are to the base 2.

Thus we see that there is a relationship between variance and entropy. Suppose now that $R_k$ is spanned by $x_n,\ldots,x_{n-k+1}$. Any vector $x$ in $R_n$ may be decomposed into an $x_1\in R_k$ and an $x_2\perp R_k$ such that $x=x_1+x_2$. This implies that $Ax=Ax_1+Ax_2$.

Because $R_k$ and its orthogonal complement $R_k^{\perp}$ are spanned by proper vectors (these are orthogonal because $A$ is symmetric) $Ax_1\in R_k$ and $Ax_2\perp R_k$ for all $x$, that is, $R_k$ reduces $A$ into a matrix $A_1$ of order $k$ and a matrix $A_2$ of order $n-k$. $A_1$ acts only within $R_k$, $A_2$ only within $R_k^{\perp}$.

A consequence of this decomposition of $A$ by $R_k$ is the decomposition of the $n$-dimensional distribution $f$ (see 3.1.1.3) into a $k$-dimensional distribution

$$f_1(x_1)=\frac{|A_1|^{\frac{1}{2}}}{(2\pi)^{k/2}}e^{-\frac{1}{2}(x_1,A_1x_1)}$$

and an $(n-k)$-dimensional distribution

$$f_2(x_2)=\frac{|A_2|^{\frac{1}{2}}}{(2\pi)^{(n-k)/2}}e^{-\frac{1}{2}(x_2,A_2x_2)}$$

The decomposition of the density function $f$: $f(x)=f_1(x_1)\,.\,f_2(x_2)$ results in similar decompositions for variance and entropy: $|V|=|V_1|\,.\,|V_2|$ and $H=H_1+H_2$.

### 3.2. Data compression

In section 2.3.2 we discussed the possibility of a small subset of predicates saying almost as much as the whole set. In such a case we spoke of 'data compression'. An analogous problem may be posed for the normal distribution.

Suppose we have a projection of $x$ on an arbitrary $k$-dimensional subspace, is it possible to choose this subspace so that the variance of this projection is almost as much as that of $x$? Or, equivalently, that its entropy is almost as much as that of $x$? In that case we have a redundancy of dimensions and we achieve data compression by substituting the projection for $x$ itself.

Bellman's result (3.1.1) now becomes useful: it states that of all $k$-dimensional subspaces the one containing the largest part of total entropy is the one spanned by the proper vectors belonging to the $k$ largest proper values of $V$. Whether the largest part is actually large, depends on the distribution of the proper values. The more nearly they are equal, the less data compression is possible. $x_1$, the projection of $x$ on $R_k$, is a linear combination of projections on the proper vectors $x_n, \ldots, x_{n-k+1}$. These were called by Hotelling the 'principal components': They decompose $R_n$ in such a way that in the corresponding factorization of $|V|$ one factor is the largest and therefore the other is the smallest.

The fact, that the $k$-dimensional subspace containing the maximum part of the total entropy is the subspace spanned by the proper vectors belonging to the $k$ largest proper values of $V$, was derived in a paper by J. Tou and R. Heydorn (Tou 1967). They did not mention the fact that their problem and solution are only a restatement of Hotelling's well-known result on principal components.

A more general result has been obtained by Watanabe (1965) by showing that the Karhunen–Loève expansion has a similar entropy-extremizing property. The greater generality lies in the fact that this expansion may be used for samples as well as for distributions.

### 3.3. Excess-entropy and weight of evidence

Let $H_0(H_1)$ be the hypothesis that the random variable $X$ is from the population with probability density function $f_0(f_1)$ and suppose that one observation, $x$, is made on this random variable. From the definition of conditional probability:

$$\Pr\{H_i|x\} = \frac{\Pr\{H_i \wedge x\}}{\Pr\{x\}} = \frac{\Pr\{x|H_i\} \cdot \Pr\{H_i\}}{\Pr\{x \wedge H_0\} + \Pr\{x \wedge H_1\}} \Rightarrow$$

$$\Pr\{H_i|x\} = \frac{f_i(x) \cdot \Pr\{H_i\}}{f_0(x) \cdot \Pr\{H_0\} + f_1(x)\Pr\{H_1\}} \quad \text{for} \quad i=0,1.$$

$$\frac{\Pr\{H_0|x\}}{\Pr\{H_1|x\}} = \frac{f_0(x) \cdot \Pr\{H_0\}}{f_1(x) \cdot \Pr\{H_1\}}.$$

I.J. Good, in his study on probability and the weight of evidence (Good 1950), introduced the quantities:

$$O(H_0/x)=\frac{\Pr\{H_0|x\}}{\Pr\{H_1|x\}} \text{ and } O(H_0)=\frac{\Pr\{H_0\}}{\Pr\{H_1\}}$$

to represent, respectively, the odds of $H_0$ given $x$ and the initial odds of $H_0$. Their ratio he called the factor in favour of the hypothesis $H_0$ in virtue of the result of the observation. Its logarithm is his 'weight of evidence':

$$\log\frac{f_0(x)}{f_1(x)}=\log\frac{\Pr\{H_0|x\}}{\Pr\{H_1|x\}}-\log\frac{\Pr\{H_0\}}{\Pr\{H_1\}} \tag{1}$$

This is a random variable, depending on $X$. Kullback (1968) is concerned with an information-theoretical interpretation of the log likelihood-ratio (Alexander 1967). His integral $I(0:1)$ coincides with the expected weight of evidence in favour of $H_0$ under the assumption that $H_0$ is true:

$$I(0:1)=\int_x f_0(x)\,.\,\log\frac{f_0(x)}{f_1(x)}\,.\,\mathrm{d}x.$$

Let us now consider the case where we have a set $x$ of random variables partitioned as $x=\{x_0, x_1\}$. $f$ is supposed to be the probability density function of $x$; $g$ and $h$ are the marginal probability density functions of $x_0$ and $x_1$ respectively.

Consider the null hypothesis $H_0$ that $x_0$ and $x_1$ are dependent and the alternative hypothesis

$H_1: f(x)=g(x_0)\,.\,h(x_1)$ for all $x$. In this case:

$$\begin{aligned}
I(0:1)&=\int_x f(x)\,.\,\log\frac{f(x)}{g(x_0)h(x_1)}\,\mathrm{d}x\\
&=\int_x f(x)\log f(x)\,\mathrm{d}x-\int_x f(x)\log g(x_0)\,\mathrm{d}x-\\
&\qquad\qquad\int_x f(x)\log h(x_1)\,\mathrm{d}x\\
&=-H(f)-\int_{x_0}\left[\int_{x_1} f(x)\,\mathrm{d}x_1\right]\log g(x_0)\,\mathrm{d}x_0-\\
&\qquad\qquad\int_{x_1}\left[\int_{x_0} f(x)\,\mathrm{d}x_0\right]\log h(x_1)\,\mathrm{d}x_1\\
&=H(g)+H(h)-H(f)\\
I(0:1)&=C(x_0,x_1).
\end{aligned}$$

In 2.2 we pointed out that excess-entropy may be regarded as a measure of dependence. In this particular situation we may be more specific. It is the expected weight of evidence in favour of dependence supposing that a particular form of dependence exists.

### 3.4. Clustering

Let $f(X)$ be the normal density function of an $n$-dimensional random vector $X=(x_1, \ldots, x_n)$. Let $X$ be partitioned as $(X_1, X_2)$ with $X_1=(x_1, \ldots, x_k)$ and $X_2=(x_{k+1}, \ldots, x_n)$ and let the corresponding partition of $V$ be:

$$V=\begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix}.$$

For the excess-entropy between $X_1$ and $X_2$ we find

$$C(X_1,X_2)=H(X_1)+H(X_2)-H(X)$$
$$=\tfrac{1}{2}\log\frac{|V_{11}|\cdot|V_{22}|}{|V|}.$$

This quantity is defined for any nonsingular matrix $V$ and any partition in it. When $V$ is reduced into $V_{11}$ and $V_{22}$ (when there are only zero elements in $V_{21}$), it is zero. It may therefore be used to indicate to what extent $V$ is almost reduced as is in fact done when using the likelihood ratio test for independence between $X_1$ and $X_2$. In 1.1 we saw that the fact that the blocks $A_{11}$ and $A_{22}$ do not reduce the matrix manifests itself as the transfer of the error in one partial approximation to the other and vice versa. Especially in view of 1.2, we tentatively described this phenomenon as 'information transfer'. In the special case where the matrix is symmetric and positive definite we have shown that this description is compatible with the mathematical definition of information.

## 4. ENTROPY IN MARKOV CHAINS

Let us consider a Markov chain $M$ with a finite number $n$ of *states* $s$ and a discrete *time parameter* $t$; $s_t=j$ means that $M$ is in state $j$ at time $t$. For every value $t$ of the time parameter there is a probability distribution over the states: $\Pr\{s_t=j\}=a_j^t$.

$M$ must be in some state: $\sum_{j=1}^{n} a_j^t=1$ for all $t$.

We will also use the matrix $P$ of transition probabilities whose elements are: $p_{ji}=\Pr\{s_t=j \mid s_{t-1}=i\}$. These we will suppose to be independent of time. $P$ connects successive distribution vectors $A_t^T=(a_1^t, \ldots, a_n^t)$ in the following way:

$$a_j^t=\sum_{i=1}^{n} p_{ji}a_i^{t-1} \quad \text{for} \quad j=1, \ldots, n \quad \text{or} \quad A_t=PA_{t-1}. \tag{1}$$

Columns of $P$ add up to unity (if $M$ is in any state $i$ at time $t-1$, it is certain to be in some state at time $t$), so we may define the *conditional entropy*:

$$H_i=-\sum_{j=1}^{n} p_{ji}\log p_{ji}$$

under condition that $M$ is in state $i$.

If we have any probability distribution $A^T=(a_1, \ldots, a_n)$ over the states of $M$ we may consider its elements as weights to produce a weighted average of the conditional entropies $H_i$:

$$H=\sum_{i=1}^{n} a_i H_i. \tag{2}$$

Many interesting Markov chains have the property that $\lim_{t\to\infty} A_t$ exists for every probability distribution and is independent of it. The entropy (2) obtained by taking $A=\lim_{t\to\infty} A_t$ is, in information theory, defined to be the entropy of the Markov chain (see, for instance, Khinchin 1957).

#### 4.1. Fusing two states

Suppose it can no longer be decided whether $s_t=j$ or whether $s_t=k$, but only whether $s_t=j$ or $s_t=k$. Then we say that states $j$ and $k$ are *fused*, say into $j'$. We see at once that:

$$a_{j'}=a_j+a_k \tag{3}$$

and

$$p_{j'i}=p_{ji}+p_{ki} \tag{4}$$

$p_{ij'}$ is obtained from (1) as follows:

$$a_i=\sum_{m\neq j,k} p_{im}a_m+p_{ij}a_j+p_{ik}a_k \quad \text{for} \quad i\neq j,\ i\neq k.$$

If we put

$$p_{ij'}=\frac{p_{ij}a_j+p_{ik}a_k}{a_j+a_k}, \tag{5}$$

we get

$a_i=\sum_{m\neq j'} p_{im}a_m+p_{ij'}a_{j'}$, as it should be.

Similarly for the case $i=j$ or $i=k$:

$$\begin{aligned} a_j&=\sum_{m\neq j,k} p_{jm}a_m+p_{jj}a_j+p_{jk}a_k \\ a_k&=\sum_{m\neq j,k} p_{km}a_m+p_{kj}a_j+p_{kk}a_k \\ \hline a_{j'}&=\sum_{m\neq j',k} p_{j'm}a_m+p_{j'j}a_j+p_{j'k}a_k \end{aligned} \tag{6}$$

If we put

$$p_{j'j'}=\frac{p_{j'j}a_j+p_{j'k}a_j}{a_j+a_k},$$

the last two terms may be replaced by $p_{j'j'}a_{j'}$, as they should be.

To summarize, the effect of fusing two states, $j$ and $k$, is to replace $a_j$ by $a_j+a_k$, the $j$th row of $P$ by the sum of the $j$th and $k$th rows and the $j$th column of $P$ by the weighted sum of the $j$th and $k$th columns with weights $a_j/(a_j+a_k)$ and $a_k/(a_j+a_k)$ respectively. Finally, $a_k$, the $k$th row of $P$ and the $k$th column of $P$ are deleted.

### 4.2. Excess-entropy as a measure of clustering

Fusion of two states may occur in any chain having not less than that number of states. The result is a chain again, where two states, if available, may be fused again. In short, as many states as are present may be fused.

Consider the sets of states $X=\{1, 2, \ldots, j\}$ and $Y=\{j+1, \ldots, n\}$. Besides the original chain $M$ with states $\{X \cup Y\}$ we also consider the chain $M_x$ with states $\{X, y\}$ and the chains $M_y$ with states $\{x, Y\}$ where $y(x)$ is the state resulting from fusion of all states of $Y(X)$.

$$\begin{array}{c} \\ X \\ Y \end{array}\begin{array}{c} \overbrace{\phantom{p_{11}\cdots p_{1j}}}^{X} \quad \overbrace{\phantom{p_{1,j+1}\cdots p_{1n}}}^{Y} \\ \begin{pmatrix} p_{11} & \cdots & p_{1j} & p_{1,j+1} & \cdots & p_{1n} \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{j1} & & p_{jj} & p_{j,j+1} & & p_{jn} \\ p_{j+1,1} & \cdots & p_{j+1,j} & p_{j+1,j+1} & \cdots & p_{j+1,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{n1} & \cdots & p_{nj} & p_{n,j+1} & \cdots & p_{nn} \end{pmatrix} \end{array}$$

$M$'s matrix of transition probabilities

$$\begin{pmatrix} p_{11} & \cdots & p_{1j} & p_{1y} \\ \vdots & & \vdots & \vdots \\ p_{j1} & \cdots & p_{jj} & p_{jy} \\ p_{y1} & \cdots & p_{yj} & p_{yy} \end{pmatrix}$$

$M_x$'s matrix of transition probabilities (states of $Y$ fused into $y$)

$$\begin{pmatrix} p_{xx} & p_{x,j+1} & \cdots & p_{xn} \\ p_{j+1,x} & p_{j+1,j+1} & \cdots & p_{j+1,n} \\ \vdots & \vdots & & \vdots \\ p_{nx} & p_{n,j+1} & & p_{nn} \end{pmatrix}$$

$M_y$'s matrix of transition probabilities (states of $X$ fused into $x$).

Applying formulae (3)–(6) for fusing states of $X$ and also for states of $Y$ we find:

$$a_x=\sum_{i\in X} a_i;\ a_y=\sum_{i\in y} a_i$$

$$p_{iy}=\sum_{j\in y}\frac{a_j}{a_y}p_{ij},\ i\in X \quad \text{and} \quad p_{ix}=\sum_{j\in X}\frac{a_j}{a_x}p_{ij},\ i\in Y.$$

Let us now introduce the quantities:

$$T_{xx} = -\sum_{i\in X} a_i \sum_{k\in X} p_{ki} \log p_{ki}, \quad T_{xy} = -\sum_{i\in X} a_i \sum_{k\in Y} p_{ki} \log p_{ki},$$

$$T_{yx} = -\sum_{i\in Y} a_i \sum_{k\in X} p_{ki} \log p_{ki}, \quad T_{yy} = -\sum_{i\in Y} a_i \sum_{k\in Y} p_{ki} \log p_{ki}.$$

According to (2) we then have for the entropy of $M$:

$$H = T_{xx} + T_{xy} + T_{yx} + T_{yy}.$$

We will now obtain inequalities for the 'cross terms' $T_{xy}$ and $T_{yx}$.

$$T_{xy} = -\sum_{k\in Y} a_x \sum_{i\in X} \frac{a_i}{a_x} p_{ki} \log p_{ki}$$

Application of 2.2.2 to the inner sum, where this time $\lambda_i = \frac{a_i}{a_x}$ and $f(x) = x \log x$, yields:

$$T_{xy} \leqslant -\sum_{k\in Y} a_x p_{kx} \log p_{kx}$$

and similarly

$$T_{yx} \leqslant -\sum_{k\in X} a_y p_{ky} \log p_{ky}$$

'$H(X)$' and '$H(Y)$' will be used to denote the entropies $M_x$ and $M_y$ respectively. According to (2) we have:

$$H(X) = T_{xx} - \sum_{k\in X} a_y p_{ky} \log p_{ky} - \sum_{i\in X} a_i p_{yi} \log p_{yi} - a_y p_{yy} \log p_{yy}.$$

Because of (7) it follows that $H(X) \geqslant T_{xx} + T_{yx}$.

Similarly we find that $H(Y) \geqslant T_{yy} + T_{xy}$, whence the main result:

$$H \leqslant H(X) + H(Y). \tag{8}$$

Again, as in the case of probability schemes and object–predicate tables, we may regard the concomitant *excess-entropy*:

$$C(X, Y) = H(X) + H(Y) - H \geqslant 0 \tag{9}$$

as a measure of dependence, this time between states of $X$ and states of $Y$.

A Markov chain may have a clustering structure in the sense that if it is in a state of $X(Y)$ at time $t$, it has a very small probability of being in $Y(X)$ at time $t+1$. It is clear that this is the more so as $p_{xx}$ and $p_{yy}$ are closer to 1. The excess-entropy defined in (9) is one possible measure of such clustering. When we consider all Markov chains with a partition $\{X, Y\}$ of the set of $n$ states, where $p_{xx} < 1$ and $p_{yy} < 1$, then the equality (9) is sharp, that is, 0 is the greatest lower bound of $C(X, Y)$. Thus we see that $C(X, Y)$ may be used as a measure of clustering, smaller values corresponding to stronger clustering.

If we are given the probability matrix $P$ of some Markov chain, the stationary probability distribution $A$ may be obtained by solving the system of linear equations $(P-I)A = 0$.

Suppose that we have a clustering structure in the above sense, namely that $p_{xx}$ and $p_{yy}$ are near unity. Then the system may profitably be solved by

the iterative method mentioned in 1.1. Again, as in the case of a positive definite matrix, we see that the cause of continuation of iteration, which we tentatively called 'information transfer', may be explained in terms of entropy which is fundamental to the mathematical definition of information.

**Acknowledgement**

The research reported here was supported by the Hugo de Vries Laboratory for Systematic Botany in the University of Amsterdam as part of its program to develop mathematical methods for analysis of plant-ecological data.

## REFERENCES

Alexander, C. (1967) *Notes on the synthesis of form.* Cambridge, Mass.: Harvard University Press.

Bellman, R. (1960) *Introduction to matrix analysis.* New York: McGraw-Hill.

Good, I.J. (1950) *Probability and the Weighing of Evidence.* London: Griffin.

Khinchin, A.I. (1957) *Mathematical foundations of information theory.* New York: Dover.

Kullback, S. (1968) *Information theory and statistics.* New York: Dover.

Mowshowitz, A. (1967) *Entropy and the complexity of graphs* (thesis). University of Michigan.

Rescigno, A. & Maccacaro, G.A. (1961) Information Content of Biological Classifications, in *Information theory: fourth London symposium* (ed. Cherry, C.). London: Butterworths.

Watanabe, S. (1960) Information theoretical analysis of multivariate correlation. *IBM Journal of Res. and Dev.*, 66–82.

Watanabe, S. (1965) Karhunen-Loève expansion and factor analysis. *Proc. 4th conf. on information theory*, 635–59. Prague.

Tou, J. (ed.) (1967) *Computer and information sciences-II.* New York: Academic Press

Williams, W. & Lambert, J.M. (1959) Multivariate methods in plant ecology I. *The Journal of Ecology*, **47**, 83–101.

# PATTERN RECOGNITION

# 21

# A Grammar for the Topological Analysis of Plane Figures

O. P. Buneman
Department of Machine Intelligence and Perception
University of Edinburgh

## INTRODUCTION

The topology of a plane black and white figure presented on a finite mesh or retina can be simply described by a tree. An algorithm is presented which determines this tree from information about the figure which is picked up during a single scan of the retina. The algorithm strongly resembles a grammar in that it consists of a generative grammar (Chomsky 1969) and a procedure for choosing which rule of the grammar to operate. Minsky and Papert (1969) have discussed similar problems of topological analysis in their

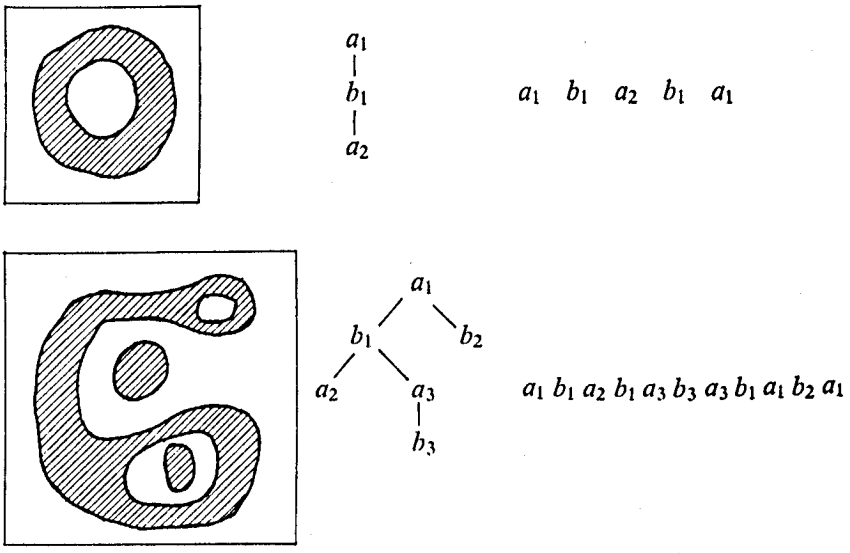



Figure 1. Two figures with their description by trees and by strings

book *Perceptrons*. While they are concerned primarily with parallel computation, they do give a neat algorithm which operates by a combination of scanning and boundary tracing, for determining whether or not a figure is connected.

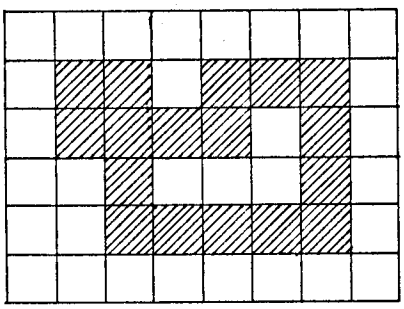
Examples of this tree description are given in figure 1. Each node refers to a connected component of the figure, 'a' for white and 'b' for black; the topmost node represents the outermost (white) component and the downward branches represent successive inclusions. It is fairly clear that this description is adequate in a topological sense and a simple proof of this is given in the appendix.

Any such tree may also be described by a string of its nodes in the manner also shown in figure 1. These strings are simple to derive: each is the set of nodes encountered on an exhaustive walk through the tree. The purpose of this paper is to define a grammar whose rewriting rules are operated by a scan of the retina and whose terminal strings are precisely these topological descriptions.

## THE SCAN

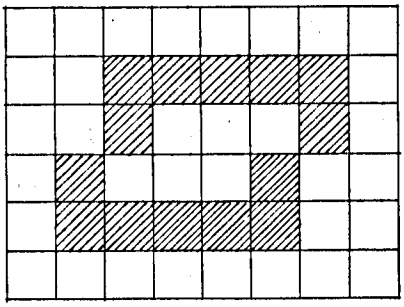
The retina is supposed to be a square mesh, and we impose the convenient but not fundamental restriction that it shall have white borders. We also have to introduce an asymmetry between black and white in that we allow white squares to be connected by their edges or vertices, but black squares only by their edges (see figure 2). This asymmetry could be avoided by using a hexagonal bee retina or by restricting further the class of figures. The operations on such a figure which do not alter the topology have been given by Hilditch (1969).

The scan records events that occur on adjacent lines of the retina; a black segment on one line of the retina may be connected to one or more of the black segments on the line beneath it; it may also be connected to no other



2 white components

1 black component

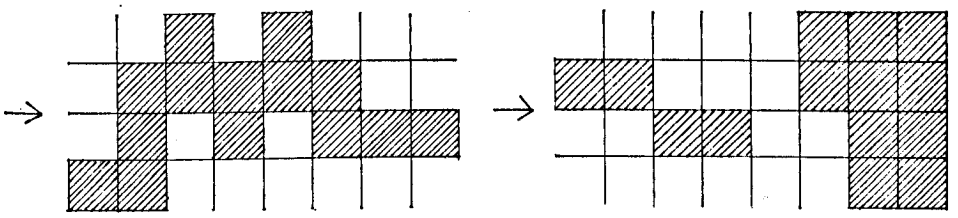


2 black components

1 white component

Figure 2

such segments. We shall call this number (always a non-negative integer) the *branching* number, and we can, in a similar way, associate a branching number with each white segment. (See figure 3.) The only information that we shall use in the implementation of the grammar is the sequence of branching numbers that is recorded as the scan proceeds television-fashion over the retina.



A black segment with branching number 3.

A white segment with branching number 2.

Figure 3

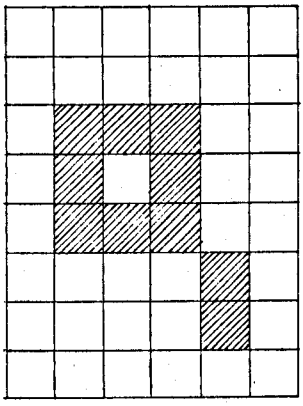
## THE GRAMMAR

Each new branching number that is read off the retina will cause a string of symbols to be rewritten. The way in which this rewriting takes place will depend on both the branching number and the current form of the string. The *symbols* of which this string is composed will be the set $A_1, A_2, A_3, \ldots B_1, B_2, B_3, \ldots a_1, a_2, a_3, \ldots b_1, b_2, b_3, \ldots$. Of these we shall call the lower case letters $a_1, a_2, \ldots b_1, b_2, \ldots$ the *terminal* symbols. A possible string will be, for example, $A_1\ b_1\ a_2\ B_1\ A_1$. The rewriting rules will be such that at the end of the scan the string will be the topological description exemplified in figure 1.

The rewriting rules will also ensure that, at the beginning of each line of scan, the nonterminal (upper case) symbols will refer in their order to the segments encountered on that line. If, for example, we found that at the beginning of a line of scan we had already generated the string $A_1\ b_1\ a_2\ B_1\ A_1$, then we should expect to find successively a white, black, and white segment on that line. An example of a simple figure and its sequence of branching numbers is given in figure 4.

The branching number 1 determines the trivial rewriting rule, namely that nothing is altered. Any other branching number will cause us to rewrite either the nonterminal symbol which refers to the segment with that number, or a substring containing it. In expounding the rules we shall underline this symbol. This is necessary as there may be several occurrences of the same symbol in a string.

Before embarking on a precise formulation of the rules which may at first seem somewhat opaque, an example is given which is the sequence of strings determined by the branching numbers of figure 4. It may be found useful to examine this in some detail to understand the basic operations involved.



1/2/121/11011/102/111/101

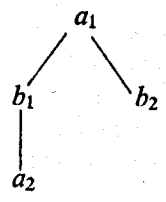
Figure 4. A simple figure and its branching numbers. The strokes indicate the ends of lines of scan, but they are not needed to operate the rules.

Applications of the trivial rule have been omitted so that the underlined symbols are those which refer to segments whose branching numbers *next* determine application of a non-trivial rule. The following example is the sequence of strings that would be produced by the branching numbers given in figure 4.

*Example* 1

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| start. | $A_1$ | | | | | | | |
| Scan line: | 1. | $A_1$ | $\underline{B}_1$ | $A_1$ | | | | |
| | 2. | $A_1$ | $B_1$ | $\underline{A}_2$ | $B_1$ | $A_1$ | | |
| | 3. | $A_1$ | $b_1$ | $a_2$ | $\underline{B}_1$ | $A_1$ | | |
| | 4. | $a_1$ | $b_1$ | $a_2$ | $b_1$ | $\underline{A}_1$ | | |
| | 5. | $a_1$ | $b_1$ | $a_2$ | $b_1$ | $A_1$ | $\underline{B}_2$ | $A_1$ |
| | | $a_1$ | $b_1$ | $a_2$ | $b_1$ | $a_1$ | $b_2$ | $\underline{A}_1$ |
| end. | | $a_1$ | $b_1$ | $a_2$ | $b_1$ | $a_1$ | $b_2$ | $a_1$ |

Tree:



It will be seen from example 1 that the strings have been constructed with two basic operations: the insertion of nonterminal symbols and the reduction of nonterminal to terminal symbols. These are implemented at the appearance of components on the scan line and at their disappearance. It will be necessary to introduce one more operation, which is the identification of symbols. This is because a single component of the figure may look like several components above the line of scan. Consideration of a simple U-shaped figure will show that, until the lower part of the figure is reached, there is nothing on the scan to say that this is not two components.

The rewriting rules are now given. These rules operate on substrings and it will be convenient to use the letters $\alpha$, $\beta$ to denote substrings of terminal symbols and $\Sigma$ to denote an arbitrary substring.

1. The start symbol is $A_1$.

2. For branching number 1:

$\underline{A_k} \rightarrow A_k$. (The trivial rule.)

3. For branching number $n>1$:

$\underline{A_k} \rightarrow A_k\, B_e\, A_k\, B_{e+1}\, A_k \ldots B_{e+n-1}\, A_k$

where the integers $e, e+1, \ldots e+n-1$, do not suffix any other $B$ or $b$.

4. For branching number 0:

$B_i \alpha\, \underline{A_k} \beta\, B_i \rightarrow b_i \alpha\, a_k \beta\, B_i$.

5. For branching number 0:

$A_k \Sigma\, B_i \alpha\, \underline{A_k} \beta\, B_j \rightarrow \alpha A_k \Sigma\, \beta B_j$ $\quad i \neq j$

or $\quad B_i \alpha\, \underline{A_k} \beta\, B_j \Sigma\, A_k \rightarrow \beta B_j \Sigma\, \alpha A_k$.

also re-suffix all $B_i\, b_i\, B_j\, b_j$ with the least of $i$ and $j$.

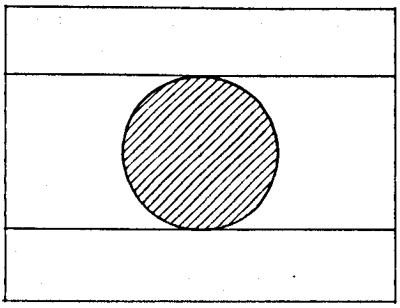
6. At the end of the scan, rewrite the remaining $A_1$ as $a_1$.

In addition rules 2*, 3*, and 4* are like 2, 3, and 4 with $A$s and $B$s (white and black) interchanged.

The procedure for underlining a nonterminal symbol can also be made formal: at the application of any rule always underline the next nonterminal symbol. Here the initial $A_1$ of a string can be thought of as following the final $A_1$ of the previous string. If this is done, the only information needed from the retina is the *uninterrupted* sequence of branching numbers.

These rules will be discussed in detail later, meanwhile we embark on some more examples. Only scan lines which give rise to nontrivial rules are drawn in figures and, as before, only the nonterminal symbols which are rewritten by nontrivial rules are underlined.

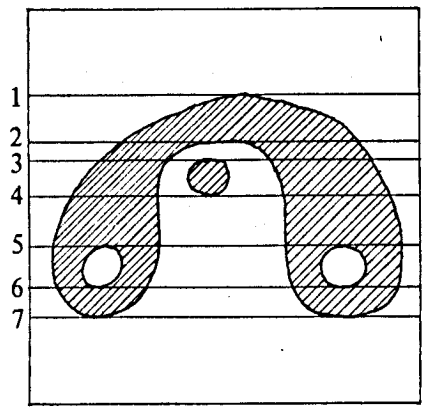
*Example 2*



| | | | | |
|---|---|---|---|---|
| Start. | $\underline{A}_1$ | | | by rule 1 |
| 1. | $A_1$ | $\underline{B}_1$ | $A_1$ | 3 |
| 2. | $a_1$ | $b_1$ | $\underline{A}_1$ | 4* |
| End. | $a_1$ | $b_1$ | $a_1$ | 6 |

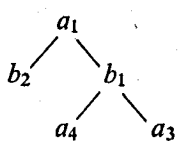
Tree: $a_1$ / $b_1$

*Example 3*

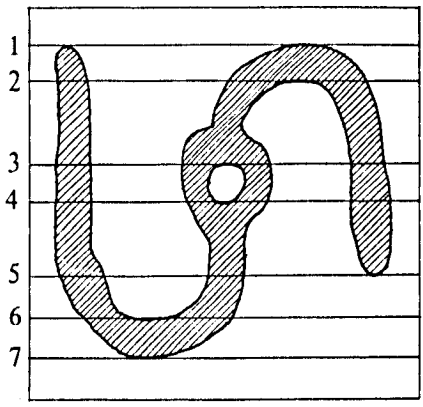



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Start. | $\underline{A}_1$ | | | | | | | | | | by rule 1 |
| 1. | $A_1$ | $\underline{B}_1$ | $A_1$ | | | | | | | | 3 |
| 2. | $A_1$ | $B_1$ | $\underline{A}_2$ | $B_1$ | $A_1$ | | | | | | 3* |
| 3. | $A_1$ | $B_1$ | $A_2$ | $\underline{B}_2$ | $A_2$ | $B_1$ | $A_1$ | | | | 3 |
| 4. | $A_1$ | $\underline{B}_1$ | $a_2$ | $b_2$ | $A_2$ | $B_1$ | $A_1$ | | | | 4* |
| 5. | $A_1$ | $B_1$ | $A_3$ | $B_1$ | $a_2$ | $b_2$ | $A_2$ | $\underline{B}_1$ | $A_1$ | | 3* |
| | $A_1$ | $B_1$ | $\underline{A}_3$ | $B_1$ | $a_2$ | $b_2$ | $A_2$ | $B_1$ | $A_4$ | $B_1$ | $A_1$ 3* |
| 6. | $A_1$ | $b_1$ | $a_3$ | $B_1$ | $a_2$ | $b_2$ | $A_2$ | $B_1$ | $\underline{A}_4$ | $B_1$ | $A_1$ 4 |
| | $A_1$ | $b_1$ | $a_3$ | $\underline{B}_1$ | $a_2$ | $b_2$ | $A_2$ | $b_1$ | $a_4$ | $B_1$ | $A_1$ 4 |
| 7. | $a_1$ | $b_2$ | $A_1$ | $b_1$ | $a_4$ | $b_1$ | $a_3$ | $\underline{B}_1$ | $A_1$ | | 5* |
| | $a_1$ | $b_2$ | $a_1$ | $b_1$ | $a_4$ | $b_1$ | $a_3$ | $b_1$ | $\underline{A}_1$ | | 4* |
| End: | $a_1$ | $b_2$ | $a_1$ | $b_1$ | $a_4$ | $b_1$ | $a_3$ | $b_1$ | $a_1$ | | 6 |

Tree:



*Example 4*




| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Start. | $\underline{A}_1$ | | | | | | | | | by rule 1 |
| 1. | $A_1$ | $B_1$ | $A_1$ | $\underline{B}_2$ | $A_1$ | | | | | 3 |
| 2. | $A_1$ | $B_1$ | $A_1$ | $\underline{B}_2$ | $A_2$ | $B_2$ | $A_1$ | | | 3* |
| 3. | $A_1$ | $B_1$ | $A_1$ | $B_2$ | $\underline{A}_3$ | $B_2$ | $A_2$ | $B_2$ | $A_1$ | 3 |
| 4. | $A_1$ | $B_1$ | $A_1$ | $b_2$ | $a_3$ | $B_2$ | $A_2$ | $\underline{B}_2$ | $A_1$ | 4 |
| 5. | $A_1$ | $B_1$ | $\underline{A}_1$ | $b_2$ | $a_3$ | $B_2$ | $A_1$ | | | 5* |
| 6. | $A_1$ | $b_1$ | $a_3$ | $\underline{B}_1$ | $A_1$ | | | | | 5 |
| 7. | $a_1$ | $b_1$ | $a_3$ | $b_1$ | $\underline{A}_1$ | | | | | 4* |
| End. | $a_1$ | $b_1$ | $a_3$ | $b_1$ | $a_1$ | | | | | 6 |

Tree: $a_1$
|
$b_1$
|
$a_3$

## DISCUSSION

The notion that grammars may be applied to pictures is not new, but a certain amount of caution is needed in attempting to carry the analogy between pictorial and linguistic analysis too far. It is not clear, for example, that there is anything in visual perception that corresponds to our ability to assess the grammatical correctness of an utterance. Nor can we necessarily carry the analogy as far as substituting 'picture' for 'utterance' for we do not understand the former to refer to something that is always generated by a human being. Thus even if we could overcome the objection that a picture is not a string of symbols, it is difficult to conceive of a grammar that would generate all pictures and be less than trivial.

It is possible that a generative grammar, or something that resembles a grammar could be used in the description of a picture. Begging the all-important question of what types of description are relevant to the way we interpret what we see, it is reasonable to assert that any description will have

some sort of structure. This is, in a sense, equivalent to the assertion that an English sentence has some structure that is independent of its meaning. Among the rewriting rules given earlier, rules 1, 3, 3*, 4, 4*, and 6 are sufficient to generate all the distinct terminal strings *without* reference to a figure. These rules then constitute a grammar for this limited language of topological descriptions, but they are not sufficient for effecting a description from a figure. This algorithm can then be described as having two components: a grammar and a control, the former being the set of rewriting rules, the latter being the rules that determine their use.

There are various objections to formally identifying the rewriting rules of the algorithm given here with a context-sensitive generative grammar; the really serious difficulty is that the rewriting rules operate on substrings that are not explicit. For example, rule 5 operates on a string that contains an arbitrary substring; moreover rule 5 can also delete symbols. The rewriting rules of the algorithm can be formulated as a transformational grammar, and this is done in the appendix, though the main purpose of this is to make it easy to show that these rules work. The difficulty is then that it becomes harder to keep the reference of the 'tree' to the scan, which is automatically done by the nonterminal symbols.

The topological interest of this grammar is secondary, though the interested reader may care to amuse himself by constructing similar grammars for figures on other two-dimensional surfaces. It is perhaps also of interest that certain properties of strings can be proved either from examination of the rules of the grammar or by invoking topology. For example, when rule 5 is applied, the existence of another $A_k$ in the string can be deduced either from properties of the rules, or by the Jordan Curve Theorem.

Mark Steedman (1969) has written a very elegant implementation of this for a computer. Steedman's program operates with a similar but not identical method to that already given. His program not only produces the topological tree, but also allows one to predicate about various topological properties. For example, one may ask whether two points in a figure are in the same component, or whether two figures are topologically equivalent. The program will also decompose a figure into its connected components.

**Acknowledgement**

This work was supported by a research grant from the Royal Society.

## REFERENCES

Chomsky, N. (1969) in *Handbook of Mathematical Psychology*, Vol. II. New York: Wiley & Sons.

Hilditch, C.J. (1969) An application of Graph Theory in pattern recognition. *Machine Intelligence 3*, pp. 325–47 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Minsky, M. & Papert, S. (1969) *Perceptrons*. Cambridge, Mass: M.I.T. Press.

Steedman, M.J. (1969) Dissertation. Department of Machine Intelligence and Perception Diploma. Edinburgh University.

## APPENDIX

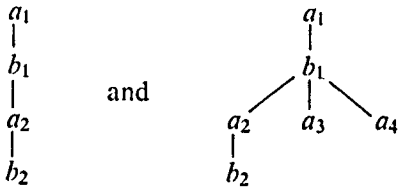
### Similar trees describe isotopic figures

Two trees are said to be similar if there is a bijection between their respective sets of nodes which commutes with the branching relations. By suitably chopping the corners off black squares to avoid the difficulties mentioned earlier, each component becomes a piecewise linear manifold and any such manifold is a disc with $n$ holes. Moreover any two such manifolds with a common outer boundary are isotopic in the sense that there is an isotopy of one which preserves the outer boundary and maps the inner boundaries of one to the inner boundaries of the other in any given order. The isotopy can now be constructed inductively on the level of the tree. The topmost component is in each case a disc with, say, $n$ holes. Construct an isotopy of the type just mentioned so that these holes are mapped in the order given by the bijection. Now perform a similar operation for each of the nodes on the next level down, and so on.

In this case 'isotopic' can be replaced by 'homeomorphic' but similar constructions can be done on other retinas (e.g., an annulus) where similar trees preserve the stronger relation of isotopy.

### The algorithm as a set of transformation rules

Any terminal or nonterminal string can be regarded as a tree provided we first rewrite its nonterminal as the corresponding terminal symbols. At the ends of scans 3 and 6 in example 4 we could in this manner construct the trees:




But this construction has lost the reference of the nodes to the sequence of branching numbers. In order to recover this we attach to each of these trees the functions:

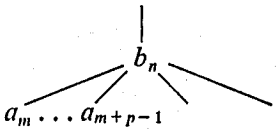
$$\Phi: \begin{array}{l} 1\to a_1 \\ 2\to b_1 \\ 3\to a_2 \\ 4\to b_2 \\ 5\to a_2 \\ 6\to b_1 \\ 7\to a_1 \end{array} \quad \text{and} \quad \Phi': \begin{array}{l} 1\to a_1 \\ 2\to b_1 \\ 3\to a_2 \\ 4\to b_1 \\ 5\to a_1 \end{array}$$

Thus, $\Phi(3)=a_2$ means that $a_2$ refers to the third segment of the scan line. Such a function is called a *labelling* and a tree together with a labelling is called a *labelled tree.* The rules of the algorithm can now be interpreted as transformations of the tree together with rules for modifying the labelling. In this way we can examine the rules:

*Rule 1* initiates the tree with $a_1$ and associates with it the function $\Phi(1)=a_1$.

*Rules 2 and 2** change neither the tree nor the labelling.

*Rules 3 and 3** cause new branches to appear on a labelled node. For example, if the $k$th segment has branching number $p>1$ and corresponds, say, to $b_n$ so that $\Phi_k(k)=b_n$. The tree is then altered to:




and the labelling changes to $\Phi'$ as follows:

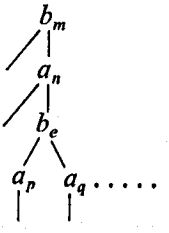
$$\Phi'(k)=b_n,\ \Phi'(k+1)=a_m,\ \Phi'(k+2)=b_n \ldots$$
$$\Phi'(k+2p-1)=a_{m+p-1},\ \Phi'(k+2p)=b_n.$$

Also $\Phi'(i)=\Phi(i)$ for $i\leqslant k$

$\Phi'(i)=\Phi(i-2p)$ for $i\geqslant k+2p$.

*Rules 4 and 4** operate when the labelling jumps down a branch and immediately back. The lower node is unlabelled in an obvious way and the tree is left unchanged.

*Rules 5 and 5** operate when the labelling jumps through a node, or jumps up and then down. It identifies two nodes and shifts their subtrees if necessary. If for the labelled subtree,

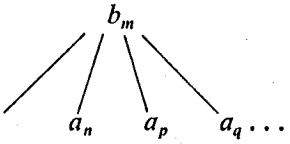



$\Phi(k-1)=b_m$

$\Phi(k)\ =a_n$

$\Phi(k+1)=b_e$

the $k$th segment has branching number 0, *Rule 5* reconstructs this tree as:




so that the whole subtree that is topped $b_e$ is shifted and attached to $b_m$. The labelling is altered as for rule 4.

*Rule 6* unlabels the tree; at the last line of scan, the labelling will be $\Phi(1)=a_1$.

Having done this, it is not hard to prove that these rules work, though to do so in detail is rather lengthy. The labelled tree constitutes a 'hypothesis' about the figure from the information already picked up during the scan. One now has to check that the changes in this hypothesis produced by the application of a rule are consistent with the changes that are made by the observation of a new branching number. It is easier to do this by checking that this is true for a single component and working, as before, inductively on the level of the tree.

# 22

# Shape Analysis by Use of Walsh Functions

N. H. Searle
Metamathematics Unit
University of Edinburgh

## INTRODUCTION

Walsh functions, and other sets of orthogonal boolean functions, are useful tools in the approximation of a given function. In the case of Walsh functions there are two particular advantages over other methods of function approximation. Firstly, the coefficients to be used in expressing the given function as a linear combination of the Walsh functions can be computed by a parallel algorithm. Secondly, where the given function represents a shape, or other object with spatial meaning, there is an interpretation of the approximation procedure under which the function is analysed into its spatial components. The former advantage is of particular importance where the amount of information to be processed is large, as it is when dealing with visual input to a computer.

An optical image is normally presented to a digital computer as an array, or retina, of regular cells. Associated with each cell is a digitized value, which may correspond to the intensity of the original image, say. This set of values is the *retinal function.* The cells may be square or hexagonal. The latter will in general give a better fit to the original image, but for conceptual reasons it is simpler to discuss the processing of retinal functions in terms of the former.

A *shape* is a binary-valued retinal function. For the purposes of analysing a shape in terms of its spatial structure, Walsh coefficients may also be regarded as binary-valued, positive coefficients taking the value 1, and negative coefficients the value 0, say.

In a pattern recognition system there are normally three stages: input, transformation, and discrimination. The transformation of the input serves the purpose of providing a description on the basis of which discrimination, or recognition, can take place. Complete systems of artificial pattern recognition can only hope to rival human performance when they are enhanced by

reasoning and linguistic capabilities of a high order. In the absence of sufficiently well-developed techniques in these areas, the present paper concerns itself solely with the first stage of the transformation process, and the place that Walsh analysis might have in the process.

Certain problems in the field of pattern recognition have already been tackled with considerable degrees of success. An obvious example is optical character recognition. This is a simple problem in the sense that the domain and range of the patterns are reasonably restricted, and that the recognition process takes place on a single character at a time. The complexity of, say, giving a description of a scene as viewed by a robot through a television camera eye is clearly much greater. In such a situation a general purpose pattern recognition system would suggest itself, rather than a collection of special purpose recognition systems. The methods described here are not special purpose, nor do they claim to be more than possibly a small contribution to a general purpose system.

There are circumstances under which recognition of visual input is not required. For example, a botanist may wish to have an accurate numerical description of the shape of a leaf, in order that he can compare it with other leaves grown under differing conditions. For such an application, he would not be interested in a specification of the type of leaf. The Walsh analysis initially transforms the input data into a new, more meaningful, set of numerical data, and this, rather than a complete recognition-type description, may be sufficient for, say, statistical work.

## WALSH FUNCTIONS

The Walsh functions are a complete set of orthogonal functions (Walsh 1923). They can be derived in several distinct ways, which give rise to more than one ordering of the functions. Two orderings are of particular interest, one for its computational properties, and the other for its interpretative properties. The first definition, for the interval [0, 1), can be extended to the whole or part of the real line and to $n$ dimensions, and is essentially the same as that originally proposed by Walsh.

*Definition* 1. The Walsh functions are $f_0, f_1, \ldots f_r, \ldots$

$$f_0(x) = 1, \qquad 0 \leqslant x < 1$$

$$\text{For } k \geqslant 1, f_k(x) = \begin{cases} f_l(2x), & 0 \leqslant x < \frac{1}{2} \\ (-1)^{k+l} f_l(2x-1), & \frac{1}{2} \leqslant x < 1 \end{cases}$$

where $l = \left[\frac{k}{2}\right]$.

It can be seen from the above definition that the Walsh functions are step-functions, which take the value $+1$ or $-1$ on each of the $2^n$ equal subintervals of $[0, 1) - \left[\frac{m}{2^n} \quad \frac{m+1}{2^n}\right), m = 0, 1, \ldots, 2^n - 1$ — where, for a given Walsh function, $f_k(x)$, $n$ is the smallest integer such that $k < 2^n$.

Therefore, the functions may be adequately represented by patterns of $+1$s and $-1$s.

*Example*

$$
\begin{array}{lcccccccc}
f_0 = & 1 & & & & & & & \\
f_1 = & 1 & -1 & & & & & & \\
f_2 = & 1 & -1 & -1 & 1 & & & & \\
f_3 = & 1 & -1 & 1 & -1 & & & & \\
f_4 = & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
f_5 = & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
f_6 = & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
f_7 = & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1
\end{array}
$$

Note that each function is odd or even with respect to the midpoint of the interval and that the pattern $1\ 1\ -1\ -1$, say, represents the same function ($f_1$ in this case) as the pattern $1\ -1$; the difference being due to whether we consider $f_1$ to be one of the first four Walsh functions, or one of the first two.

The ordering of the functions which arises from this first definition of the functions will be referred to as the Walsh ordering, since it is identical to that produced by Walsh's own definition. It has the important property that the $k$th function, $f_k$, has exactly $k$ discontinuities on the interval $[0, 1)$.

If the patterns corresponding to the first $2^n$ Walsh functions are expanded so that they are all of length $2^n$ values, we can form the Walsh matrix, $W_n$. For example, when $n=3$, we obtain the following matrix, whose rows are the function values of the first eight Walsh functions:

$$
W_3 = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1
\end{pmatrix}
$$

It will be proved that $W_n$ is symmetrical for all positive integers $n$.

The next three definitions of the Walsh functions give orderings of the functions which are identical to one another. That they do so can be seen upon close inspection. In this new ordering the $k$th function does not in general have $k$ discontinuities on $[0, 1)$. Further, if $W'_n$ is the matrix of the Walsh function values in this new ordering, the $k$th row of $W'_n$ is not in general equal to the $k$th row of $W'_m$, where $m \neq n$, i.e., the $k$th function amongst the first $2^n$ Walsh functions is not also the $k$th amongst the first $2^m$ functions, in this ordering.

*Definition* 2. $W'_n$ is the matrix whose rows are the patterns of $+1$s and $-1$s which represent the first $2^n$ Walsh functions.

$$W'_0=(1)$$

$$\text{For } n\geqslant 1,\ W'_n=\begin{bmatrix} W'_{n-1} & W'_{n-1} \\ W'_{n-1} & -W'_{n-1} \end{bmatrix}$$

This result is due to Lechner (1961).

These matrices are sometimes referred to as Hadamard matrices, but, as the notation used above suggests, here they will be considered to be a modified version of the Walsh matrices. The rows of $W'_n$ are a permutation of the rows of $W_n$, i.e., the first $2^n$ Walsh functions are the same set of functions under both definitions 1 and 2.

*Example*

$$W_2=\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \qquad W'_2=\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

*Definition* 3. The Rademacher functions are orthogonal boolean functions which are not complete in $L^2$. If $R_0(x)$, $R_1(x), \ldots R_m(x), \ldots$ are the Rademacher functions, then $R_n(x)$ is a step-function which takes the values $+1$ and $-1$ alternately in the $n+1$ equal sub-intervals of $[0, 1)$. They can be extended to other intervals and to $n$ dimensions.

$$R(x)=1 \qquad 0\leqslant x<1$$

$$\text{For } n\geqslant 1,\ R_n(x)=\begin{cases} 1 & \dfrac{2k}{2^n}\leqslant x<\dfrac{2k+1}{2^n} \\ -1 & \dfrac{2k+1}{2^n}\leqslant x<\dfrac{2k+2}{2^n} \end{cases}$$

$$\text{for } k=0, 1, \ldots, 2^n-1.$$

As for the Walsh functions, the appropriate normalizing factor, $1/2^n$, has been omitted for clarity.

In fact, $R_n(x)=f_{2^n-1}(x)$ in the Walsh ordering; however, we are concerned here with the definition of the Walsh functions in terms of the Rademacher functions, rather than vice versa.

The Walsh functions can be defined as all possible products of Rademacher functions. No Rademacher function appears more than once in a given product, since $R_m(x) \cdot R_m(x)$ is unity everywhere, for all $m$.

To determine which Rademacher functions are to be used in forming a given Walsh function, we write:

$$f_0(x) = R_0(x)$$

and $$f_{n_1 n_2 \ldots n_m}(x) = R_{n_1}(x) \,.\, R_{n_2}(x) \,.\, \ldots \,.\, R_{n_m}(x)$$

where $0 < n_1 < n_2 < \ldots < n_m$, and $f_{n_1 n_2 \ldots n_m}(x) = f_a(x)$,

where $a$ has unity in the $n_1, n_2, \ldots, n_m$th places after the binary point in its binary expansion and zero elsewhere.

It can be shown that

$$f_a(x) \,.\, f_b(x) = f_{a \oplus b}(x)$$

where $0 \oplus 1 = 1 \oplus 0 = 1$ and $1 \oplus 1 = 0 \oplus 0 = 0$;
and also that

$$f_a(x) = (-1)^{\{a,x\}}$$

where $\{a, x\}$ is the number of places in which both $a$ and $x$ have unity in their binary expansions.

It follows immediately that $f_a(x) = f_x(a)$ for all suitable $a$ and $x$.

Thus, for all non-negative integers $n$, $W'_n$ is symmetrical.

*Definition* 4. Suppose $p$ is a pattern of numbers; in particular, a pattern of $+1$s and $-1$s representing the values of a Walsh function as described above. Then we define the basic patterns

$$\overline{0} = 1 \quad 1$$

and

$$\overline{1} = 1 \quad -1$$

and the pattern products

$\overline{0} \,.\, p = p \quad p$ (i.e., the pattern $p$ followed by a repetition of itself)

and

$\overline{1} \,.\, p = p \quad -p$ (i.e., the pattern $p$ followed by the pattern in which each element is the negation of the corresponding element in $p$).

*Example*

$$\begin{array}{lllllllll} & \overline{1} = 1 & -1 & & & & & & \\ \overline{0} \,.\, \overline{1} = & 1 & -1 & 1 & -1 & & & & \\ \overline{1} \,.\, \overline{0} \,.\, \overline{1} = & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \end{array}$$

It is easily seen that the first $2^n$ Walsh functions can be expressed as all possible products of $n$ factors, each of which is a basic pattern. In other words, we may define a Walsh function as a basic pattern or the pattern product of a Walsh function and a basic pattern.

Note that $\overline{d}_1 \,.\, \overline{d}_2 \,.\, \ldots \,.\, \overline{d}_m \,.\, \overline{0} = \overline{d}_1 \,.\, \overline{d}_2 \,.\, \ldots \,.\, \overline{d}_m$.

The table below demonstrates the pattern product ordering of the first eight Walsh functions, covering the cases $n = 0, 1, 2, 3$.

| Walsh ordering × | .000 | .001 | .010 | .011 | .100 | .101 | .110 | .111 | Pattern product | New order |
|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | $\bar{0}.\bar{0}.\bar{0}$ | 0 |
| 001 | 1 | 1 | 1 | 1 | −1 | −1 | −1 | −1 | $\bar{1}.\bar{0}.\bar{0}$ | 4 |
| 010 | 1 | 1 | −1 | −1 | −1 | −1 | 1 | 1 | $\bar{1}.\bar{1}.\bar{0}$ | 6 |
| 011 | 1 | 1 | −1 | −1 | 1 | 1 | −1 | −1 | $\bar{0}.\bar{1}.\bar{0}$ | 2 |
| 100 | 1 | −1 | −1 | 1 | 1 | −1 | −1 | 1 | $\bar{0}.\bar{1}.\bar{1}$ | 3 |
| 101 | 1 | −1 | −1 | 1 | −1 | 1 | 1 | −1 | $\bar{1}.\bar{1}.\bar{1}$ | 7 |
| 110 | 1 | −1 | 1 | −1 | −1 | 1 | −1 | 1 | $\bar{1}.\bar{0}.\bar{1}$ | 5 |
| 111 | 1 | −1 | 1 | −1 | 1 | −1 | 1 | −1 | $\bar{0}.\bar{0}.\bar{1}$ | 1 |

If we interpret the pattern product as a binary number, we obtain the new ordering, which is stated explicitly in the final column of the table. We shall refer to this as the pattern product ordering.

The concept of pattern products can be extended to $n$ dimensions, and as such can be used to define $n$-dimensional Walsh functions.

The pattern products in the penultimate column are closely related to the reflected binary code, and it is this relationship which gives the following result:

$f_{d_1 d_2 \ldots d_m}(x)$ in the pattern product ordering is

$f_{d_m . d_m \oplus d_{m-1} \ldots d_m \oplus d_{m-1} \oplus \ldots \oplus d_1}$ in the Walsh ordering.

A similar expression gives the inverse relationship.

It now follows that, in the Walsh ordering,

$$f_{d_1 d_2 \ldots d_m}(.\, r_1 r_2 \ldots r_m) = (-1)^{r_m \wedge d_1} f_{d_1 \oplus d_2\, d_1 \oplus d_3 \ldots d_1 \oplus d_m}(.\, r_1 r_2 \ldots r_m)$$

where $0 \wedge 1 = 1 \wedge 0 = 0 \wedge 0 = 0$ and $1 \wedge 1 = 1$.

If we continue to apply this transformation, we eventually obtain

$$f_{d_1 d_2 \ldots d_m}(.\, r_1 r_2 \ldots r_m) = (-1)^{r_m \wedge d_1 \oplus r_{m-1} \wedge d_1 \oplus d_2 \oplus \ldots \oplus r_2 \wedge d_{m-2} \oplus d_{m-1}} f_{d_{m-1} \oplus d_m}(.\, r)$$

where $r = r_1 r_2 \ldots r_m$.

Since $f_0 = 1$ and $f_1 = 1 \quad -1$

$$f_{d_{m-1} \oplus d_m}(.\, r_1 r_2 \ldots r_m) = (-1)^{r_1 \wedge d_{m-1} \oplus d_m}.$$

Hence,

$$f_{d_1 d_2 \ldots d_m}(.\, r_1 r_2 \ldots r_m) = (-1)^{r_m \wedge d_1 \oplus r_{m-1} \wedge d_1 \oplus d_2 \oplus \ldots \oplus r_1 \wedge d_{m-1} \oplus d_m}$$

and so

$$f_{d_1 d_2 \ldots d_m}(.\, r_1 r_2 \ldots r_m) \cdot f_{r_1 r_2 \ldots r_m}(.\, d_1 d_2 \ldots d_m) = 1$$

i.e., $f_{d_1 d_2 \ldots d_m}(.\, r_1 r_2 \ldots r_m) = f_{r_1 r_2 \ldots r_m}(.\, d_1 d_2 \ldots d_m)$

i.e., $W_n$ is symmetrical for all non-negative integers $n$.

The two orderings of the Walsh functions both give matrices which are symmetrical for all $n \geqslant 0$.

The interpretation of the Walsh functions as pattern products leads to a result of great computational importance – the Fast Walsh Transform – which is described below.

## FUNCTION APPROXIMATION

Methods of function approximation which employ algebraic polynomials are not ideally suited to computer use, because of the large number of multiplications which are involved in the computation. The same criticism applies to the use of trigonometric functions, and it is only by approximating a given function with a linear combination of orthogonal boolean functions, preferably complete in $L^2$, that all multiplications can be avoided in the calculations.

Suppose $g(x)$ is an integrable function on [0, 1). Then we can express $g$ as a linear combination of the Walsh functions, $f_0, f_1, \ldots, f_r, \ldots$

$$g(x) = \sum_{i=0}^{\infty} c_i \cdot f_i(x)$$

where $c_i$ is the $i$th Walsh coefficient, and is defined as the correlation between the given function, $g$, and the $i$th Walsh function, $f_i$:

$$c_i = \int_0^1 g(x) \cdot f_i(x)\, dx.$$

We can approximate a given function as closely as we please, since the partial sums,

$$s_N = \sum_{i=0}^{N} c_i \cdot f_i(x)$$

converge to the function, $g$, if $g$ is integrable square on [0, 1].

Also, the coefficients, $c_i$, tend to zero as $i$ tends to infinity, since

$$\sum_{i=0}^{\infty} c_i^2 = \int_0^1 g^2(x)\, dx.$$

Suppose $g(x)$ is integrable on [0, 1), and that we wish to approximate to $g$ by means of a linear combination of $2^n$ Walsh functions.

Then we first approximate to $g(x)$ by a step function $g_k$:

$$g_k = \frac{g_{\max} + g_{\min}}{2}, \quad k = 0, 1, \ldots, 2^n - 1$$

where $g_{\max}$ and $g_{\min}$ are the maximum and minimum values of $g(x)$ in the interval $\left[\frac{k}{2^n}, \frac{k+1}{2^n}\right)$.

Then we calculate the $2^n$ Walsh coefficients, $c_0, c_1, \ldots, c_{2^n-1}$, so that

$$g_k = \sum_{i=0}^{2^n-1} c_i \cdot f_i(x)$$

and $$c_i=\frac{1}{2^n}\sum_{k=0}^{2^n-1} g_k \cdot f_i(x), \quad \frac{k}{2^n}\leqslant x<\frac{k+1}{2^n}.$$

Since $f_i(x)$ always takes the value $+1$ or $-1$, for all $i$, the calculation of the Walsh coefficients only involves the addition or subtraction of the appropriate $g_k$; apart from the multiplication by the normalizing factor, $1/2^n$, which is not really a multiplication, but a shift, in any case.

## THE FAST WALSH TRANSFORM

Suppose we wish to evaluate the coefficient corresponding to the Walsh function whose pattern product representation is, say, $\bar{1} \cdot \bar{0} \cdot \bar{1}$. The coefficient required is

$$\tfrac{1}{8}(g_0-g_1+g_2-g_3-g_4+g_5-g_6+g_7)$$

where the $g_i$ are the function values, and the pluses and minuses correspond to the $+1$s and $-1$s of the pattern $\bar{1} \cdot \bar{0} \cdot \bar{1}$.

Now $$g_0-g_1+g_2-g_3-g_4+g_5-g_6+g_7$$
$$=(g_0-g_1)+(g_2-g_3)-(g_4-g_5)+(g_6-g_7)$$

which is a difference of sums of differences of $g_0, \ldots, g_7$, which we write as DSD – a D denoting a difference and an S a sum.

Note that there is a direct relationship between the Ds and the $\bar{1}$s of the pattern product and between the Ss and the $\bar{0}$s.

Hence we may compute all the $2^n$ coefficients as shown in the table below.

| For $n=3$ | Calculation of coefficients | | |
|---|---|---|---|
| Original data | Stage 1 | Stage 2 | Stage 3 |
| $g_0$ | $a_0=g_0+g_1$ | $b_0=a_0+a_2$ | $c_0=b_0+b_4$ |
| $g_1$ | $a_1=g_0-g_1$ | $b_1=a_1+a_3$ | $c_1=b_1+b_5$ |
| $g_2$ | $a_2=g_2+g_3$ | $b_2=a_0-a_2$ | $c_2=b_2+b_6$ |
| $g_3$ | $a_3=g_2-g_3$ | $b_3=a_1-a_3$ | $c_3=b_3+b_7$ |
| $g_4$ | $a_4=g_4+g_5$ | $b_4=a_4+a_6$ | $c_4=b_0-b_4$ |
| $g_5$ | $a_5=g_4-g_5$ | $b_5=a_5+a_7$ | $c_5=b_1-b_5$ |
| $g_6$ | $a_6=g_6+g_7$ | $b_6=a_4-a_6$ | $c_6=b_2-b_6$ |
| $g_7$ | $a_7=g_6-g_1$ | $b_7=a_5-a_7$ | $c_7=b_3-b_7$ |

Consider the fourth row: we first form $a_3$, the difference between $g_2$ and $g_3$; then the difference $a_1-a_3$, which is a difference of differences (since $a_1$ is the difference $g_0-g_1$); and finally the sum $b_3+b_7$, which is therefore the sum of differences of differences, since $b_3$ and $b_7$ are such.

Thus $c_3/2^n$ is a SDD and therefore is the value of the coefficient corresponding to the Walsh function $\bar{0} \cdot \bar{1} \cdot \bar{1}$ i.e., $f_4$ in the Walsh ordering.

To compute $2^n$ coefficients only $n \cdot 2^n$ additions and subtractions are required.

*Example*

| | | | |
|---|---|---|---|
| $g_0=0.2$ | $a_0=0.6$ | $b_0=2.0$ | $c_0=4.0$ |
| $g_1=0.4$ | $a_1=-0.2$ | $b_1=-0.4$ | $c_1=0.8$ |
| $g_2=0.6$ | $a_2=1.4$ | $b_2=-0.8$ | $c_2=-1.2$ |
| $g_3=0.8$ | $a_3=-0.2$ | $b_3=0$ | $c_3=0$ |
| $g_4=0.7$ | $a_4=0.8$ | $b_4=2.0$ | $c_4=0$ |
| $g_5=0.1$ | $a_5=0.6$ | $b_5=1.2$ | $c_5=-1.6$ |
| $g_6=0.9$ | $a_6=1.2$ | $b_6=-0.4$ | $c_6=-0.4$ |
| $g_7=0.3$ | $a_7=0.6$ | $b_7=0$ | $c_7=0$ |

and hence the coefficients in the Walsh ordering are

$$0.5 \quad 0 \quad -0.4 \quad -1.2 \quad 0 \quad 0 \quad -1.6 \quad 0.8$$

This method of computing the Walsh coefficients is closely analogous to the algorithm known as the Fast Fourier Transform (FFT), and for this reason is called the Fast Walsh Transform (FWT). It has elsewhere been called the Fast Walsh Fourier Transform and the Fast Hadamard Fourier Transform.

The FWT can be modified, as can the FFT, so that it becomes equivalent to the repeated execution of a much simpler algorithm (Pease 1968). The simplified algorithm is defined below, and each execution of it is equivalent to one stage of the FWT. It must be carried out $n$ times, therefore, to produce $2^n$ coefficients. (It is perhaps worth mentioning here that if it is executed $2n$ times, the coefficients produced after $n$ executions will be transformed back into the original data values; i.e., the FWT and its modified version are their own inverses.) Now that each stage of the modified FWT is identical, it is possible to think realistically (from an economic point of view) about constructing a special purpose machine to carry out each stage of the algorithm in parallel.

If the original data is $g_0, g_1, \ldots, g_1$, as above, then a single stage of the modified FWT is defined by:

| | | |
|---|---|---|
| $g'_0=g_0+g_1$ | | $g_0=g'_0$ |
| $g'_1=g_0-g_1$ | | $g_1=g'_2$ |
| $g'_2=g_2+g_3$ | | $g_2=g'_4$ |
| $g'_3=g_2-g_3$ | followed by | $g_3=g'_6$ |
| $g'_4=g_4+g_5$ | | $g_4=g'_1$ |
| $g'_5=g_4-g_5$ | | $g_5=g'_3$ |
| $g'_6=g_6+g_7$ | | $g_6=g'_5$ |
| $g'_7=g_6-g_7$ | | $g_7=g'_7$ |

or

$$
\begin{aligned}
g'_0 &= g_0 + g_1 \\
g'_1 &= g_2 + g_3 \\
g'_2 &= g_4 + g_5 \\
g'_3 &= g_6 + g_7 \quad \text{followed by} \quad g_i = g'_i, \quad \text{all } i. \\
g'_4 &= g_0 - g_1 \\
g'_5 &= g_2 - g_3 \\
g'_6 &= g_4 - g_5 \\
g'_7 &= g_6 - g_7.
\end{aligned}
$$

Although the FWT can be extended to $n$ dimensions, there is in fact no need to do so. An $n$-dimensional Walsh function is the product of $n$ one-dimensional Walsh functions; but such a product also defines another one-dimensional Walsh function in the sense of Definition 3. Hence, to each $n$-dimensional Walsh function there corresponds a one-dimensional Walsh function (the reverse is not true); and the computation of an $n$-dimensional coefficient (i.e., a coefficient corresponding to an $n$-dimensional function) can be executed by means of the (one-dimensional) FWT.

The coefficients produced by the FWT are in the pattern product ordering, and they must be permuted so as to be in the Walsh ordering, for reasons which are made clear below. The permutation is effected by means of a simple algorithm (see Appendix 2).

The algorithms of the FWT and the modified FWT are also in Appendix 2.

## SHAPES

A shape is defined as a bounded region which is the interior of a simple closed curve.

In a cellular array, each component of which takes the value 1 or 0 (black or white), a retinal shape is a set of connected cells of value 1.

A square cell has as neighbours the four cells which have common edges with it. A boundary corner is a corner of a cell of value 1, which is also a corner of a cell of value 0. A boundary cell has value 1 and at least two boundary corners.

The retinal function is the set of 1*s* and 0*s*. We now consider a series of transformations upon such a function.

The coordinates of the boundary cells can be found by means of an edge-following algorithm (*see* Appendix 2) (Ledley 1965), or by a straightforward parallel computation which produces for a cell having value $C$, and neighbours having values $C_1$, $C_2$, $C_3$ and $C_4$, the value of the function

**if** $C=1$ **and** $C_1+C_2+C_3+C_4<4$ **then** 1 **else** 0 **close.**

The form of this function follows directly from the alternative definition of a boundary cell as one which has value 1 and at least one neighbour having value 0. In the case where all four neighbours have value 0 the central cell is a complete retinal shape in itself.

The centre of gravity of the two-dimensional lamina represented by the retinal function is easily found as the point whose coordinates are the averages of the coordinates of the cells forming the shape. The distances between the centre of gravity and the boundary corners is now calculated. Note that the boundary corners are equidistant along the perimeter of the retinal shape.

Let the distances from the centre of gravity to the boundary corners be $g_1, g_2, \ldots, g_m$. These are first approximated to by a set of values $g'_1, g'_2, \ldots, g'_N$, where $N$ is of the form $2^n$, and where

$$\frac{1}{N}\sum_{k=1}^{N} g'_k = \text{constant}.$$

The FWT can now be applied to this new set of function values to produce $N$ coefficients.

Since the point from which the measurements were taken (the centre of gravity) is relative to the retinal shape, the function values are translation independent, and so therefore are the resulting Walsh coefficients. Also, because the average value of $g'_k$ is constant, the transformation from the retinal shape to the coefficients is size independent.

However, the coefficients are rotation dependent, and so they do not prove a unique specification of the retinal shape. This problem can be partially overcome by using other sets of orthogonal boolean functions which are 'cyclical'.

For example:

$$\begin{aligned} f_0 &= \ 1 \quad\ 1 \quad -1 \quad\ 1 \\ f_1 &= \ 1 \quad\ 1 \quad\ 1 \quad -1 \\ f_2 &= -1 \quad\ 1 \quad\ 1 \quad\ 1 \\ f_3 &= \ 1 \quad -1 \quad\ 1 \quad\ 1 \end{aligned}$$

The typical function in such a set is a digital sequence with pseudonoise properties. Now the set of coefficients is unique. If a standard rotational position for a retinal shape is somehow fixed which gives coefficients

$$c_0 \quad c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5 \quad c_6 \quad c_7$$

then in general the same shape will produce coefficients

$$c_k \quad c_{k+1} \ldots c_7 \quad c_0 \quad c_1 \ldots c_{k-1}$$

It appears that such cyclical orthogonal boolean functions can be formed in complete sets of order $4m$ (Paley 1933), whereas Walsh functions can only be formed in complete sets of order $2^n$. However, Walsh functions have the important interpretative advantage of their resolution properties (below). Also, the Walsh functions are the only infinite, complete set of boolean orthogonal functions.

The transformation of the two-dimensional retinal shape into a one-dimensional set of function values (the distances between the centre of gravity and the boundary corners) has certain drawbacks, foremost of which is the

inability to deal with disconnected retinal 'shapes', in particular shapes with 'holes' in them. This problem could be overcome by standardizing the two-dimensional function with respect to translation, size, and rotation, and then applying the Walsh transform to the resulting retinal function of 1*s* and 0*s*. The computation involved is much greater than in the one-dimensional case, and we examine below how we can develop methods to deal with shapes with 'holes' on a one-dimensional basis; and in the process evolve a technique for processing retinal functions which represent a complex set of shapes.

There is another disadvantage of the centre-of-gravity/boundary corner method, which is that a certain amount of information is lost in the process of transforming the retinal function to the one-dimensional form. However, the information which is lost is of no importance for the purposes of spatial spectra analysis.

To reduce errors as much as possible, it is desirable that the number of boundary corners be close to a power of 2.

## RESOLUTION GRAPHS

The $k$th Walsh function in the Walsh ordering has exactly $k$ discontinuities on [0, 1); and the $k$th Walsh coefficient is therefore a measure of the correlation between a given function and a standard function with $k$ discontinuities. We can construct a spatial spectrum of a shape (a resolution graph) simply by plotting $c_m^2$ against $1/m$, for $m=1, 2, \ldots, 2^n-1$.

The Walsh coefficients can be used to determine whether the differences between shapes are due to, say, large-scale or small-scale features, and the resolution graph enables one to interpret the coefficients for such purposes at a glance.

Some shapes and their resolution graphs are shown in figure 1.

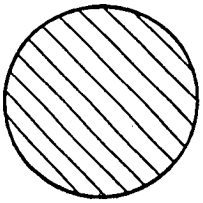
If the given function is produced from a shape in the manner described above, then a slight modification must be made in the construction of the resolution graph. In the Walsh functions, $f_{2m-1}$, $m=1, 2, \ldots$, there is an extra discontinuity between the initial and final values of the function. The spatial spectrum is formed by plotting

$$\frac{c_{2m}^2+c_{2m-1}^2}{2} \text{ against } \frac{1}{2m} \text{ for } m=1, 2, \ldots, \frac{N-2}{2}.$$

## PATTERNS OF SHAPES

The resolution graph of the retinal function shown in figure 2 would be extremely difficult to interpret, except to an experienced person, or where only a small amount of information was to be extracted. It is therefore desirable to consider figure 2 as a pattern of shapes; that is, as a black shape with two white shapes superimposed upon it. A tree which describes the topological structure of such a figure can be produced by Buneman's methods (*see* p. 383), although Buneman would regard the white areas as holes in the black shape.

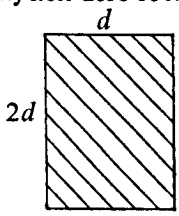
(a)

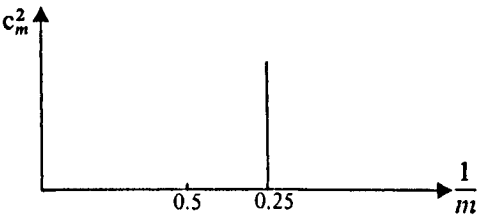





The only non-zero coefficient is $c_0$, which does not appear in the resolution graph.

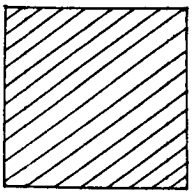
(b)

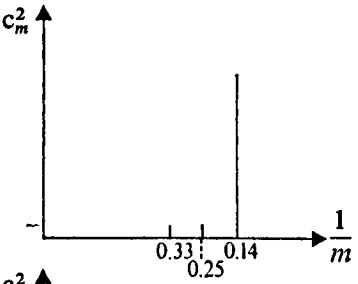


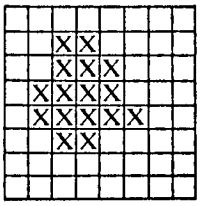

The height of the peak at 0.25 depends upon the size of the rectangle.

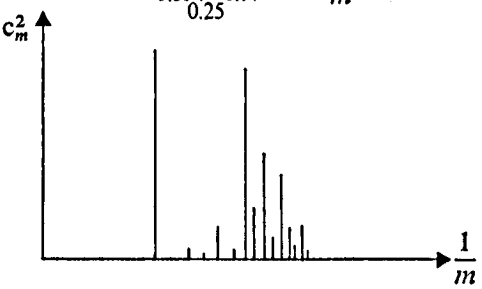
(c)



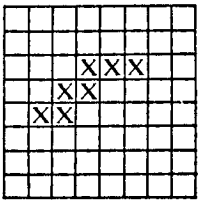



(d)






(e)



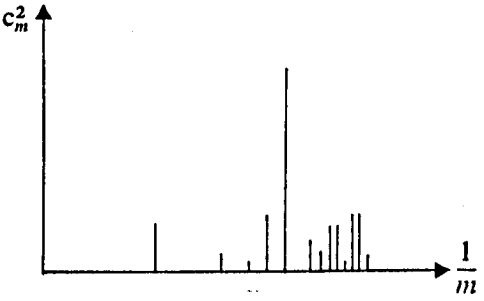



Figure 1

There are several methods, including one proposed by Buneman, for producing a set of retinal functions, each of which represents a shape, from a retinal function which represents a pattern of shapes. Unfortunately, this task of decomposition cannot be carried out by means of a parallel algorithm (Minsky and Papert 1969).

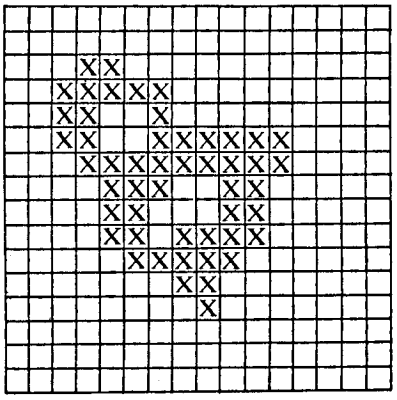


Figure 2. The x*s* denote black squares

The specification of a retinal function such as figure 2 consists, therefore, of a terminal string, produced by Buneman's grammar, with a set of Walsh coefficients attached to each node of the corresponding tree, i.e., as a combination of the sets of Walsh coefficients.

**Acknowledgements**

This work was supported by a grant from the Science Research Council. I am extremely grateful to many people, especially Dr B. Meltzer, for their suggestions and interest.

## REFERENCES

Lechner, R. (1961) A transform theory for functions of binary variables. Theory of switching. Harvard Computation Lab., Cambridge, Mass. Progress Report No. BL–30, Section X, 1–37.

Ledley, R.S. (1965) *Use of computers in biology and medicine*, p. 340. New York: McGraw-Hill.

Minsky, M. & Papert, S. (1969) *Perceptrons*, p. 74. Cambridge, Mass.: MIT Press.

Paley, R.E.A.C. (1933) On orthogonal matrices. *J. Math. Phys.*, **12**, 311–20.

Pease, M.C. (1968) An adaptation of the fast fourier transform for parallel processing. *J. Ass. comput. Mach.*, **15**, 252–64.

Polyak, B.T. & Shreider, Yu.A. (1966) The application of Walsh functions in approximate calculations. *Problems in theory of mathematical machines: collection II.* pp. 174–90. Moscow: Fizmatgiz.

Walsh, J.L. (1923) A closed set of orthogonal functions. *Amer. J. Math.*, **55**, 5–24.

## APPENDIX 1

The original retinal function is binary-valued and the main reason for not working with the data in this form is the increased complexity of the two-dimensional calculations which would be involved. Even in one dimension the most convenient form in which to represent the outline of a retinal shape would be by a pair of ternary-valued functions, one recording whether in moving from one boundary cell to the next the $x$-coordinate increases, decreases, or stays unaltered, the other giving the corresponding information about the $y$-coordinates.

It would appear to be useful, therefore, if special methods could be found for dealing with functions which take only a finite number of different values, and in particular binary and ternary functions. There is a reasonable hope that this might be achieved.

## APPENDIX 2

### 1. Permutation of $2^n$ Walsh coefficients from pattern product ordering to Walsh ordering.

```
function conv n n; vars i k;
logand(logbits(j), 1)→k;
for i, step (1, 1, n−1) do
if logand(logbits(logand(j, logshift(n, 1−i))), 1)=0 then goto back close;
k+logshift(1, i)→k;
back: repeat;
k
end;
```

note: *logbits*($j$) is a function whose value is the number of bits which are 1 in the binary expansion of $j$.

### 2. Fast Walsh Transform

```
function fwt n; vars i j k l p q w;
0→k;
back logshift:(1, k)→1;
for i, step (0, 21, 2**n−21) with j, step (0, 1, l−1) do
w(i+j)→p;
w(i+j+1)→q;
p+q→w(i+j);
p−q→w(i+j+1);
repeat;
k+1→k;
if not(k=n) then goto back close;
end;
```

### 3. Modified Fast Walsh Transform

```
function walsh n; vars i j h g a k m;
0→i;
2**n→m;
```

```
back: for j, step(0, 2, (m−1)/2) do
logshift(j, −1)→k;
g(k)→h(j)+h(j+1);
g(k+m/2)→h(j)−h(j+1);
repeat;
h→j;
g→h;
j→g;
i+1−i;
if not(i=n) then goto back close;
end;
```

#### 4. Edge follower

```
function follow; vars startx starty k a c;
read( startx starty);
0→k;
startx→i;
starty→j;
1: check;
move(i, j−1, 2); move(i−1, j 4); move(i, j+1, 7); goto 8;
10: check;
move(i+1, j, 8); move(i, j−1, 2); move(i−1, j, 4); goto 7;
100: check;
move(i−1, j, 4); move (i, j+1, 7); move(i+1, j, 8); goto 2;
200: check;
move(i, j+1, 7); move(i+1, j, 8); move(i, j−1, 2); goto 4;
2: if not(c(i+1, j−1)=1) then goto 3; i+1→i; 3: j−1→j; goto 10;
4: if not(c(i−1, j−1)=1) then goto 6; j−1→j; 6: i−1→i; goto 1;
7: if not(c(i−1, j+1)=1) then goto 9; i−1→i; 9: j+1→j; goto 100;
8: if not(c(i+1, j+1)=1) then goto 11; j+1→j; 11: i+1→i; goto 200;
30: stop;
end;
function check; vars i j c startx starty k;
if i=startx and j=starty and not (k=2) then goto 30 close
end;
function move i j x;
if c(i, j)=1 then goto x close
end;
```

The above functions have been very freely translated from the original language.

# 23

# Conic Sections in Automatic Chromosome Analysis

Keith A. Paton
Medical Research Council Computer Unit
London

## INTRODUCTION

This paper describes a technique for use in automatic chromosome analysis. As a preliminary it may be useful to answer four questions, namely. what are chromosomes, how can we see them, how do we count them, and why should we count them?

Chromosomes are thread-like objects, mainly of nucleic acids, which occur in the nuclei of cells and carry genetic material. The nucleus of a normal human cell is now known to contain 46 chromosomes, but this was not established until Tjio and Levan (1956) introduced a method which allowed the microscopic examination of individual chromosomes. This technique enables the cytogeneticist to examine cells in the form shown in figure 1,

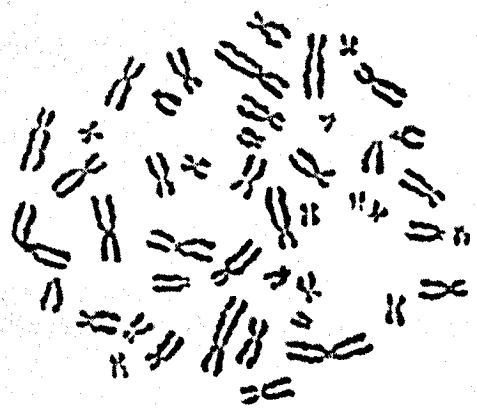


Figure 1. A metaphase spread

where the individual chromosomes are condensed to a few microns long and, in general, occupy distinct regions of the slide. Within each chromosome may be recognized a lightly stained constriction called the centromere and the London report (1963) suggests that relative size and centromere position may be used to identify ten recognizably distinct groups of normal chromosomes. The chromosome complement of an individual cell may then be described by giving the number of chromosomes in each of the ten groups together with a suitable description of any abnormal chromosomes. Traditionally this description is presented as shown in figure 2 where the chromosomes have been arranged in the various groups.

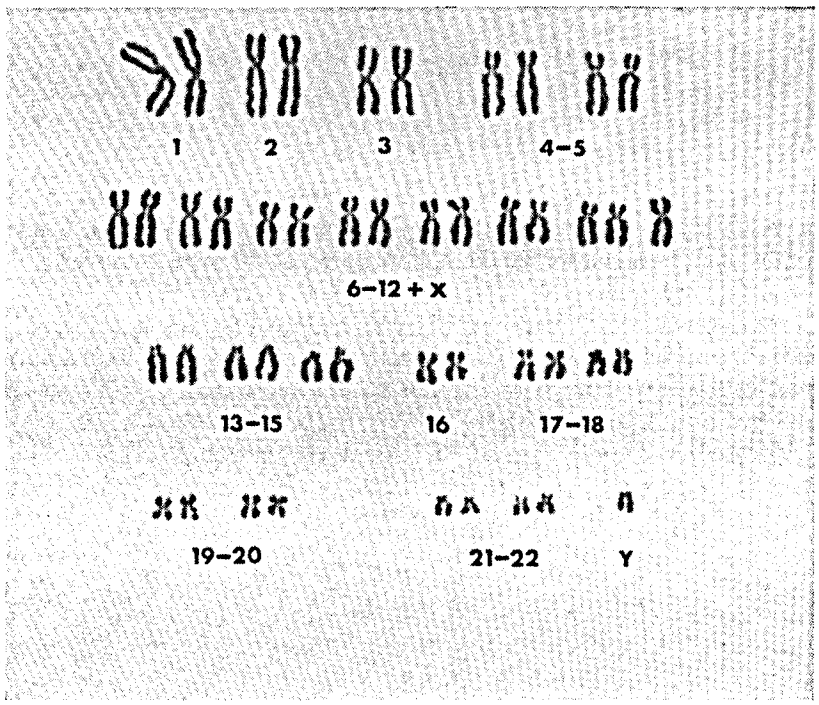



Figure 2. The karyotype of figure 1

Such a description is employed in population cytogenetics where one aim is to establish the frequency of chromosome abnormalities in the general population (Court Brown 1967). A slightly different form of description is employed in radiation cytogenetics where the effect of ionizing radiation may be conveniently measured as the yield of those easily recognizable aberrations known as dicentric and ring chromosomes (Evans 1962). When the dose effect relationship has been discovered the aberration yield may be used as a biological indicator of dose.

Since these and other applications demand the analysis of very many cells there are being developed systems of automatic chromosome analysis

(Butler *et al.* 1968, Ledley *et al.* 1965, Mendelsohn *et al.* 1969; Neurath *et al.* 1969; Rutovitz *et al.* 1969).

From the standpoint of pictorial pattern recognition one outstanding problem is that of centromere detection. As Rutovitz (1969) has pointed out no one method seems likely to work in all cases, for individual chromosomes appear in many different shapes. Thus a comprehensive system for automatic chromosome analysis might well be equipped with several different centromere-finding techniques, each one appropriate to a particular chromosome configuration. Under such a system the first step in the analysis of a chromosome might well be the recognition of its shape in order to ensure that the most appropriate of these techniques was used to find the centromere. In this paper we discuss a method of describing the shape of a chromosome (or anything else!) in terms of its best conic section and illustrate this method by distinguishing between some of the various chromosome shapes shown in figure 3 as idealized line drawings and in figure 4 as they really occur.

## CHROMOSOME SHAPES

As we have said the methods used to find the centromere will depend very much on the shape of the chromosome. Thus Rutovitz (1967) has shown that when the chromosome has bilateral symmetry, as is the case in 3.1 to 3.3, the axis of symmetry may usually be found by the method of principal axes, and that the best defined trough on the profile obtained by 'projecting' the chromosome on this axis is often sufficient to define the position of the centromere. In the case of 3.4 and 3.5 boundary tracing techniques such as those developed by Ledley *et al.* (1966) Gallus (1968) and Rutovitz (1970) are probably useful. In the more difficult cases of 3.6 and 3.7 Hilditch (1969) has used an elegant 'stripping' technique to reduce the chromosome to a 'skeleton', i.e., a line drawing which preserves the connectivity and general shape of the original. Subsequent analysis can then be expressed in graphical terms, the skeleton being a graph whose nodes indicate important points of the chromosome, such as tips of arms. As might be expected, special measures are required in the case of 3.7, where the skeleton contains a cycle, since the chromosome is doubly connected. Finally we show two cases of two chromosomes appearing as one; in 3.8 the two can be separated by ignoring points of low density in the object, whereas in 3.9 a 'disentangling' technique such as that introduced by Hilditch (1969a) would be necessary.

Since, in general, a simple technique will be faster than a more complicated one it is prudent strategy to try the simple technique first and to resort to the complicated one only when the simple one has failed. When, as in our system (Rutovitz 1969), only two or three techniques are involved, this strategy is satisfactory. However if the system is extended to, say, four or five different complicated techniques it becomes important to select at the beginning the technique which has the best chance of success; otherwise time is lost in applying complicated techniques which fail because they are inappropriate.

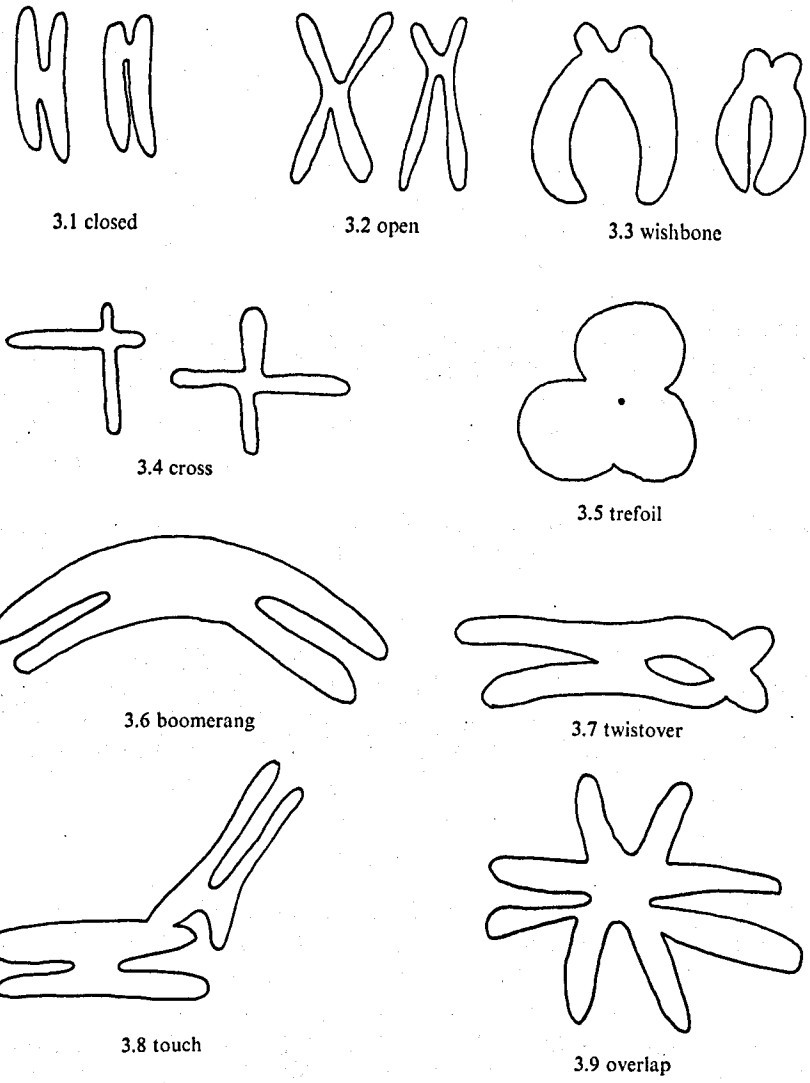



Figure 3. Some idealized chromosome configurations

The examples quoted above suggest that the selection may well be made primarily on the basis of shape. In short, we wish to distinguish between the shapes of 3.1 to 3.9 not because we are interested in the shapes themselves but because by taking account of the shape we hope to find the centromere more efficiently.

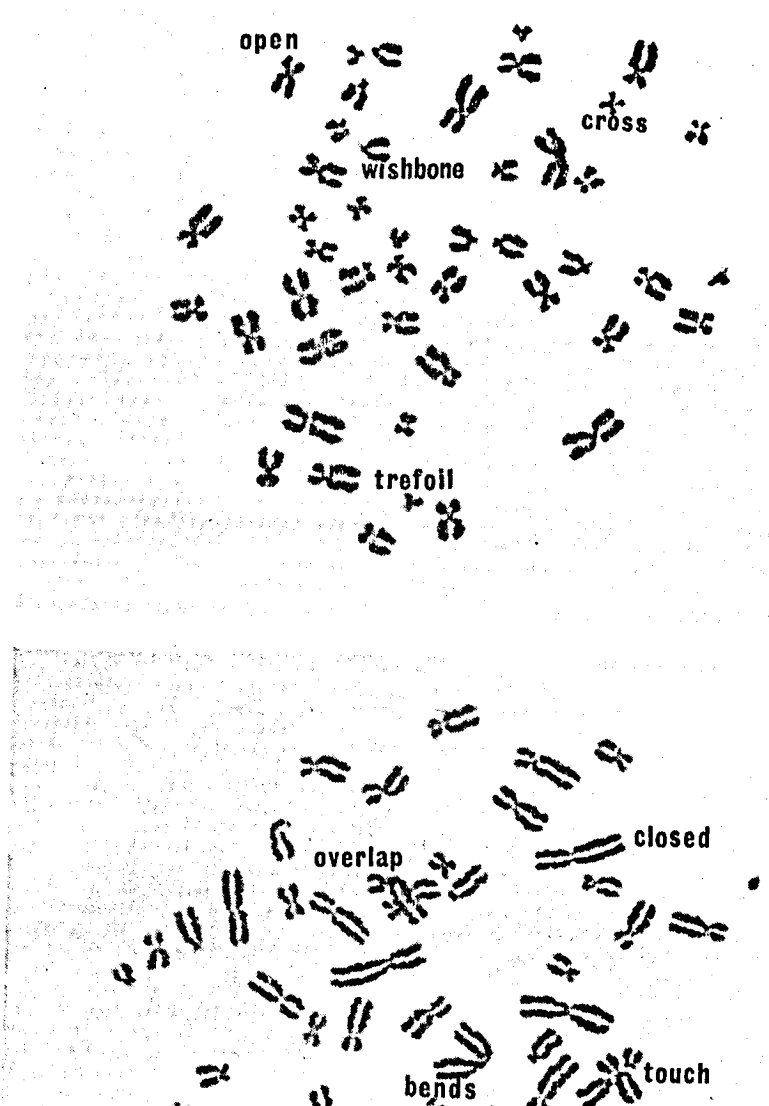



Figure 4. Some actual chromosome configurations

## CHROMOSOME PATTERNS

The patterns in which we are interested are obtained as follows. Cells in metaphase are prepared by the method of Moorhead (1960), as modified by Jacobs *et al.* (1964), recorded at high magnification on 35 mm. film and read into the computer store via FIDAC (Ledley 1965). The scene on the slide is thus represented as an integral function on a subset, $Q$, of the integral plane. The labelling algorithm (Hilditch 1969b) is then invoked (with an appropriately chosen threshold) in order to express the 'significant' portion of the set $Q$ as the sum of several components, each one a maximal connected set of points at which the density exceeds the threshold. Each component represents a single area of interest on the slide which we shall call an object; in favourable conditions objects and chromosomes are in 1–1 correspondence. For the sake of discussion we shall define the pattern derived from an object as the function $p$ for which $p(x, y)$ is the absorbancy at point $(x, y)$ of the slide if this point belongs to the chosen object and zero elsewhere.

Since we are interested only in configuration we shall wish to discard those properties of the pattern which are dependent on the coordinate system. Thus we shall say that pattern $p$ is a transformation of pattern $q$ if there exist $\lambda>0$, $\delta=\pm 1$, $A>0$, $u_0$, $v_0$, $\theta$ such that $p(x, y)=A\,q(u, v)$ whenever

$$\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}\delta\lambda & 0\\0 & \lambda\end{pmatrix}\begin{pmatrix}\cos\theta & \sin\theta\\-\sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}u-u_0\\v-v_0\end{pmatrix}. \tag{1}$$

Here $A$, $(u_0, v_0)$, $\theta$, $\lambda$, $\delta$ define a change in intensity, origin, orientation, size, and sense respectively. Since 'is a transformation of' is an equivalence relation on the set, $P$, of all patterns a natural simplification is to seek a suitable canonical set, namely a subset, $C$, of $P$ such that any pattern in $P$ is equivalent to exactly one pattern in $C$. Then any property of a pattern $p$ established by operating on its correspondent in $C$ is independent of the coordinate system. This approach has been used in character recognition by Alt (1962), Guiliano *et al.* (1961) and Hu (1962) though in their case a different group of transformations is involved since orientation cannot be ignored.

An attempt is made in the appendix to derive a canonical set in terms of moments. We shall adopt the convention that the $(i, j)$th moment of a pattern denoted by a small letter is that capital letter with suffix $i, j$. Thus $P_{ij}=\iint x^i y^j p(x, y)\,dx\,dy$. The set, $C_2$, of patterns for which $P_{00}=1$; $P_{10}=P_{01}=0$; $P_{20}+P_{02}=1$; $P_{11}=0, P_{20}<P_{02}, P_{21}>0, P_{12}>0$ would be a canonical set for transformations of the form (1) were it not for the existence of some exceptional patterns such as those for which $P_{10}=P_{01}=0$; $P_{11}=0$ and $P_{20}=P_{02}$. Such patterns can have no unique correspondent in $C_2$.

Since our description of the pattern in terms of its best conic section will involve a return to the original coordinates we shall find it most convenient to retain the orientation and sense of the original axes and consider only transformations of the form $p(x, y)=A\,q(u, v)$ whenever

$$\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}\lambda & 0\\0 & \lambda\end{pmatrix}\begin{pmatrix}u-u_0\\v-v_0\end{pmatrix}.$$

For these transformations the set, $C_1$, of patterns for which $P_{00}=1$; $P_{10}=P_{01}=0$; $P_{20}+P_{02}=1$ is canonical. The correspondent in $C_1$ of a given $p$ will be called the standard pattern of $p$.

## THE CONIC SECTION APPROXIMATION

The method we shall use for pattern description is an extension of the principal axis technique introduced to chromosome analysis by Rutovitz (1967). Here the pattern is approximated by a line through the pattern centroid about which the second moment is a minimum. The minimum value is given by the smaller eigenvalue, $\lambda_2$, of the matrix of standard moments $\begin{pmatrix} P_{20} & P_{11} \\ P_{11} & P_{02} \end{pmatrix}$ and, as Rutovitz points out, the smaller the value of $\lambda_2$ the better is the approximation to the pattern. In fact $\lambda_2$ provides a scale-free measure of the appropriateness of the principal axis description of the pattern, taking values near 0 for long thin patterns and values near $\frac{1}{2}$ for patterns such as 3.3 and 3.4 for which the principal axis is but poorly defined. Indeed patterns such as these seem more akin to ellipses and pairs of perpendicular straight lines. We are thus led to the notion of a best conic section approximation for a pattern.

In a given frame of reference any conic section may be uniquely represented as $Q(x, y)=ax^2+2hxy+by^2+2gx+2fy+c=0$ provided that $a^2+4h^2+b^2+4g^2+4f^2+c^2=1$ and that the first non-zero element of the vector $(a, 2h, b, 2g, 2f, c)'$ – which we shall refer to as a conic-vector – is positive. Let us define* the 'distance' from the point $(u, v)$ to the conic $Q(x, y)=0$ as $|Q(u, v)|$. Then we may take account of the density of the pattern by defining the total weighted squared distance from the pattern to the conic as $D=\iint p(u, v)|Q(u, v)|^2\,du\,dv$ and a best conic as one for which $D$ is least. In terms of moments, $D$ may be rewritten as $s'Bs$, where $B$ denotes the real symmetric matrix

$$\begin{pmatrix} P_{40} & P_{31} & P_{22} & P_{30} & P_{21} & P_{20} \\ P_{31} & P_{22} & P_{13} & P_{21} & P_{12} & P_{10} \\ P_{22} & P_{13} & P_{04} & P_{12} & P_{03} & P_{02} \\ P_{30} & P_{21} & P_{12} & P_{20} & P_{11} & P_{10} \\ P_{21} & P_{12} & P_{03} & P_{11} & P_{02} & P_{01} \\ P_{20} & P_{11} & P_{02} & P_{10} & P_{01} & P_{00} \end{pmatrix}$$

and $s$ the conic vector $(a, 2h, b, 2g, 2f, c)'$. The matrix $B$ is, of course, defined by and, we hope, characteristic of the pattern $p$.

To minimize $D$ by choice of conic is to minimize $s'Bs$ subject to $s's=1$, a well-known problem. Since $D$ is non-negative the matrix $B$ is non-negative definite, whence the eigenvalues of $B$ may be written $\lambda_1\geqslant\lambda_2\ldots\geqslant\lambda_6\geqslant 0$. There is an orthonormal basis, say $x_1, x_2, \ldots x_6$, such that $Bx_i=\lambda_i x_i$ for

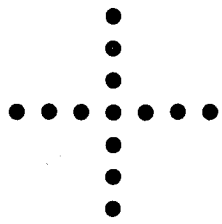
* Use of the Euclidean distance leads to the attempt to minimize $\iint p(u, v)|E(u, v)|^2\,du\,dv$, where $E(u, v)$ is itself the result of a minimization.

$i=1, 2, \ldots 6$, and if $\lambda_5 > \lambda_6$ then $s'Bs$ takes its minimum value of $\lambda_6$ when $s=x_6$. If $\lambda_5=\lambda_6$ the situation is more complicated. It is shown in Halmos (1958) that if the smallest eigenvalue has multiplicity $m+1$, then there are $m+1$ linearly independent vectors $x_6, x_5, \ldots, x_{6-m}$ such that any vector $s=\sum_{i=6-m}^{6} t_i x_i$ satisfying $\sum_{i=6-m}^{6} t_i^2=1$ satisfies $s'Bs=\lambda_6$.

The smaller the value of $\lambda_6$ the better is the fit of the best conic to the pattern. By taking the pattern in standard form we ensure that $\lambda_6$ does not depend on the original size of the pattern and is thus a scale-free measure of the success of the approximation. The best conic is well defined if $\lambda_6$ is much smaller than $\lambda_5$ and poorly defined if $\lambda_6$ and $\lambda_5$ are nearly equal. Thus an appropriate measure of the sharpness with which the best conic is defined is the quantity $(\lambda_5-\lambda_6)/(\lambda_5+\lambda_6)$, which takes value 0 for the extreme case of $\lambda_5=\lambda_6$, and increases with the relative separation of $\lambda_5$ and $\lambda_6$. We shall refer to $\lambda_6$ as 'goodness of fit' and $(\lambda_5-\lambda_6)/(\lambda_5+\lambda_6)$ as 'sharpness'.

*An example*

Let us, for simplicity, consider the cross-shaped pattern of thirteen dots of unit density.



Then $P_{40}=P_{04}=13/16$; $P_{20}=P_{02}=\frac{1}{2}$; $P_{00}=1$ and all other moments vanish. Inspection reveals that the second column of the matrix $B$ consists entirely of zeros whence $s'Bs$ attains its minimum value of zero when $s'$ takes the value $(0, 1, 0, 0, 0, 0)$. Thus the conic $xy=0$ is a best conic, as it should be. It may be verified that the other eigenvalues are roughly 26/16, 13/16, 1/2, 1/2, 3/16 and that the conic corresponding to $\lambda_5=3/16$ is $x^2-y^2=0$.

## DISCRIMINATION BETWEEN DIFFERENT CLASSES

Our first task is to use the best conic in an attempt to discriminate between objects of the different classes shown in figure 3. Before computation begins we obtain a representative sample by asking an independent observer to assign each chromosome in four cells to the most appropriate one of the nine classes. This provides sufficient examples of the common classes 3.1–3.5, to which we shall confine our attention. Since no selection is involved we reduce the risk that our sample contains only specially favourable examples of the classes.

The next step is to superimpose the best conic on the original pattern. The best conic is computed for the standard pattern and an appropriate linear transformation is used to obtain the best conic for the original pattern. In practice the pattern is defined as a function on the integral plane and to superimpose the best conic is to substitute the value '$C$' for the value of the function at the points which lie on the conic. We shall say that the point $(i, j)$ lies on the conic $Q(x, y)=0$ if $Q(u, v)=0$ for some $(u, v)$ in $[i-\frac{1}{2}, i+\frac{1}{2}) \times [j-\frac{1}{2}, j+\frac{1}{2})$. For a given pattern let $A$ be the set of points which lie on the best conic and $B$ the set for which $p(i, j)>0$. Then we define the conic skeleton, $C$, as the intersection of $A$ with $B$. Since points of greatest density are most significant in the determination of the best conic this definition of skeleton is relatively insensitive to noise which is usually of comparatively low density.

Figure 5 shows patterns from the various classes and their skeletons, not necessarily representative. Skeletons for classes 1 and 2 are both hyperbolae but with more or less parallel arms for class 1 and more curved arms for class 2. For class 3 there are two varieties of elliptical skeleton, the whole ellipse for chromosomes with their arms close together and the half-ellipse for chromosomes with their arms well spread. For class 4 the skeleton is an almost-rectangular hyperbola. There seems to be no characteristic skeleton for class 5.

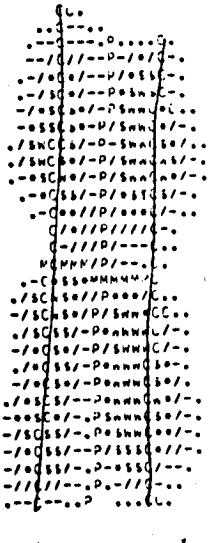
If the best conic approximation is to be used to discriminate between patterns it is necessary to obtain some form of quantitative description of the properties of the conic. One such description is provided by the classical reduction of the conic to principal axes. If the conic is neither a parabola nor a pair of straight lines then, as is well known, it is uniquely determined by the five parameters $x_0, y_0, \emptyset, A, B$ where $(x_0, y_0)$ is the centre, $\emptyset$ the inclination of the major axis to the $Y$-axis, and $|A|, |B|$ the lengths of the major, minor semi-axes respectively. Details of the reduction to principal axes and the rule of signs for $A, B$ are to be found in the appendix. Since we are most interested in the conic we shall choose to align the $X$-axis along the conic major axis with origin at the conic centre and to assign senses to the axes in such a way that the pattern centroid $(x_1, y_1)$ has positive coordinates. Our five basic variables are now $x_1, y_1, \emptyset, A, B$.

The exceptional cases of parabola and straight lines occur when

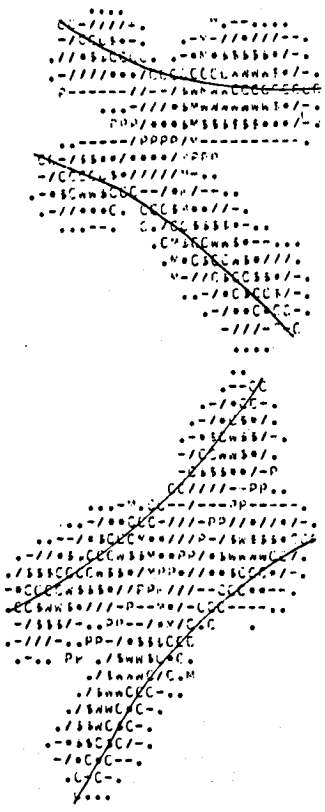
$$C=ab-h^2 \quad \text{or} \quad T=\begin{vmatrix} a & h & g \\ h & b & f \\ g & f & c \end{vmatrix}$$

vanish and since zero is rarely achieved in computation these cases occur but rarely in practice. Instead we obtain ellipses or hyperbola for which $C$ or $T$ is very small.

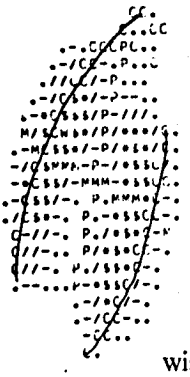
Other variables considered are 'size'$=AB$, 'slimness'$=A/B$, 'type'$=2AB/(A^2+B^2)$, 'angle between the asymptotes'$=\tan^{-1}(2AB/(A^2-B^2))$ 'generacy'$=T$, 'sharpness'$=(\lambda_5-\lambda_6)/(\lambda_5+\lambda_6)$, and 'goodness'$=\lambda_6$.



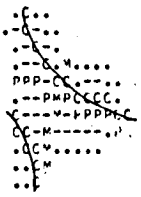
closed

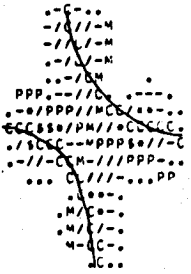


open

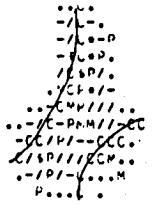


wishbone





cross



trefoil

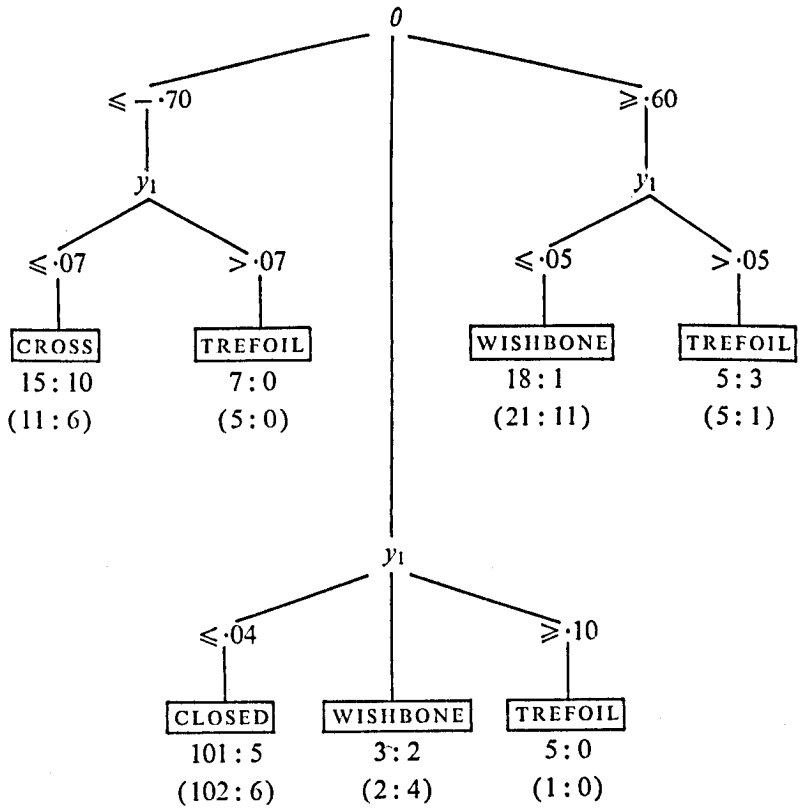
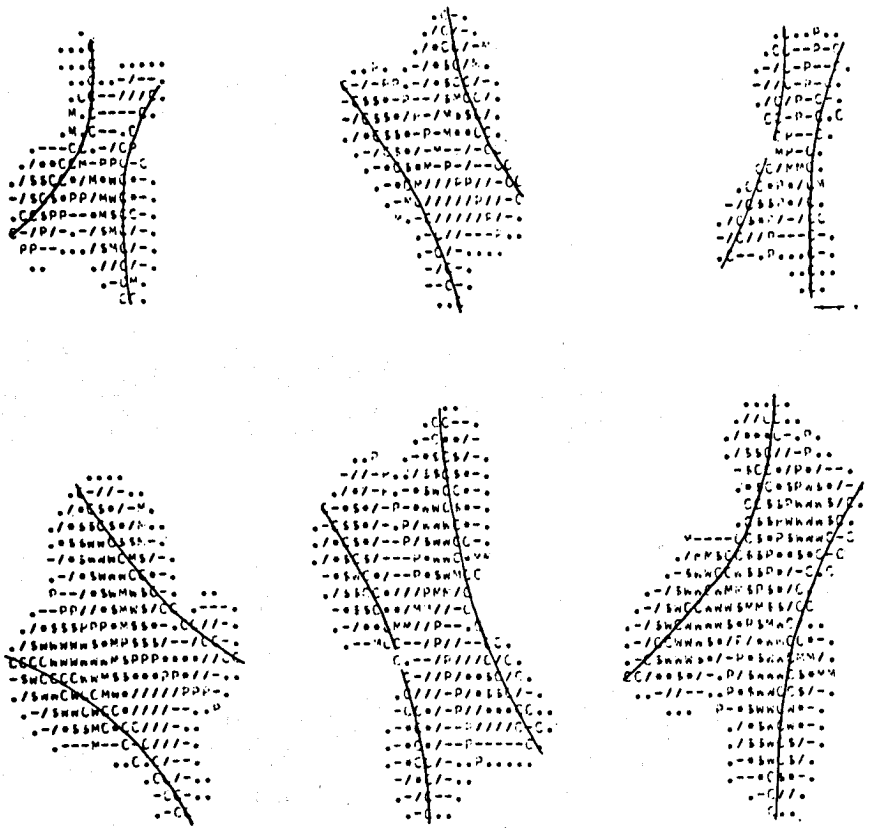


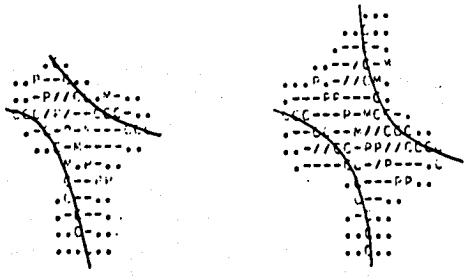

Figure 6. A possible decision tree. (Figures under the boxes denote success rates; thus 101 assignments to 'closed' were correct and 5 were wrong. The upper figures refer to the patterns with density, the lower figures to the patterns where density has been ignored.)

Of the twelve variables we have so far defined only seven are independent. This is because the first ten are all deduced from the conic vector $(a, 2h, b, 2g, 2f, c)'$, which contains but five independent variables. These ten may therefore be thought of as different expressions of five basic variables and our task is to find which, if any, of these expressions offer discrimination between the different pattern classes. At this point the distinction between classes 3.1 and 3.2 was dropped, partly because there are so many borderline cases, partly because the centromere may be found by the same method, that of principal axes, in both cases.

◀Figure 5. Some conic skeletons. (The symbols have been chosen in an attempt to reflect the density of the chromosome. '*P*' denotes the principal axis, '*M*' the minor axis, '*C*' the best conic)



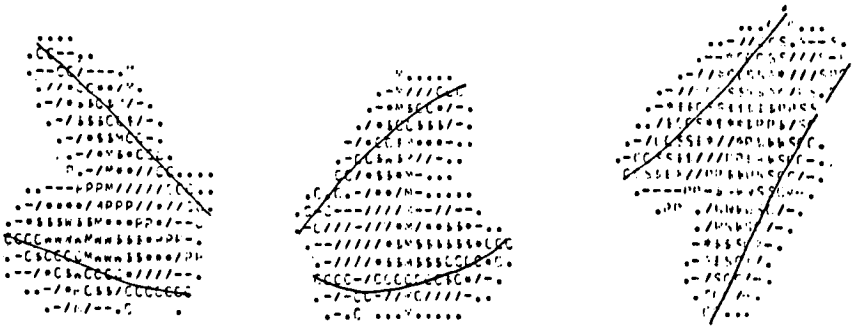
Six closed or open classed as cross



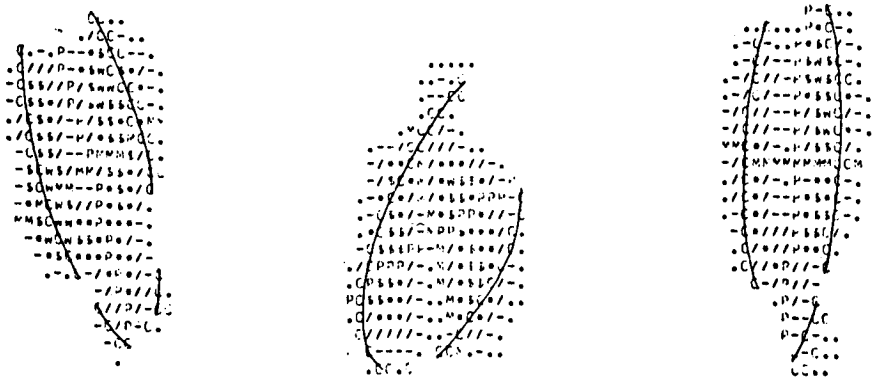
Two trefoil classed as cross

(a)

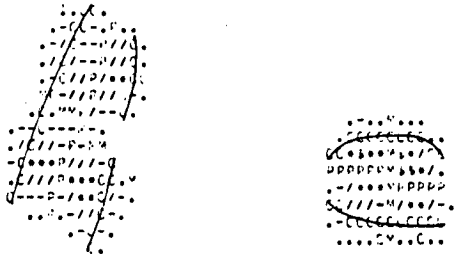
Figure 7. Some chromosome patterns misidentified



Three wishbone classed as closed



Three wishbone classed as trefoil



Two trefoil classed as closed, open

(b)

The skeletons in figure 5 suggest that pattern classes 3.1 to 3.4 may be characterized in terms of the variable $\theta = \tan^{-1}(2AB/(A^2-B^2))$, which may be interpreted, in the case of the hyperbola, as the angle between the asymptotes. It is found by experiment that trefoils (class 3.5) are split unequally by the major axis of their best conic, a property which may be conveniently expressed in terms of $y_1$, the distance from the pattern centroid to the conic major axis. This suggests the following ALGOL decision rule:

class = **if** $y_1 > \beta$ **then** trefoil **else**
**if** $\theta < \gamma$ **then** cross **else**
**if** $\theta < \delta$ **then** wishbone **else** closed;

where $\beta, \gamma, \delta$ are suitably chosen constants. In practice better results are obtained from a more complicated rule in which $\theta$ is used first and different constants are used according to the value of $\theta$. The rule is shown in the form of a decision tree in figure 6. Here the pair $a : b$ under a box denotes that of the patterns classed to that box $a$ were correctly and $b$ were wrongly classified. 154 correct and 21 wrong decisions were made.

For the sake of interest we show in figure 7 most of the 21 cases where the verdict of the computer did not agree with that of the independent observer. It is tempting to explain the errors by asserting that these are just the cases where the classes truly overlap but we shall resist this temptation.

In order to find whether the density information in the pattern contributes to the success of the method we use the decision tree of figure 6 on binary patterns, defined by $p(x, y) = 1$ if the point $(x, y)$ belongs to the object and $p(x, y) = 0$ otherwise. The pairs $(a : b)$ under the boxes refer to this situation. Since the discrimination on shape alone yields 147 correct and 28 incorrect decisions it appears that the density information contributes but little. The performance is summarized in the tables 1 and 2.

| | | TRUE SHAPE | | | |
|---|---|---|---|---|---|
| | | Closed, Open | Wishbone | Cross | Trefoil |
| ASSIGNED SHAPE | Closed, open | 101 | 3 | 0 | 2 |
| | Wishbone | 1 | 21 | 1 | 1 |
| | Cross | 7 | 1 | 15 | 2 |
| | Trefoil | 0 | 3 | 0 | 17 |

Table 1. Misclassification using density

| | | TRUE SHAPE | | | |
|---|---|---|---|---|---|
| | | Closed, Open | Wishbone | Cross | Trefoil |
| ASSIGNED SHAPE | Closed, open | 102 | 3 | 3 | 0 |
| | Wishbone | 6 | 23 | 1 | 8 |
| | Cross | 1 | 2 | 11 | 3 |
| | Trefoil | 0 | 0 | 1 | 11 |

Table 2. Misclassification using shape alone

## THE RECOGNITION OF CHROMOSOMES OF THE 13–15 GROUP AND THE PROBLEMS OF MISSING OBSERVATIONS

The classification of human chromosomes to the groups recognized by the London report (1963) is usually done by relative size and centromeric index, where the relative size of a chromosome is defined as

$$\frac{\text{size of chromosome}}{\text{total size of all chromosomes in cell}}.$$

In principle, therefore, we may not make any assignments until we have found the sizes of all the chromosomes in the cell and the technique must be modified when the size of even one chromosome is unavailable.

If, however, we could identify all (or even most) of the chromosomes belonging to a particular group then missing observations are no longer a problem. We merely write

$$\text{(assumed) total size of all chromosomes in cell} = \frac{\text{actual size of chromosome in given group}}{\text{usual relative size of chromosome in given group}}$$

and then define, for other chromosomes,

$$\text{(assumed) relative size of chromosome} = \frac{\text{actual size of chromosome}}{\text{(assumed) total size of all chromosomes in cell}}.$$

Now inspection reveals that the chromosomes of the 13–15 group are usually of the wishbone shape, shown in figure 3.3, which can be reliably distinguished by the decision tree of figure 6. Analysis of four cells gives the results in table 3.

| | TRUE GROUP | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 13–15 | other | 13–15 | other | 13–15 | other | 13–15 | other |
| ASSIGNED SHAPE | | | | | | | | |
| Wishbone | 6 | 2 | 4 | 1 | 5 | 3 | 4 | 0 |
| Other | 0 | 37 | 2 | 35 | 1 | 36 | 2 | 39 |
| | Cell 1 | | Cell 2 | | Cell 3 | | Cell 4 | |

Table 3.

The values for individual cells do not sum to 46 because not quite all the chromosomes in each cell were analysed. We assume that among those picked out as 'wishbones' the members of the 13–15 group should form a subset with the least variation in size. We therefore select the set of four chromosomes of least range in size as a subset of the 13–15 group and estimate the total cell size as above. The process may be conveniently tabulated as shown in table 4.

| | *Size of chromosomes picked as wishbone* | *Estimated cell size* | *Actual cell size* | *Discrepancy* |
|---|---|---|---|---|
| Cell 1 | 143,179,211,214,217,229,239,267 | 12,800 | 13,502 | 6% |
| Cell 2 | 192,221,226,230,237 | 13,450 | 13,740 | 3% |
| Cell 3 | 117,198,212,225,238,255,259,274 | (a) 14,370<br>(b) 15,070 | 14,152 | 2%<br>6% |
| Cell 4 | 241,242,248,265 | 14,660 | 14,273 | 3% |

Table 4

In cell 3 two sets of four were adjudged equally good since their ranges of 34, 36 were very close. In each case the four chromosomes were all members of the 13–15 group. The discrepancy between estimated and true cell size arises because of the variation in size among members of the 13–15 group.

## THE CONIC SKELETON

At this point it may be of interest to compare our notion of skeleton with those used by other authors. Kirsch *et al.* (1957) have suggested that a certain class of digitized picture may be meaningfully described in terms of finite segments of straight lines; McCormick (1963) and Narasimham (1964) have used this approach successfully in the description and analysis of bubble chamber photographs. For binary pictures in which pattern boundaries are well defined, Blum (1964) has suggested a description in terms of the medial axis, which is the focus of pattern points which have no unique nearest-boundary point, distance being defined only along paths wholly within the pattern. Blum has shown that the medial axis and its associated set of distances to the boundary are sufficient to allow regeneration of the original pattern. The medial axis is conveniently found in practice by successive deletion of pattern points as described by Philbrick (1966) and Rosenfeld and Pfaltz (1966).

As we have said, Blum's technique operates on binary patterns with well-defined boundaries. However, chromosome patterns are not binary and their boundaries are not very well defined (Patau 1965). Taking account of the density information present in the chromosome pattern, Hilditch (1969) creates a skeleton by deleting points from the pattern in such a way that the original connectivity is preserved and, subject to this, high density points are preserved rather than low density ones.

This ensures that the skeleton tends to lie along the higher density ridges of the pattern rather than in a central position determined by outline alone.

The conic skeleton we have defined is a thin-line representation of the original chromosome pattern, which seems relatively stable with respect to those changes of thresholds which produce small changes of outline. For most chromosome patterns the skeleton lies in a position which seems intuitively appropriate. This is due more to the simplicity of most chromosome patterns than to any general power in the conic skeleton technique. We have only to compare the conic skeleton and the Hilditch skeleton of an $n$-pointed star (with $n>5$) to realize what restrictions we have placed on the skeleton.

The task of extending the conic skeleton technique to deal with more general shapes seems very difficult. Formally it is sufficient to replace the quadratic form $Q(x, y)$ by the $n$-ic form $Q_n(x, y)$ with $\sum_{2}^{n+1} j$ independent coefficients, and the minimization of the quantity $D=\iint p(u, v)\,|Q_n(u, v)|^2\,du\,dv$ leads to an $n$-ic approximation $Q_n(x, y)=0$. Although it is easy to see how such things could be used in particular cases, e.g., '$S$' as a cubic,

'8' as the product of two circles, '*H*' as the product of three lines – the general technique has several drawbacks. As $n$ increases computation time increases as $n^6$ and instead of becoming a closer approximation to the main structure of the pattern the $n$-ic skeleton passes indiscriminately through more and more pattern points until skeleton and pattern coincide. There are no 'principal axes' for the general $n$-ic form and such systematic classifications as do exist (Salmon 1879) have little intuitive appeal as pattern descriptors.

One reason why the conic skeleton is a comparatively crude approximation to the pattern is that all pattern points affect the form of the skeleton as a whole. By contrast the other skeletons are defined in terms of sequential local operations (Rosenfeld and Pfaltz 1966) in which pattern points affect directly the form of the skeleton only in their own neighbourhood. It is therefore only natural that such skeletons should reflect the local structure of the pattern more faithfully than does the type of skeleton we have proposed. However the operations involved in our technique (formation of moments, solution of the eigenvalue problem, inscription of the best conic) are all well suited to rapid execution by a parallel processing computer, whereas the other skeleton techniques are more suited to sequential processing.

The connectivity of the conic skeleton may well be different from that of the pattern. Following Rosenfeld and Pfaltz (1966) let us say that two points are adjacent in the integral plane if their distance apart is less than 1·5 and that a subset $S$ of the integral plane is connected if for any two points $q$, $r$ of $S$ there exist points, $p_0, p_i, p_n$ of $S$ such that $p_0=q$, $p_n=r$ and $p_i$ is adjacent to $p_{i-1}$ for $1 \leqslant i \leqslant n$.

The labelling algorithm ensures that the set $B$, the pattern, is connected (for if not it would be regarded as the union of two or more distinct patterns). In most cases $B$ is simply connected though when chromosome arms touch $B$ becomes doubly connected, and so on.

The set $A$, the best conic, has two components in the case of most hyperbolae and one component otherwise. However the set $C$, the conic skeleton, may have more than two components; for example, figure 8 shows a twisted chromosome where the best conic, a hyperbola, leaves the chromosome and rejoins later after traversing part of the background. The largest number of components so far observed in a conic skeleton is four. It is not surprising that the skeleton is not an accurate representation of the pattern as far as connectivity is concerned for it is designed in a manner that takes no account of topology.

## CURVILINEAR CO-ORDINATES

It has been suggested (Paton 1969) that the best conic may be used to define a system of curvilinear coordinates with whose aid the pattern could be straightened out along and perpendicular to its best conic approximation. Unfortunately the system chosen has proved to be of no help in chromosome analysis.

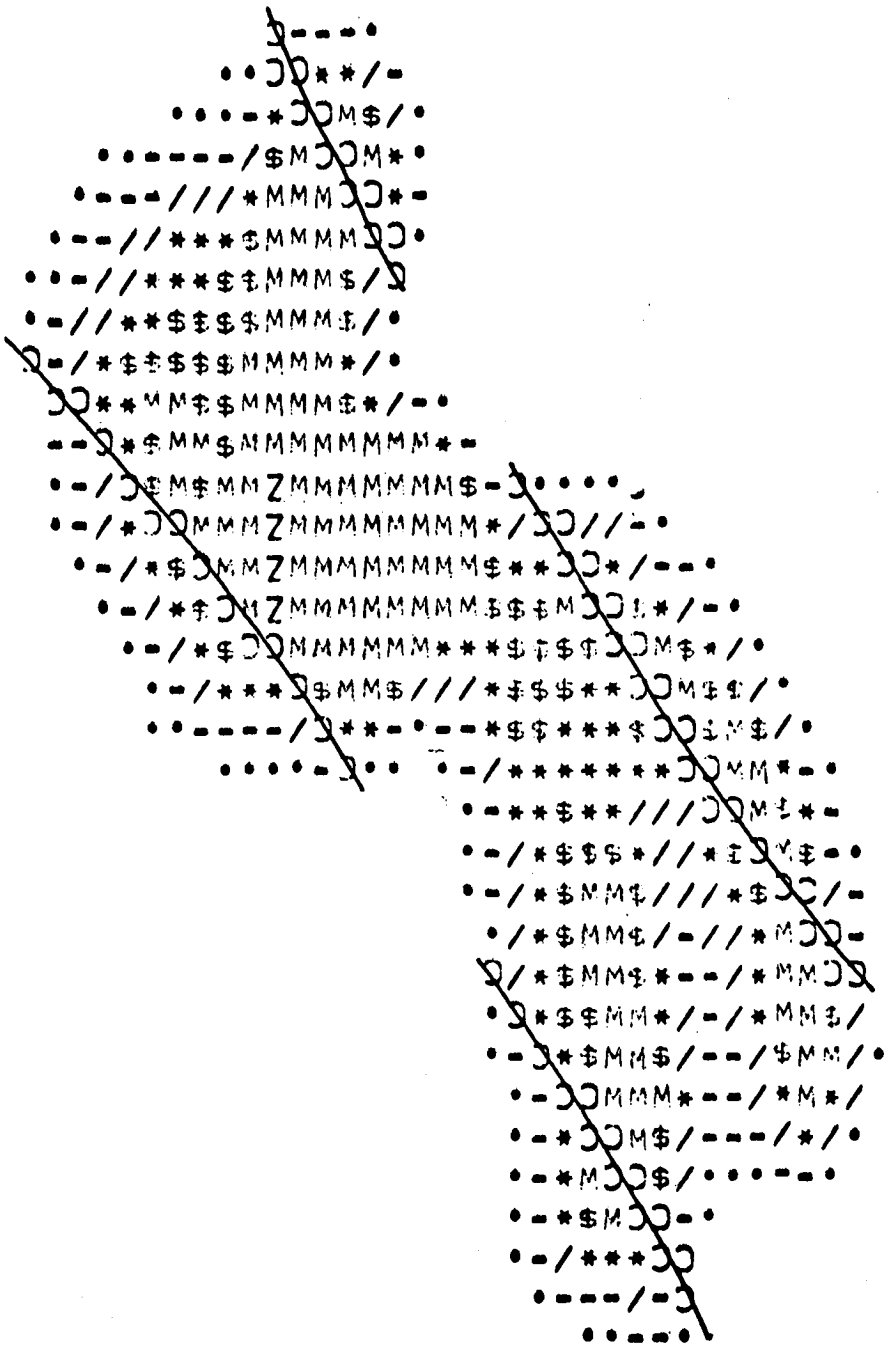


Figure 8. A twisted chromosome and its skeleton

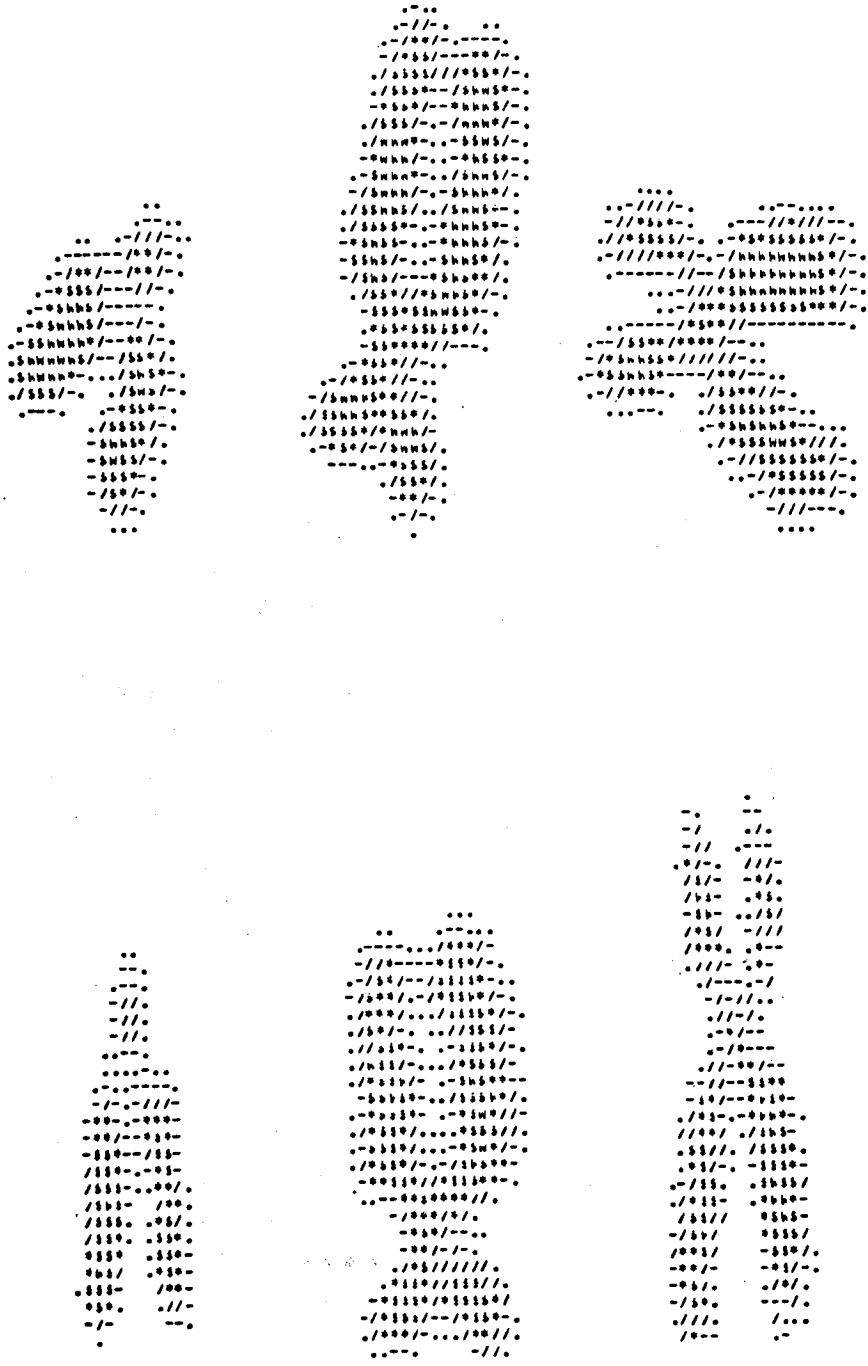


Figure 9. Some chromosome patterns (above) and their affine transforms (below)

## AN AFFINE TRANSFORMATION

An interesting transformation suggested by the best conic is that which yields a pattern whose best conic has its axes in some chosen ratio. Suppose that $p(x, y)$ denotes the pattern, translated to the conic centre and rotated so that the X-axis lies along the conic major axis. Let the major and minor semi-axes of the best conic for $p$ have length $a$, $b$ respectively and suppose that a new pattern $g$ is defined by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} q & 0 \\ 0 & q^{-1} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \text{ and } g(u, v) = p(x, y).$$

Then the lengths of the semi-axes of the best conic for $g$ have length $a'$, $b'$ given by $a' = qa$ and $b' = b/q$. Thus, if the desired axis-ratio is $r$ then $q$ should be chosen as $\sqrt{(rb/a)}$. When the best conic is a hyperbola the new pattern $g$ is a representation in which the arms of the chromosome are constrained to lie, as far as is possible, along the lines inclined at $\pm\theta$ to the $U$-axis, where $\theta = \tan^{-1}(r^{-1})$. Figure 9 shows some examples of this transform for an axis-ratio of $r = 5{\cdot}6$.

## CONCLUSION

The value of the conic section approximation in automatic chromosome analysis cannot be discussed without considering in some detail the relative merits of the various techniques used in centromere finding. For the moment, therefore, we shall be content to record that the average time spent in deriving the best conic is 0·36 seconds (on the IBM 7094) and that the success rate of the very simple decision rule given in figure 6 is about 86%.

### Acknowledgements

The work described in this paper is part of the research of the MRC Clinical and Population Cytogenetics Research Unit, to whose members I record my thanks for assistance and advice. Mrs Muriel Szewczyk typed the manuscript. I would like to thank Dr D. Rutovitz for his keen interest and helpful criticism and Dr C.C. Spicer for his encouragement of the continuation of this work in the MRC Computer Unit.

## REFERENCES

Alt, F.L. (1962) Digital pattern recognition by moments. *Optical character recognition*, p. 153 (ed. Fischer). Washington: Spartan Books.

Blum, H. (1964) *A transformation for extracting new descriptors of shape.* Bedford, Mass: Air Force Cambridge Research Laboratories.

Butler, J.W. *et al.* (1968) Automatic classification of chromosomes. *Data acquisition and processing in biology and medicine, V* (ed. Enslein). New York.

Court Brown, W.M. (1967) *Human population cytogenetics.* Amsterdam: North Holland.

Evans, H.J. (1962) Chromosome aberrations induced by ionizing radiations. *Int. Rev. Cytol.*, **13**, 221.

Gallus, G. (1968) Contor analysis in pattern recognition for human chromosome classification. *A.b.d.c.e. n.* 2.

Guiliano, V.E. *et al.* (1961) Automatic pattern recognition by a gestalt method. *Information and Control*, **4**, 332.

Halmos, P. (1958) *Finite dimensional vector spaces.* Princeton, N. J.: Van Nostrand.
Hilditch, C.J. (1969) Linear skeletons from square cupboards. *Machine Intelligence 4,* pp. 403–20 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.
Hilditch, C.J. (1969a) Chromosome recognition. *Annals of New York Academy of Sciences,* **157,** 1, 339.
Hilditch, C.J. (1969b) A system of automatic chromosome analysis. *NATO Summer School on Image Processing, Pisa* (in press).
Hu, M.K. (1962) Visual pattern recognition by moment invariants. *IRE Trans. on Information Theory,* **IT–8,** 179.
Jacobs, P.A. *et al.* (1964) Cytogenetic studies in leucocytes on the general population. Subjects of ages 65 years and more. *Ann. Human Genetics,* **27,** 353.
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.
Ledley, R.S. *et al.* (1965) FIDAC. Film input to digital automatic computer and associated syntax-directed pattern-recognition programming system. *Optical and electro-optical information processing,* p. 591 (ed. Tippett). Cambridge, Mass: MIT.
Ledley, R.S. & Ruddle, F.H. (1965) Automatic analysis of chromosome karyograms. *Mathematics and computer science in biology and medicine.* London: Medical Research Council.
Ledley, R.S. *et al.* (1966) BUGSYS – a programming system for picture processing – not for debugging. *CACM,* **9,** 2, 79–84.
London Report (1963) The London conference on the normal human karyotype. *Cytogenetics,* **2,** 264–8.
McCormick, B.H. (1963) The Illinois pattern recognition computer – Illiac III. *IRE Trans. Electronic Comp.* **EC–12,** 5.
Mendelsohn, M.L. *et al.* (1969) Computer-oriented analysis of human chromosomes. II Integrated optical density as a single parameter for karyotype analysis. *Annals of New York Academy of Sciences,* **157,** 1, 376.
Moorhead, P.S., *et al.* (1960) Chromosome preparations of leukocytes cultured from human peripheral blood. *Experimental Cell Research,* **20,** 613.
Narasimhan, R. (1964) Labelling schemata and syntactic descriptions of pictures. *Information and Control,* **7,** 151.
Neurath, P.W., *et al.* (1969) Man-machine interaction for image processing. *Annals of New York Academy of Sciences,* **157,** 1, 324.
Patau, K. (1965) Identification of chromosomes. *Human chromosome methodology,* p. 155 (ed. Yunis). London: Academic Press.
Paton, K. (1969) Conic sections in chromosome analysis. Pattern Recognition (in press).
Philbrick, O. (1966) *A study of shape recognition using the medial axis transform.* Bedford Mass: Air Force Cambridge Research Laboratories.
Rosenfeld, A. & Pfaltz, J.L. (1966) Sequential operations in digital picture processing. *J. Ass. comput. Mach.,* **13,** 471–94.
Rutovitz, D. (1967) *Machines to classify chromosomes? Human Radiation Cytogenetics.* Amsterdam: North Holland.
Rutovitz, D. *et al.* (1969) A system of automatic chromosome analysis. Symposium on medical uses of computers. Birmingham.
Rutovitz, D. (1970) Centromere finding: some shape descriptions for small chromosome outlines. *Machine Intelligence 5,* pp. 435–62 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.
Salmon, G. (1879) *A treatise on the higher plane curves.* New York: Stechert.
Tjio, J.H. & Levan, A. (1956) The chromosome number of man. *Hereditas,* **42,** 1.

## APPENDIX 1: THE DERIVATION OF A CANONICAL SET OF MOMENTS

Let $p$ be a non-negative function on the real plane, with $p$ zero except within a finite distance of the origin. Let the $(i,j)$th moment of a function denoted by a small letter be that capital letter with suffix $(i,j)$

thus $P_{ij}=\iint x^i y^j p(x,y)\,dx\,dy.$

*Step* 1. Let $T=P_{00}$, $\bar{x}=P_{10}/P_{00}$, $\bar{y}=P_{01}/P_{00}$ and define $g(x-\bar{x}, y-\bar{y})=T^{-1}$ $p(x,y)$. Then

$$G_{00}=1;\ G_{10}=G_{01}=0. \tag{1}$$

Here $T$ is the total density of the pattern $p$, and $\bar{x}$, $\bar{y}$ the position of its centroid in the original frame of reference.

*Step* 2. Let $r=\sqrt{(G_{20}+G_{02})}$ and define $h(x/r, y/r)=g(x,y)$. Then in addition to (1),

$$H_{20}+H_{02}=1. \tag{2}$$

Here $r$ is the radius of gyration of the original pattern about its centroid. The function $h$ will be called the standard pattern for $p$.

*Step* 3. If the axes are rotated through $\theta$, and $u$, $v$, $k$ defined by

$$\begin{pmatrix}u\\v\end{pmatrix}=\begin{pmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix};\ k(u,v)=h(x,y)$$

then $K_{11}=[(H_{02}-H_{20})\sin 2\theta]/2+H_{11}\cos 2\theta$ and this vanishes when

$$\tan 2\theta=2H_{11}/(H_{20}-H_{02}).$$

Unless $H_{20}=H_{02}$ and $H_{11}=0$ there are four solutions in $[0, 2\pi)$. Let us choose the one which makes $K_{20}<K_{02}$, and $K_{21}>0$. Then in addition to (2),

$$K_{11}=0,\quad K_{20}<K_{02},\quad \text{and}\quad K_{21}>0. \tag{3}$$

Here $K_{20}$ is the minimum second moment about the centroid and $K_{02}$ the maximum. The $V$ axis defined in terms of $\theta$ is the line of best least squares fit or principal axis for the pattern $p$.

*Step* 4. Let $d=\pm 1$ and define

$$m(d,y)=k(x,y).$$

Then $M_{12}=dK_{12}$ and unless $K_{12}=0$, $d$ may be chosen so that in addition to (3),

$$M_{12}>0. \tag{4}$$

The function $m$ will be called the canonical pattern for $p$.

## APPENDIX 2; REDUCTION OF A CONIC TO PRINCIPAL AXES

Let the conic be represented by the equation $Q(x,y)=ax^2+2hxy+by^2+2gx+2fy+c=0$. Then if $ab-h^2$ does not vanish the conic has a centre at the point $(x_0, y_0)$ defined by

$$ax_0+hy_0+g=0$$
$$hx_0+by_0+f=0$$

and the equation of the conic referred to parallel axes through the centre is

$$ax^2+2hxy+by^2+\frac{T}{ah-h^2}=0,$$

where $T$ denotes the determinant $\begin{vmatrix} a & h & g \\ h & b & f \\ g & f & c \end{vmatrix}$.

If $T$ does not vanish this may be rewritten in the form

$$\left(\frac{h^2-ab}{T}\right)(ax^2+2hxy+by^2)=1.$$

If the axes are rotated through $\theta$ so that the equation becomes

$$\left(\frac{h^2-ab}{T}\right)(a_1u^2+2h_1uv+b_1v^2)=1$$

then $h_1$ vanishes when $\tan(2\theta)=2h/(a-b)$. This yields two values of $\theta$ in $[-\pi/2, \pi/2)$ of which one gives $|b_1|\leqslant|a_1|$. By choosing this we can write the equation of the conic in the final form

$$\delta_1\frac{u^2}{A^2}+\delta_2\frac{v^2}{B^2}=1,$$

where $A\geqslant B>0$, $\delta_1$ and $\delta_2$ are independently $\pm1$ and $\delta_1\delta_2$ is the sign of $ab-h^2$. Here $2A$, $2B$ are the lengths of the major, minor axes. In fact, $\delta_1\delta_2$ is $+1$ for an ellipse, $-1$ for a hyperbola. In the latter case $\delta_1$ is $+1$ if the major axis cuts the conic and $-1$ if it does not. Let $A'=\delta_1 A$ and $B'=\delta_2 B$. Then it is clear that the variables $(x_0, y_0, \theta, A', B')$ completely specify the conic; it is shown in books on conic sections how all these quantities may be derived from the variables $(a, 2h, b, 2g, 2f, c)$.

*Note.* If $a=b$ and $h=0$ the conic is a circle, real or imaginary as $T$ is negative or positive.

# 24

# Centromere Finding: Some Shape Descriptors for Small Chromosome Outlines

Denis Rutovitz
Medical Research Council
Clinical and Population Cytogenetics Research Unit
London

**Abstract**

Location of the centromere line is the key to successful chromosome analysis, though a considerable reject rate can be tolerated provided the failure rate is insignificant. Procedures for centromere location are best arranged as a series of increasingly complex alternatives, perhaps finishing with a path for operator intervention, each of which must calculate confidence in its own success. An existing scheme of this nature, incorporated in a chromosome screening program, begins with density profile analysis on the assumption that the chromosome is reasonably straight and symmetrical, tries a number of variations on this theme, and if necessary resorts to skeletonizing to straighten out bent figures before profiling. Results are unsatisfactory for smaller chromosomes, possibly because of poor density measurements of small objects. A number of schemes of boundary analysis have been surveyed. Determination of spanning chords and finding the arc-below-chord depth appears usually to solve the problem, but preliminary overall curve fitting seems necessary in the more difficult cases. Expanding a polar coordinate representation of the boundary as a trigonometric polynomial seems to offer a satisfactory smooth curve approximation, and is directly useful in determining possible axis orientations and in some cases in distinguishing between acrocentric and metacentric chromosomes.

## INTRODUCTION

Automatic classification of human chromosomes comprises a number of different types of operations.

(1) First, a metaphase cell of sufficient quality to admit of karyotyping must be located on the slide being examined. This involves mechanical as well as

optical scanning as only a small portion of a slide can be held in an optical field of sufficient resolution at any one time. Criteria suggested for metaphase cell detection include such characteristics as total variation at a sample spacing related to the average width of chromosome arms, and other parameters which can be extracted by fairly simple special hardware. (2) Once a cell has been located the optical field must be segmented into its connected components. The aim at this stage is to separate out individual chromosomes, though this is not always possible because in some cases chromosomes will be overlapping or broken. The pattern recognition problem is that of distinguishing between whole single chromosomes, composites, pieces, or extrinsic material. One could include here a subdivision of chromosomes into different general types, and in particular the recognition of bizarre configurations, such as ring chromosomes. In any event, it is important to provide opportunities for modifying the initial segmentation to try to compensate for wrong thresholding, scanner artefacts, etc. (3) The measurement and classification of individual objects.

Naturally there can, and indeed should, be possibilities of feed-back between stages 3 and 2, as the measurement process involves recognition aspects as well. The measurements upon which chromosome classification is based, whether by subjective estimate on the part of the human operator or by measurement and calculation in a machine, are:

(a) Size. Traditionally, chromosome size has meant chromosome length, although the human cytogeneticist often compensates for variable contraction by 'adding a bit' for excessively dark chromosomes, and so on. Length is in any case something of an accident as the apparent length of a chromosome arm depends very much not only on its state of contraction but also on any folding or twisting that may have taken place when the cell was deposited on the slide. In principle, the integrated optical density should be a more reliable size measure as it should approximate to an optical estimate of the DNA content of the chromosome. In practice its biological constancy is betrayed by the difficulties of microdensitometry of objects only one or two wave lengths in width, and the photometric variability of scanning instruments, and few researchers have been able to make effective use of it as yet (*see*, however, Mendelsohn *et al.* 1969). Area has been shown a tolerable interim substitute (Rutovitz *et al.* 1970) – with a somewhat higher discriminating power than length, and of course area measurement does not pose any pattern recognition problem, other than that of the initial segmentation, whereas length requires location of arm tips and possibly the determination of curved paths in bent arms.

(b) Centromeric Index. This is the ratio of the size of the short arms of a chromosome to the size of a whole chromosome. Again this has been usually regarded as a ratio of lengths, but integrated density or area measurements are biologically preferable and somewhat simplify pattern recognition

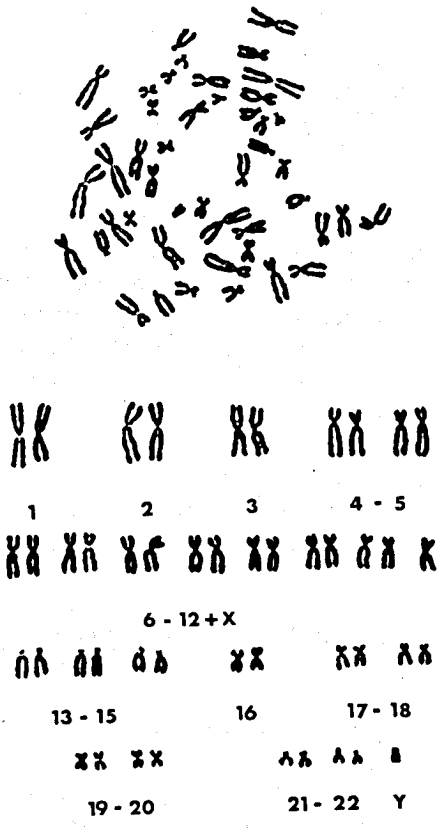
problems. Scanner photometry is usually sufficiently uniform over the area of a single chromosome to make the use of the integrated density ratio a practical proposition; then the pattern recognition problem reduces to that of locating a 'centromere line', that is, a line through the chromatid junction cutting both chromatids into two parts and separating the chromosome into small arm and long arm regions. Once such a line has been found, the centromeric index calculation consists simply of finding the sum of densities on each side of the line, and taking the ratio of the smaller one to that of the sum.

A significant feature of the problem is that an error in centromeric index calculation is far more damaging than failure to find the index. The reason is that it is possible to classify chromosomes by size alone with fair reliability, even though the area of ambiguity is widened and the likelihood of chromosome interchange between adjacent groups (see figure 1) is increased. However, a wrong centromeric index may cause a perfectly normal chromosome to be regarded as abnormal or may give rise to gross misclassification. Another point is that in an interactive system, failure to recognize the centromere automatically would trigger a call for operator assistance, and so enable the determination of the index after all, albeit in a more costly way. Thus any procedure for finding the centromere line and centromere index should calculate a measure of the confidence in its own success, which can then be tested at higher levels to decide on subsequent operations. Since also all tactics seem to find the centromeres of some chromosomes and fail on others, the strategy of a centromere-finding program should be to try a hierarchy of different fail-safe approaches arranged in the order that minimizes expected costs, where cost is probably best interpreted rather widely as taking into account store utilization, computer time, the cost of misclassification of chromosomes in terms of the requirements for additional cell analysis which may be engendered, and some appraisal of the ultimate cost of wrong analysis in terms of false positives and false negatives. Unfortunately, the determination of cost in this sense is a very lengthy exercise, but computer costs are quite straightforward and we have attempted to realize the principle in this more limited context.

The methods of centromere location currently in use by different workers (at least so far as it is known to us) are the following:

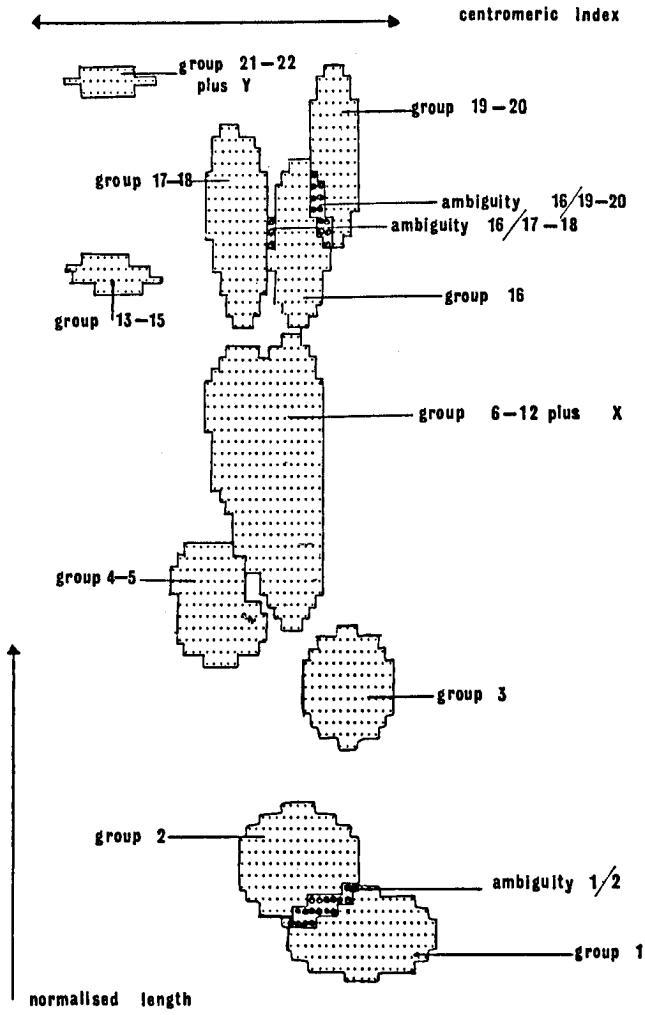
(1) Analysis of local curvature of the boundary by various methods. This approach was pioneered by Ledley (1962), initially in a purely syntax-directed setting, although a good number of *ad hoc* refinements have been introduced in his method since. In particular the exact position of the centromere is now found by a search for a local minimum in the lengths of chords joining points on the periphery of opposing chromatids, the tips of the chromatids having first been located by the syntactic procedure. Gallus and Neurath (1970) have recently introduced a simplified form of curvature analysis which aims specifically at finding the regions of negative

curvature which indicate the ends of the centromere line; the method then requires an analysis of the symmetry of the shape in relation to the various prospective centromere lines joining pairs of opposing concavities.
(2) Choice of a principal direction, with analysis of either boundary width and/or the number of intervals perpendicular to an axis direction to fix on the centromere or the centromere region as a narrow and single interval strip between wider and multiple interval strips. Systems of this type have been used by Gallus *et al.* (1968), Stone, Littlepage and Clegg (1967), and others.




(a)

(3) The determination of a principal axis direction followed by an analysis not of the object width but of the integrated density across scan lines perpendicular to the axis. This type of density based approach was suggested by Mendelsohn *et al.* (1969) and has been much worked on by ourselves.




(b)

Figure 1. (a) A metaphase spread with its chromosomes arranged into ten groups conventionally recognized by cytologists. (b) Positions of chromosome groups in the centromeric index/normalized length plane (based on J. Lejeune's figures for 50 spreads).

4. Finding a Blum-type skeleton of the object, followed by examination of the relevant parts of the resulting graphical structure.

The percentage success or failure reported by various investigators depends in part on the extent to which the material has been specially selected for automatic analysis. For example, at the Lawrence Radiation Laboratory, Stone *et al.* (1967) have been obtaining nearly 100 per cent success with idealized chromosome shapes traced on vellum and then touched-up by hand before scanning. Other reports have indicated success rates anywhere between 70 per cent and 95 per cent, but because of the very variable quality of the specimens tested, it is hardly possible to estimate the relative reliability of different methods and it will not be possible to do so until those in the field begin exchanging photographs, slides, and tapes for comparative assessment.

In our work we have attempted to use cells typical of those attempted by humans, and have a test population selected according to operator judgments of what would be acceptable to *them*, without other restriction, from a range of good, average, and bad preparations. The real test will only come when we have automated the selection of cells as well, for the important thing will be to do well on those cells which the automatic system selects. Meanwhile,

| | Right | Wrong | Attempt Abandoned | Number |
|---|---|---|---|---|
| | | per cent | | |
| Large Chromosomes (Pairs 1–5) | 97·0 | ·7 | 2·3 | 132 |
| Middle Group (6–12X) | 95·0 | 1·0 | 3·5 | 198 |
| Small Chromosomes (13–22Y) | 71·0 | 14·5[1] | 14·5 | 281 |
| All | 84·4 | 7·0 | 8·3 | 611 |

[1] The centromeric index is frequently correct, or nearly so, even if the centromere line is wrong! This is because in an acrocentric chromosome any line not cutting deeply into it gives a small value for the index.

Table 1. Centromere location in 14 cells by optical density profile analysis

we have gathered some statistics from part of our current test population, and these are shown in table 1. From this can be seen that our centromere finding is very reliable for the larger chromosomes; the reject rate starts to climb for the middle group chromosomes and both the error and reject rates are substantial for the small ones. Nevertheless it seems to be just possible for us to produce acceptable results in screening programs requiring about 20 cells per subject: this is because in producing a composite karyotype from many cells, the rather scattered nature of the pattern recognition errors which we commit does not give the clustering of chromosomes with abnormal measurements which would be associated with genuine biological variants.

It will also be seen that our present difficulties centre round the smaller chromosomes, that is, those in groups 13–15 and up. We have been working for some time on techniques aimed specifically at improving results in this area; in the next paragraph the present centromere-finding scheme will be described in detail, and this will be followed by a description of more recent work which has yet to be incorporated in the system.

## CENTROMERE FINDING BY PROFILE ANALYSIS

As remarked above, the centromere-finding scheme for which performance statistics are presented, attempts to realize the notion that a succession of procedures should be employed, each calculating its own confidence index. When considered in conjunction with other indices already found, and perhaps extraneous information such as the expected number of chromosomes of a particular variety, each new confidence estimate predicates a decision to proceed to another method or to consider the status achieved as satisfactory, or the best obtainable. The procedures are arranged in an order estimated (but not shown) to be that which gives the maximum return for the minimum cost. Thus the first method tried is an economical calculation which succeeds in about 64 per cent of cases. If a satisfactory confidence index is not achieved by this procedure a number of variations of the same approach are attempted. If all fail, we proceed to a more complex and time-expensive approach which adds a fair number of successes to the results. If all available methods fail, the centromere line is at present considered unobtainable, and the object is classified later by size alone. In a subsequent version of the system it is very likely that operator aid will be sought at this point.

The basic first attempt procedure is optical density profile* analysis in which it is assumed that the chromosome is reasonably straight and symmetrical, and that an approximate axis of symmetry can be found by fitting

* Given a rectangular coordinate system $(i, j)$, a domain $D$ in the $(i, j)$ plane, and a non-negative function $d(i, j)$ defined on $D$,

$$l(i) = \sum_{j>0,\,(i,j)\in D} d(i,j)[+\tfrac{1}{2}d(i,0)\text{if } (i,0)\in D]$$

is the left half (density) profile of $d$ with respect to the $i$-axis; the right half profile is similarly defined for $j<0$, the whole profile is $w(i)=l(i)+r(i)$

the best straight line to its density distribution. However, a number of different paths are followed depending on the values of various simple size and shape parameters already obtained. Taking the length and breadth of an object to mean the dimensions of the minimal containing rectangle with one side parallel to the axis of best least squares fit, we consider the following shape indices:

(1) Rectangle fit is the ratio of the area of the object to the area of this rectangle.
(2) The breadth/length ratio of the rectangle measures the 'fatness' of the object.
(3) The size of the object in relation to other chromosomes in the cell. At this stage this is taken to be the ratio of object area to median object area for the cell (the median is more suitable than the mean, as it is less sensitive to the presence of large extraneous objects such as undivided cell nuclei).
(4) A symmetry coefficient of the object in relation to the axis.

Let $(i, j)$ be a coordinate system, $i$ measures the distance along the axis and $j$ measures the distance perpendicular to it. Let $m_{ij}$ be the density of the object at $(i, j)$ (estimated by interpolation from the densities at points in the original coordinate system). Then the coefficient is defined by the formula

$$\sigma = 100\left\{1 - \frac{\left|\sum_i \sum_{j>0} m_{ij} - \sum_{j<0} m_{ij}\right|}{\sum_{ij} m_{ij}}\right\}.$$

Thus $\sigma$ is 100 if in each line perpendicular to the axis the sum of the densities to one side of the axis exactly equal those on the other; $\sigma$ is 0 if in each line all the density points are on one side of the axis only. This is a rather weak form of symmetry measurement, but empirically seems about right for the purpose.

If the object is neither small nor wide, the principal axis is usually correct. In this case we obtain the whole and half profiles (see figure 2) and use a specialized procedure for finding significant maxima and minima in the profiles. This ignores local minima with valley depth less than a prescribed percentage of the mean peak-to-valley height, and also ignores minima with valleys less than some preset fixed depth. For each profile it returns the number, position, and height of peaks and valleys found. If the two half profiles each have two peaks and if the position of the valley between is nearly the same; if the centromeric indices for the two half profiles calculated separately are sufficiently close and if the symmetry index of the chromosome is sufficiently high, then it is accepted that the centromere is at the position indicated by the valleys and the centromeric index is taken as the mean index for the two half profiles. A confidence number is attached to this, calculated as a function of the differences between valley positions and centromeric indices, and the symmetry value. If any of the tests fail, we proceed to try the next method, with the exception that if indices or centromere positions are close enough to be almost acceptable, the peak and valley calculation is

done for the whole profile as well, and the results compared with those for the half profile.

If a profile does not reveal any significant troughs, that is, there is one peak with no valleys, it is usually, but not always, an acrocentric chromosome (a chromosome with the centromere near the end). In this case it frequently happens that although there is no actual dip in the profile there is a point of inflection or shoulder and (as first pointed out by Gallus and Montanaro)

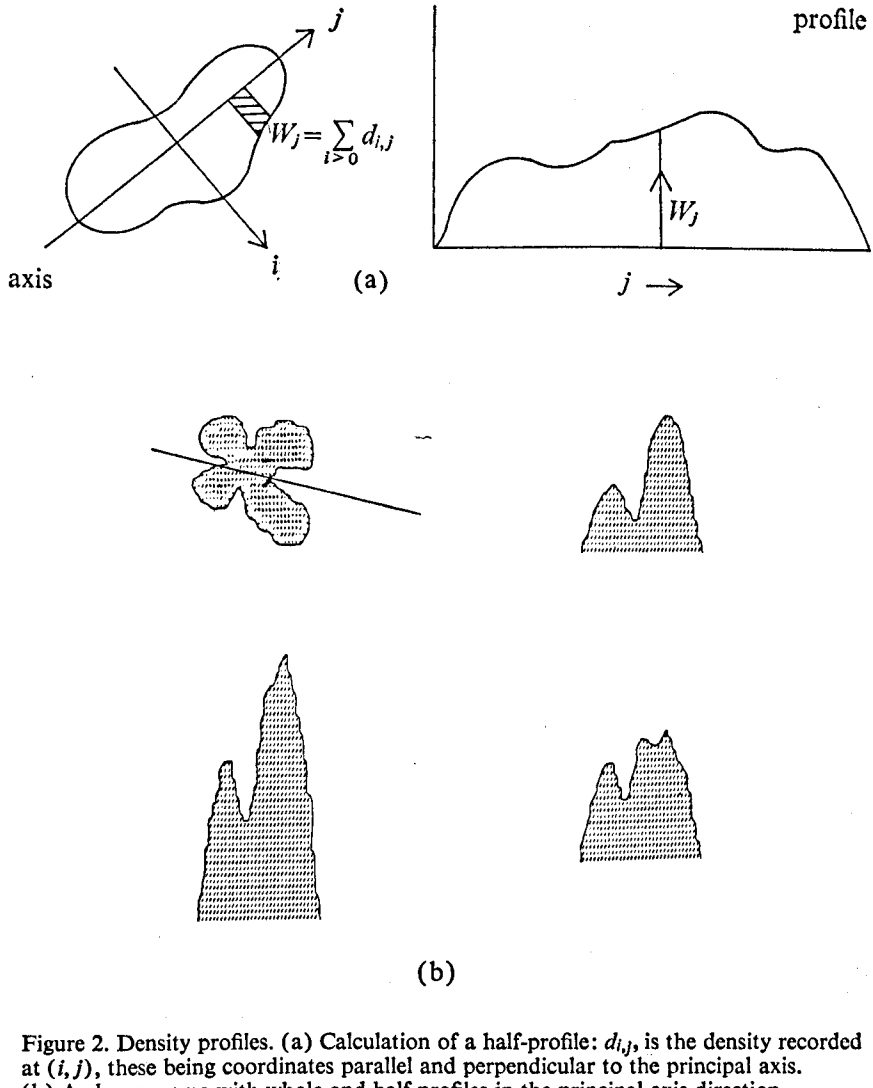
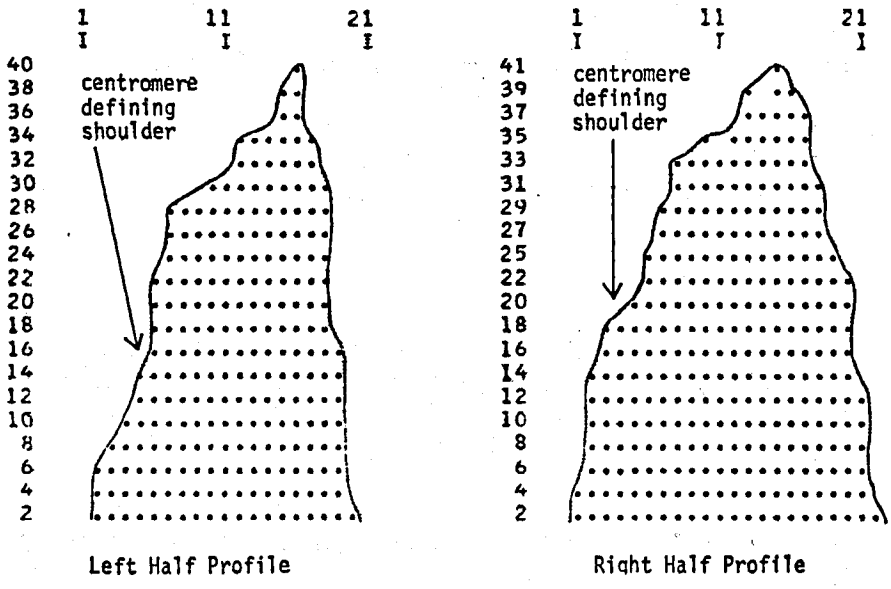



Figure 2. Density profiles. (a) Calculation of a half-profile: $d_{i,j}$, is the density recorded at $(i,j)$, these being coordinates parallel and perpendicular to the principal axis. (b) A chromosome with whole and half profiles in the principal axis direction.




LINE PRINTER RECONSTRUCTION OF AN OPTICAL DENSITY DISTRIBUTION

LINES 121 TO 140. COLUMNS 364 TO 381.

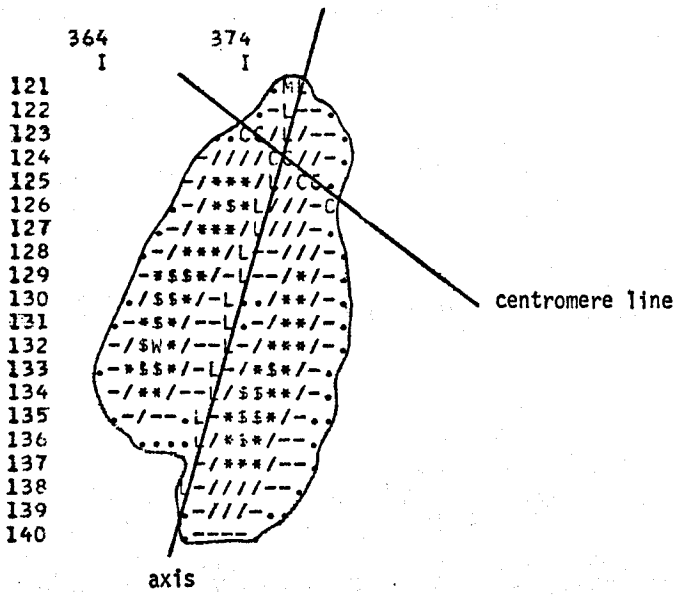



Figure 3. An acrocentric chromosome (a 13–15) with left and right half profiles showing centromere defining shoulders. The centromere line and axis found by the machine are marked in the object by C and L respectively.

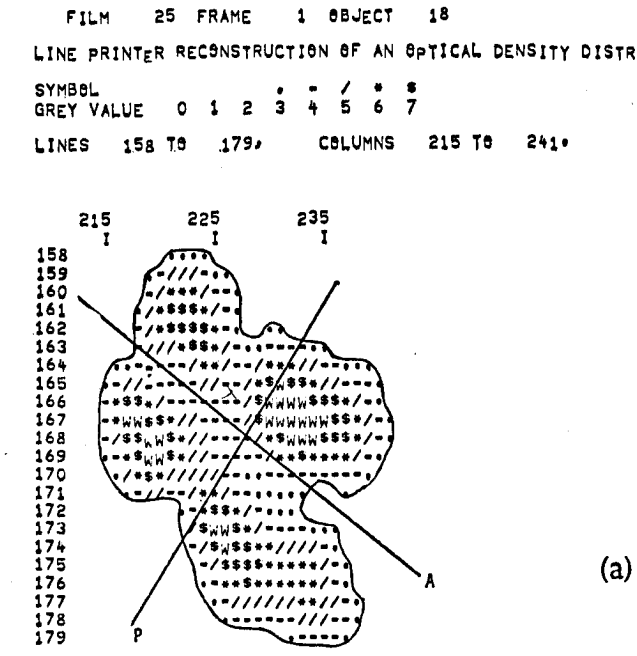



Figure 4. (a) A 2/2, 2/2 configuration. Half profiles summed perpendicular to both *A* and *P* directions all have two humps. (b) A 2/1, 1/1 configuration. Half profiles summed perpendicular to *A* will each be single-humped; the right half profile perpendicular to *P* is 2-humped, the left one is single-humped.

such shoulders define the centromere position as effectively as an actual dip (see figure 3). For this reason if no actual dip can be found a special inflection-seeking subroutine is entered which seeks minima in the differenced profile. If exactly one shoulder is found this is regarded as a possible centromere position and is treated in precisely the same way in the subsequent analysis as an actual minimum would be.

If the method fails at the first attempt, and we are dealing with a large or thin chromosome, the angle is varied slightly and this continues until certain prescribed angle change possibilities have been exhausted. Thereupon a radically different procedure, 'skeletonizing' is invoked, as described below. If skeletonizing also fails to locate the centromere line with reasonable confidence, there is a choice of two courses of action: if the object is large it is regarded as possibly a composite and in the controlling system an attempt will then be made to split it and re-enter the component parts in the system. In the case of a smallish object however the centromere is simply regarded as unobtainable.

A slightly different path through the profiling procedure is followed when objects are small or fat. In this case the program begins by seeing how well the object fills its containing rectangle parallel to the axis. A poor fit means that the axis is unreliable and in this case it immediately enters the skeletonizing procedure. A good fit means that either the principal axis or perpendicular axis is likely to be right, so half profiles both parallel and perpendicular to the axis are obtained. The number of peaks in the two pairs of half profiles are then counted.

If the two pairs show a 2/2 and 2/2 peak configuration (i.e., two peaks in each half profile), one or other of the axial or perpendicular directions is probably correct and whichever of these two directions gives best symmetry is taken as being the axis. A 2/1 and 1/1 configuration usually indicates that the 1/1 axis is right for an acrocentric chromosome (see figure 4). A 2/2 and 2/1 configuration is possible from chromosomes with one pair of arms closed as shown in figure 4a, and in this case we accept the 2/2 axis if it also gives very good symmetry. In other cases, we proceed to try the next method.

## SKELETONIZATION

The skeletonizing procedure has been described in detail elsewhere, and in particular much of the ground-work has been covered in presentations at earlier Machine Intelligence Workshops (Hilditch 1968, 1969). In brief, however, the procedure is the following: a density-sensitive stripping procedure is used which progressively removes points from the periphery of an object, while maintaining connectivity and preserving line ends, until no further points can be removed. The points of lower density are removed first, and low density connections may be broken. The resulting pattern consists of arcs, i.e., connected subsets of points each of which has exactly two neighbours in the pattern, and nodal areas which are connected subsets

of points having other than two neighbours. If the arc node structure is represented by an adjacency matrix, a first test for an acceptable chromosome skeleton is that the graph should be a tree. If not, we consider the skeleton method as having failed. If it is a tree, we do a distance transform outward from the skeleton, in such a way that not only the distance but also the path-wise nearest point is propagated (*see* Rutovitz 1968), thereby finding in two passes the skeleton point which is nearest to each chromosome point, whose intensity is then added in to that at the nearest skeleton point. When this has been completed, we have replaced the original density distribution by an equivalent 'beaded' skeleton. The skeleton is now decomposed into two half skeletons according to type: for example, if there are four tips, ends on corresponding chromatids are paired according to proximity and angle rules. The minimal connecting subgraph between the ends of each chromatid is then taken as defining the chromatid skeleton (variations on this procedure provide individual chromatid skeletons in all cases). The whole profile can be formed by adding these two, once a point-to-point correspondence has been obtained. In effect, we are straightening out the arms of the chromosome and then projecting densities on to best lines, and so reducing complicated curved and twisted shapes to simpler configurations. Once the profiles have been obtained by these means, we enter the profile analysis subroutines as before.

## ANALYSIS OF SMALL CHROMOSOME BOUNDARIES

The system as described produces information which is only just sufficiently accurate for our composite karyotyping scheme to succeed. The failure and reject rate for the smaller chromosomes is much too high to permit subtle discrimination of biological events involving them. In part this is due to our having based the entire scheme on density-sensitive methods in the belief that the complete optical intensity distribution throughout a chromosome contains more reliable and more relevant information than the boundary only, however determined. As an article of faith we still believe this. Unfortunately the difficulties of densitometric scanning with objects whose cross-section is only a wavelength or so, and with scanning equipment of limited density resolution, are such as to provide very imperfect information for the smaller chromosomes. Since it is possible for a human to spot the centromere line on calcomp tracings with a very high success rate, there evidently is adequate information in the boundary for this purpose (although sometimes it is difficult to believe!). For some time now, therefore, we have been studying boundary-only methods of centromere location with a view to improving the recognition rate for the smaller objects. Unfortunately, most of the methods described by workers who have concentrated on the boundary do not seem to offer substantially higher success rates for the difficult chromosomes than we have been achieving with the density profiles, although the incorporation of any alternative method into the system in use should

improve overall performance, because usually one method will succeed where another fails. It must be remembered, however, that we require methods which add to the success rate despite confidence bound settings which prevent an increase in the failure rate, and this is a more stringent requirement. In particular we have experimented with local curvature analysis of the Gallus type and examined some of Gallus's own results and have been left with the impression that a very worthwhile gain would be had by the incorporation of his methods into the system, but that the question of confidence estimates for this type of approach still needs further investigation.

Another technique under consideration is the following: If centroid-centred polar coordinate representations of the boundaries of smaller objects were obtained, it might be possible to find arm tips, the indentations marking the centromere, and the arm cleavage, by a straightforward analysis of the local turning points of the radius $r$ as a function of the angle $\theta$ from some datum line. Also, a three-peaked $r,\theta$ list might correspond to acrocentric profiles ('trefoil' shapes) and a four-peaked function should usually correspond to metacentric chromosomes or 'quadrifoil' shapes. (It turns out that this is only partially correct and that there are too many exceptions for the number of peaks alone to be useful in distinguishing acrocentrics from metacentrics.)

Supposing that the centromere itself cannot be found from the polar coordinate list, it might still be possible to put it to valuable use. If the frequency with which the axis is correctly determined can be increased, the number of successes obtained by the profile method will rise. Further, if there is substantial evidence that the axis found is the right one, some of the acceptance thresholds used in these routines can be relaxed, thereby reducing the number of trials made and so improving running time. For example, in 'trefoil' chromosomes the 'right' axis means a line running approximately through the mid-point of the centromere line to the 'v' between the opposite arms (*see* figure 5). With approximately circular objects the axis of best least squares fit is not at all reliable, and hence a contribution from another source would be very welcome. Now, one of the maximal radii almost always does coincide with the axis, but determining which one it is can be difficult. It was at first thought that the axis would pass through the $r$-minimum which was the smallest in proportion to the size of the adjacent $r$-maxima in length, but a glance at the examples in figure 5 will show that it is not the length of the minima in relation to the maximum $r$ that matters but the depth of the arc in this region below a spanning chord. Again, the centromere-defining shoulders do sometimes correspond to minima of the radius from the centroid, but this is only true for almost completely symmetrical trefoils such as some of the 21–22Y group; with the longer acrocentrics, such as 13–15s, two of the $r$-minima usually fall in the center of the longer sides (figure 6). But if there is an indentation at the centromere it should be revealed by an arc-below-chord test.

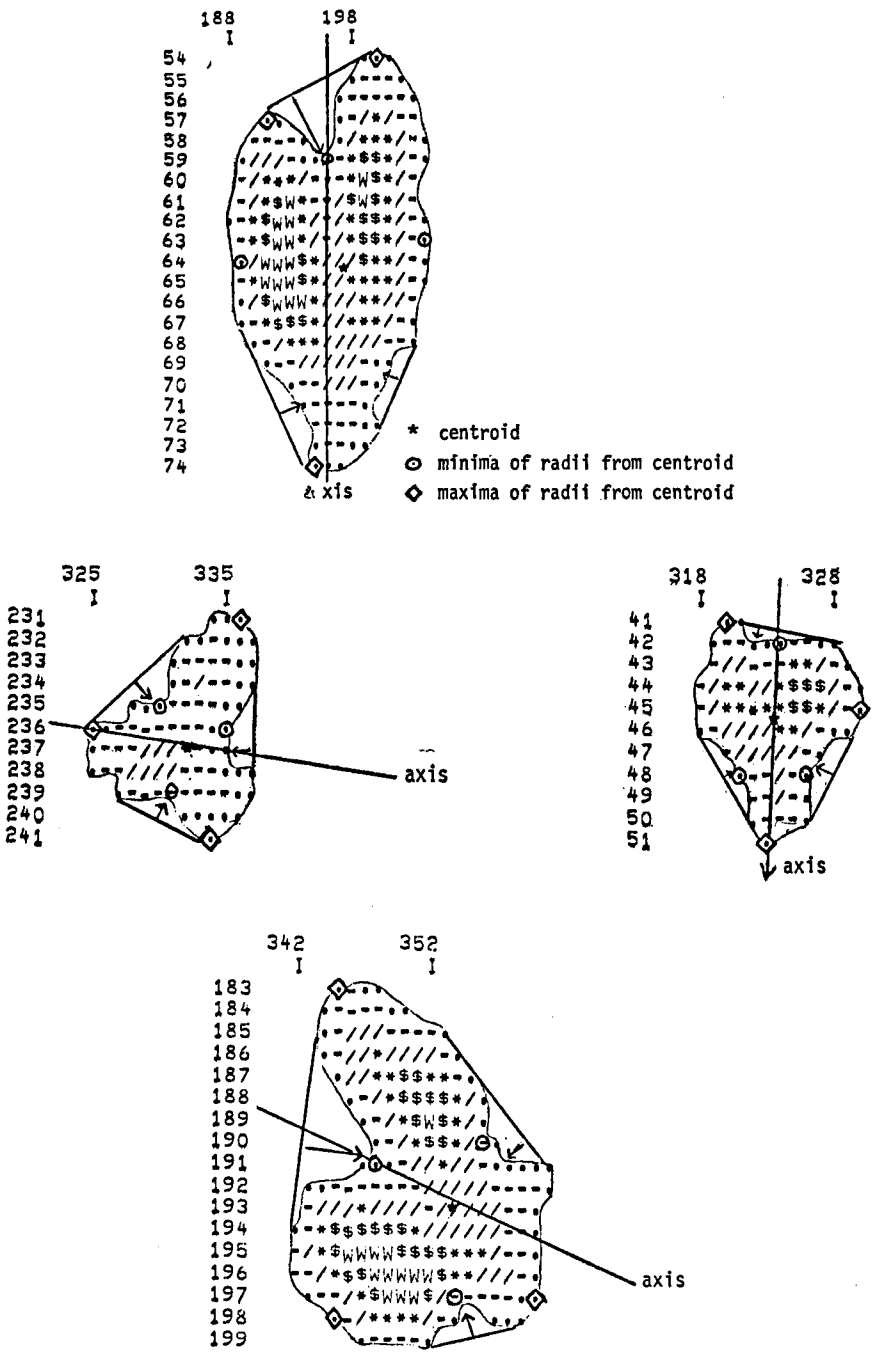
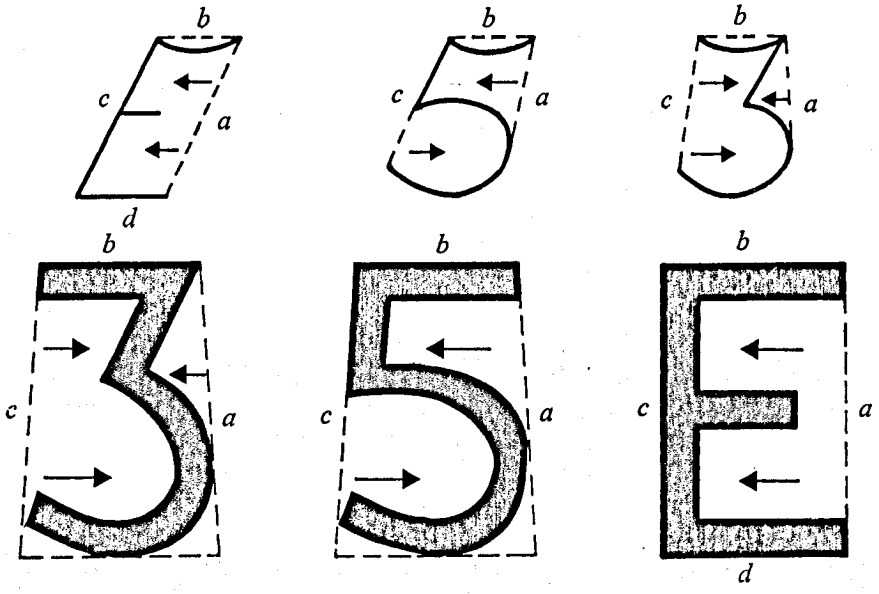
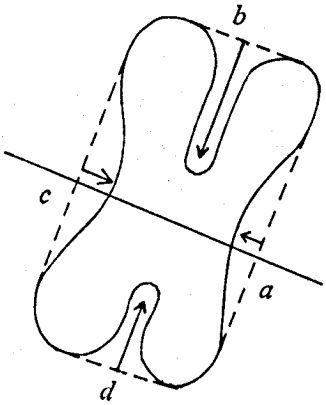
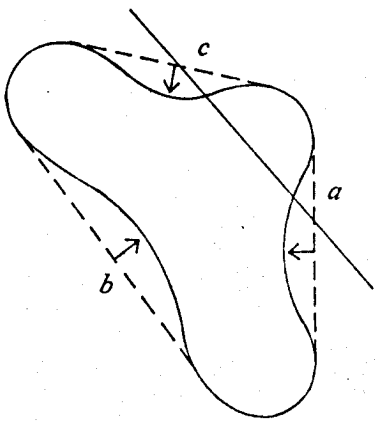
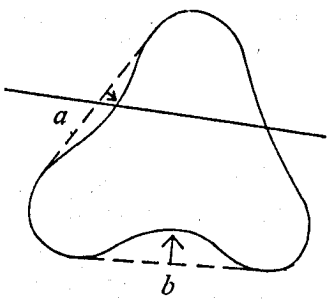



Figure 5. Some acrocentric chromosomes with maxima and minima of radii from the centroid, depths of minima below spanning chords and principal axes shown.


b
c
a
d
b
c
a
b
c
a
b
c
a
b
c
a
b
c
a
d


b
c
a
d


c
a
b


a
b

| | Lower Right Chromosome | Lower Left Chromosome | Upper Chromosome |
|---|---|---|---|
| Valley deeps | $\delta_a=\delta_b=1$<br>$\lambda_b>\lambda_a$ | $\delta_a=\delta_b=\delta_c=1$<br>$\lambda_b>\lambda_a, \lambda_c$ | $\delta_a=\delta_b=\delta_c=\delta_d=1$ |
| Length relations | | arc$(a,c)$ <arc$(a,b)$, arc$(b,c)$ | (midpoint) distance $b,d$ <distance $a,c$ |
| Inclination | $40°<\theta_{a,b}<70°$ | $50°<\theta_{a,b}, \theta_{b,c}, \theta_{a,c}<70°$ | $\theta_{b,d}<30°$, $\theta_{a,c}<30°$<br>$\theta_{a,b}>60°$, $\theta_{c,d}>60°$ |
| *cn* line | passes through $(x_a,y_a)$, $\perp^r$ to $b$ | passes through $(x_a,y_a)$ and $(x_c,y_c)$ | passes through $(x_a,y_a)$ and $(x_c,y_c)$ |

| | Figure 3s | Figure 5s | Letter Es |
|---|---|---|---|
| Valley deeps | $\delta_a=1$, $\delta_c=2$ | $\delta_a=1$, $\delta_c=1$ | $\delta_a=2$, $\delta_b=\delta_c=\delta_d=0$ |
| Position | $y_b>y_a,y_c$<br>$x_a>x_b,x_c$<br>$y_{c,1}<y_{a,1}<y_{c,2}$ | $y_b>y_a,y_c$<br>$x_a>x_b,x_c$<br>$y_{c,1}<y_{a,1}$ | $x_a>x_b,x_d>x_c$<br>$y_a>y_{a,1}>y_c>$<br>$y_{a,2}>y_d$ |
| Inclination | $\theta_a,\theta_c>70°$<br>$\theta_b<20°$ | $\theta_a,\theta_c>70°$<br>$\theta_b<20°$ | $\theta_a,\theta_c>70°$<br>$\theta_b,\theta_d<20°$ |

Figure 6. External chord list descriptions of some alphanumeric characters and chromosomes. For brevity, the following conventions are used: lower case letters denote line segments. The center of segment 1 is at $x_1, y_1$; its inclination with the $x$-axis (assumed parallel to the upper and lower edges of a page of text) is $\theta_1$ and the angle between segments $a$ and $b$ is $\theta_{a,b}$. The number of 'valley deeps' (local maxima) in the arc-below-chord depth is $\delta_1$ occurring at $(x_{1j}, y_{1j})$, with depths $\lambda_{1j}$. Each of the characters depicted in the figure possesses the features listed in exactly one of the columns.

From these considerations we were led to investigate the possible uses of spanning chords and support polygons in general, as had earlier been proposed for some purposes in chromosome analysis by Evans and Sweeney (1965), and in another context by Attneave (1965), Freeman (1961) and others.

The time taken to find an enclosing convex polygon can be considerable if

one simply looks for support lines in different directions. However, a list of external spanning chords can be obtained in an acceptable time by means of a repeated traverse of the boundary list in which

(a) chords between neighbouring points are generated.

(b) adjacent chords are combined if the triangle formed by them and the join of their opposite ends either lies outside the object, or is of area less than some preset threshold.

It seems natural to complement a list of external chords with a list of internal ones linking them; though since the external chords (including those which merely join adjacent points) initially enclose the whole boundary, it is necessary to introduce rules for choosing certain of the external chords as significant and disregarding the others. Chord length is one possible criterion, but the length of arc subtended by a chord is a better one, as a short chord that spans a long arc is an important shape determinant.

A chord list having been obtained, the next step is to fill in a property list for each of them; the contents of the property lists will naturally depend on the context in which they are used, but the properties that seem most obviously to be of interest are: length, angle, length of arc subtended, maximum depth

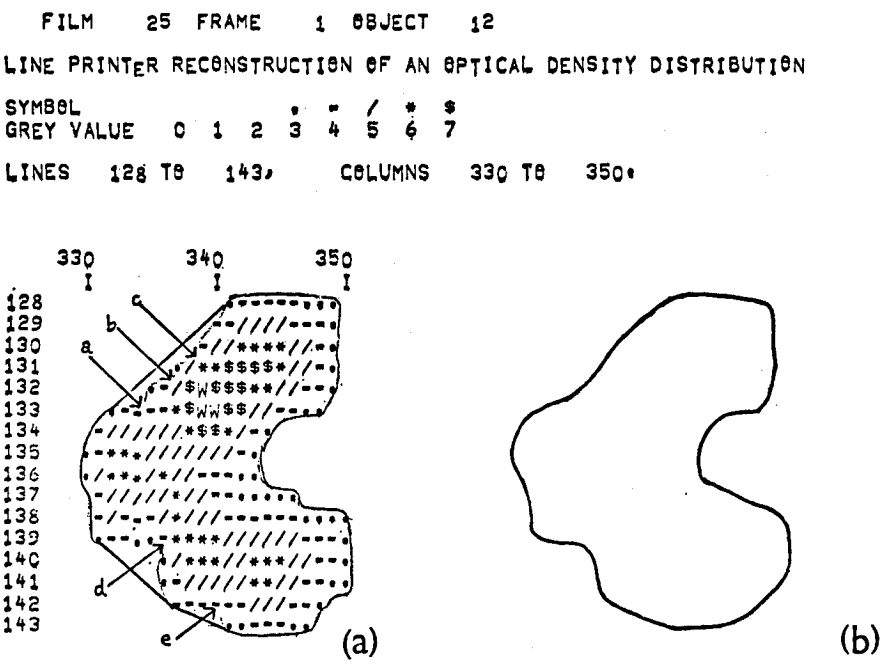



Figure 7. Where is the shoulder? Most observers perceive the chromosome boundary as being rather like that of the outline figure on the right, but measurement shows only slight differences between the depth of indentations at *a*, *b*, *c* and *d*, *e*. The measurements reflect only local features of the boundary: to produce (b) from (a) would seem to necessitate the use of global curve fitting techniques such as the use of trigonometric polynomials and polar coordinates as described in the text.

of arc-below-chord, number of local minima and maxima in the below-chord-depth and perhaps the area between the arc and the chord. In figure 7 are shown a few examples of alphabetic and numeric characters described by chord lists in which only external chords are used, and the sole property recorded is that of the number of local minima and their depths below the chord, and the orientation of the chord. It would seem that this provides a sufficient characterization of the shapes of these letters, and one which would be invariant under most of the distortions to which printed characters are subject. Figure 6 shows the same style of chord description applied to some acrocentric and metacentric chromosomes, and it seems intuitively plausible that the centromere-locating information is contained in such chord lists. Indeed, given some such statement as 'a shape with three external spanning chords at approximately 60° to each other, with a relatively small single valley below two of them and a larger single valley beneath the other', one could draw something approximating to a trefoil chromosome shape without ever having seen one. This is a long way from showing that the chord list is a complete characterization, but sufficient to encourage the expectation that it is adequate in the context. At the time of writing, this hypothesis has not been tested on sufficient data to reach a definite conclusion, but the first indications are at least very promising.

## BOUNDARY LIST PROCESSING

In this paragraph we give some details of the boundary processing techniques employed. We are not concerned here with the more basic problem of optimal boundary determination, but only with the question of how to work with boundaries, once defined.

By a boundary list we mean a tree structure of which each constituent is a list which contains

(a) entries defining a polygonal path in a coordinate plane

(b) an indication as to whether it is an internal or external boundary of the domain concerned

(c) a pointer to the immediate surrounding boundary (of a hole or an 'island', as the case may be), if any

(d) individual properties or property list pointers.

In the form in which they are held in our system, the lead-in to the list family contains miscellaneous additional information such as a type code and a 'wrap-number' telling how many of the entries at the beginning of each list are repeated at its end: it is very helpful to have 'wrap-around' of this kind in dealing with circular lists as it obviates the necessity of special-case treatment of the ends. No attempt is made to pack the numbers occurring in boundary lists as the chromosome problem is one in which the field naturally segments into numbers of fairly small, discrete components. It is never necessary to handle more than one or two component boundaries at a time, and these are all within core store capacity: packing would only waste time.

In the system used, the field components have already been separated out before boundary analysis commences, and the form in which objects are presented for boundary processing is that of a list of the lines of the picture in which a current component is situated, and for each line, a list of constituent intervals (Rutovitz 1968). The first step in boundary analysis is to extract from this 'interval-end-point form' of domain representation, the set of alternate line and column numbers which would be encountered in traversing all parts of the boundary, moving only in directions parallel to one of the coordinate axes, and keeping the object always on the right (i.e., going clockwise).* Thus a rectangle with vertices at (0, 0), (10, 0), (10, 20), (20, 0) would yield the sequence (0, 10, 20, 0), apart from wrap-around. In a list of this type with entries $l_1$, $k_1$, $l_2$, $k_2$, $l_1$, the successive boundary points are $(l_1, k_1)$, $(l_2, k_1)$, $(l_2, k_2)$, $(l_1, k_2)$. We term this the 'alternate coordinate' form of boundary list, and it is extracted from the interval end point list by an efficient but rather messy procedure which goes from the one form to the other as directly as possible by means of special-case programming to deal with each different interval configuration.

In the alternate coordinate type of list, one skips directly between points quite a long way apart provided that they can be joined by a straight line parallel to one of the axes and forming part of the boundary. Once this list has been found it is appropriate to examine it for collinear point sequences in other directions. Naturally for off-axis directions the boundary must be staircase-like, and so one can only seek lines as perfect as the digitization allows – that is, lines in which at most one of the two coordinates changes by more than 1 unit at each step, the ratio of the $x$ increments to the $y$ increments being constant. A procedure for doing this has been implemented which also transforms the alternate coordinate list into successive point form. (This requires twice the number of entries for figures such as a rectangle with sides parallel to the axis, though usually the overall list length is about the same.) For example, a figure consisting of the points marked × in the diagram

```
11  ×
12  ×  ×
13  ×  ×  ×
    1  2  3
```

would have the boundary list

(11, 1, 12, 2, 13, 3, 13, 1)

in alternate coordinate form, and

(11, 1), (13, 3), (13, 1)

in successive point form.

* Our domains are all finite subsets of the integer points of an $(x, y)$-plane. Two points are adjacent if their coordinates do not differ by more than 1. A point is in the boundary of a domain if, and only if, it is adjacent to a point not belonging to the domain.

This transformation is also useful in connection with perimeter computation, as after transforming it is reasonable to take the sum of the lengths of the joining segments as defining perimeter (which is not the case when in alternate coordinate form). Incidentally, as processing speed is important, segment length calculation is done by table-look-up for small $\Delta x$ and $\Delta y$ otherwise by direct square-root-of-sum-of-squares calculation.

A suite of subroutines for various further transformations of boundary lists has been implemented (largely the work of G. Regoliosi, personal communication). This replaces the successive point list by absolute or relative neighbour direction lists, which can be used as input lists for Gallus-type curvature extraction (Gallus and Neurath 1970). In particular it includes a procedure for extending any straight line section of the boundary as far as possible in either direction subject to the running average deviation from the line remaining with preset bounds.

Polar coordinate lists are obtained from successive point lists in a straightforward way by replacing the $x, y$ coordinates of each point by their $r, \theta$ coordinates referred to the centroid. Once again, calculation of $r$ and $\theta$ is done by a combination of conventional calculation and table look-up (this would be a good application of memo functions, were they available). A detail that should be mentioned is that in order to avoid the possibility of missing an $r$-minimum occurring within a straight line segment, additional points are interpolated in the list if the current joining segment contains a point at which the radius vector is perpendicular to it.

The chord list can, of course, be obtained either from the alternate coordinate or the successive point list. But since the chords form part of the convex support polygon, which must include the interval envelope,* it is more economical to start with special processing of the envelope. We begin, therefore, by working down its two sides as follows: Let $p=(x_1, y_1)$, $q=(x_2, y_2)$, $r=(x_3, y_3)$ be successive points going down one or the other edge of the envelope, and consider the value of the determinant

$$\Delta = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

If $\Delta$ is positive, the chord $(p, r)$ is outside the figure. If $\Delta$ is greater than some small negative threshold, it is reasonable to consider $(p, r)$ as still being an external chord (this takes care of minor indentations in the edge). In this case we replace $q$ by $r$ then reset $r$ to the value of the next point on the edge. If, on the other hand, $\Delta$ is less than this threshold value, we consider the external chord as having terminated at $q$, enter $(p, q)$ in the chord list, reset $p$ to $q$, $q$ to $r$ and take $r$ to be the next point going down the edge. Care has to be taken, of course, to keep to the clockwise convention on the left and

* I.e., the minimal set of line-segments parallel to one of the coordinate axes which covers the figure. The line segments in the interval envelope are immediately available from the complete interval list which is used to describe the domain.

anticlockwise on the right edge, and the first and last chords in each figure must be treated as special cases. However, this establishes a primary external chord list very rapidly and, if the same determinant-based rule is used to decide whether or not to combine adjacent chords, a stable list of maximal external chords usually results after a very few circuits of the edge.

At this point we fill in a property list for each chord. At present the total information carried is: internal/external, beginning point, end point, perimeter of arc subtended, maximum depth of arc-below-chord, point at which this maximum occurs, angle made by chord with $x$-axis.

The property lists established for the external chords, we now proceed to obtain an internal chord list. These consist of lines joining the ends of external chords, after certain of the latter have been excluded from consideration. An external chord is considered significant only if the length of arc subtended exceeds a given threshold: as the calculation of this parameter involves reference to the successive point form of boundary list, and the identification of the chord's ends in this list, it is best to fill in the property lists of the external ones before looking at the internal ones. The same parameters are developed for internal chords as for external, though at present this information is not much used. Relations between chords are currently established by particular case programming, but this might well be an area in which a binary relation processor or a syntax-directed matching system would be of value.

## CURVE SMOOTHING: TRIGONOMETRICAL APPROXIMATIONS TO POLAR COORDINATE FIGURES

In many cases in which it is difficult to locate the centromere by other methods, the below-arc-depth of the centromere shoulder turns out to be insignificant (see figure 7). Yet the eye picks out the shoulder as a quite distinctive point, and one is led to believe that human perceptions of this type involve some kind of outline smoothing or perhaps overall curve fitting – the shape needs a dent at one point to compensate for a bulge at another, so to speak. In recent work by Gallus and Neurath (1970) an effective boundary refinement prior to curvature analysis is accomplished by application of a Laplacian difference operator to the density distribution prior to thresholding. Perhaps because the span of the operator is about the width of a chromosome arm it brings up edges rather well, and because of the second differences involved it highlights curvature of the density surface. Salient features of chromosome boundaries are often brought out in a quite dramatic way by these means. A possibly suitable method of global smoothing of the boundary would be to replace the original discrete $(r, \theta)$ list by an expansion of $r$ in orthonormal functions. Given that a boundary list is naturally periodic, and further that trefoil and quadrifoil shapes naturally suggest curves such as $r=a+b\cos(3\theta-\alpha)$, $r=a+b\cos(4\theta-\alpha)$ respectively, trigonometric polynomials are a first choice. Since the boundary lists are not very long, and look-up tables

can be used to obtain sine and cosine terms to a sufficient accuracy, the calculation will not be prohibitive providing that the terms converge rapidly.

In figure 8, the outlines of a trefoil and quadrifoil chromosome respectively are shown with approximating trigonometric figures of order 0–6. It will be seen that the approximation is very good for the sixth order polynomial and only slightly less so for the fourth order ones.

The trigonometric polynomials depicted are developed from the formulas

$$r=a_0+\sum_{k=1}^{n}(a_k\cos k_\theta+b_k\sin k_\theta),$$

where $\theta$ is the angle made with a fixed direction by the radius vector from the origin which is initially sited at the centroid of area of the chromosome (but see below), $r$ is the length of the radius vector in the positive $\theta$ direction ($-r$ is plotted at 180°), and the coefficients $a_n$ and $b_n$ are defined by the equations

$$a_0=\frac{1}{2\pi}\int_0^{2\pi} r\,\mathrm{d}\theta,$$

$$a_k=\frac{1}{\pi}\int_0^{2\pi} r\cos k\theta\,\mathrm{d}\theta,$$

$$b_k=\frac{1}{\pi}\int_0^{2\pi} r\sin k\theta\,\mathrm{d}\theta.$$

Since the boundaries defining the $(r, \theta)$ relationship consist of straight line segments, the integrals can be evaluated speedily provided we have an estimate for

$$\int_{\theta_1}^{\theta_2} r\cos k\theta\,\mathrm{d}\theta$$

as $r$ traverses a line from $(r_1, \theta_1)$ to $(r_2, \theta_2)$.

Expressing the equation of the join of $(r_1, \theta_1)$ and $(r_2, \theta_2)$ in the form $r=p\sec(\theta-\alpha)$, our problem becomes that of evaluating

$$I_n=\int_{\theta_1}^{\theta_2} \mathrm{e}^{in\theta}\sec(\theta-\alpha)\,\mathrm{d}\theta.$$

It is a straightforward matter to set down an algorithm to calculate $I_n$, given the following three relations:

$$\begin{aligned} I_0&=\int_{\theta_1}^{\theta_2}\sec(\theta-\alpha)\,\mathrm{d}\theta=\left[\log|\sec\theta+\tan\theta|\right]_{\theta_1-\alpha}^{\theta_2-\alpha}\\ &=\left[\log\left(r/p\pm\sqrt{r^2-p^2}/p\right)\right]_{\theta_1-\alpha}^{\theta_2-\alpha}\\ &=\left|\log\left((r_2+\sqrt{r_2^2-p^2})/(r_1+\sqrt{r_1^2-p^2})\right)\right|, \end{aligned} \tag{1}$$

where the sign choice inside the log argument and the modulus outside follow from the fact that the lines are always interrupted when perpendicular

to the radius – i.e., when $\theta=\alpha$; hence $\theta_2-\alpha$ and $\theta_1-\alpha$ always have the same sign.

$$I_1=\int_{\theta_1}^{\theta_2}\sec(\theta-\alpha)\,e^{i\theta}\,d\theta$$
$$=e^{i\alpha}[(\theta_2-\theta_1)-i\log(\cos(\theta-\alpha))]\Big|_{\theta_1}^{\theta_2}$$
$$=e^{i\alpha}[\theta_2-\theta_1+i\log(r_2/r_1)] \qquad (2)$$

$$I_n=-e^{2i\alpha}I_{n-2}+2e^{i\alpha}\int_{\theta_1}^{\theta_2}e^{i(n-1)\theta}d\theta; \qquad (3)$$

this reduction formula follows immediately on making the following substitution in $I_n$:

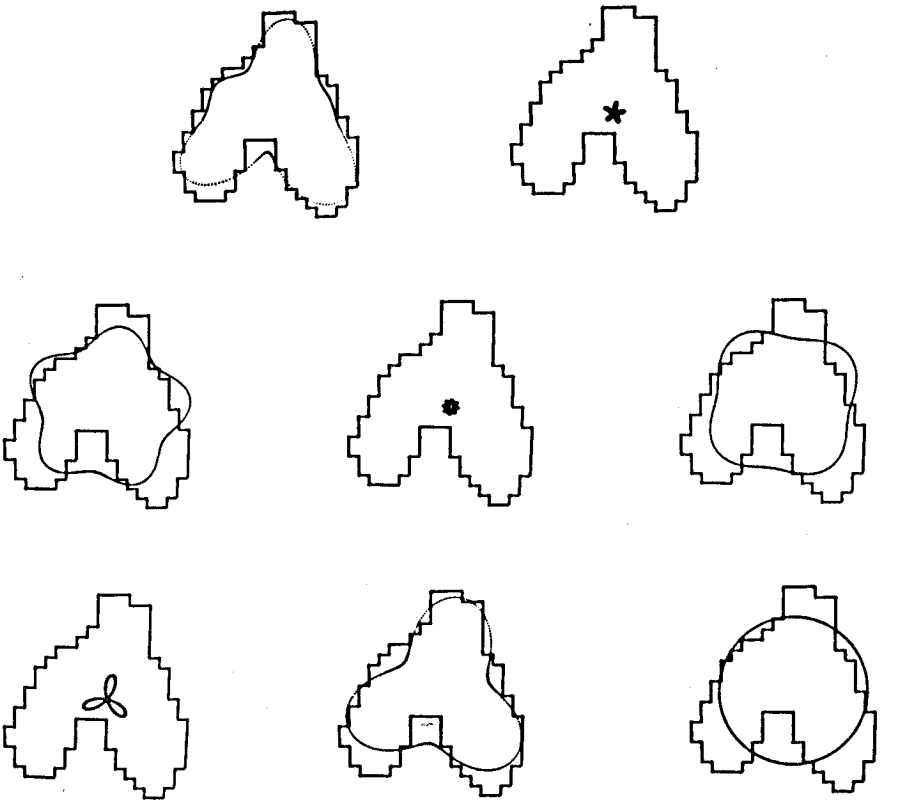
$$e^{in\theta}=e^{i(n-1)\theta}(e^{i\theta}+e^{-i\theta})-e^{i(n-2)\theta}$$
$$=2\cos\theta\,e^{i(n-1)\theta}-e^{i(n-2)\theta}.$$

The housekeeping for $I_0$ and $I_1$ is quite heavy but the computation to pass from $I_{n-2}$ to $I_n$ is minimal; since with this approach there is no restriction on segment length, and most chromosome boundaries contain a fair proportion of 2, 3, and 4 element steps, the average number of operations required is less than with any approximate method (as only the terms up to order 8, or thereabouts, are needed, the economies of the fast Fourier transform do not apply).

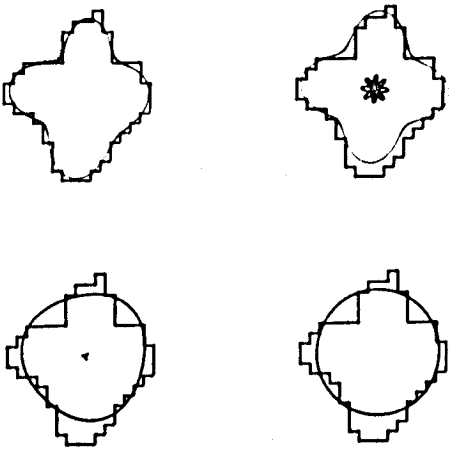
The coefficients are very sensitive to the working origin. Although the centroid of area seems a good starting point for their calculation, a better standardization of the center would be achieved if it were located so as to

---

Figure 8. (opposite) CRT displays of chromosome outlines and approximating trigonometric polynomial polar coordinate figures. (a) *Bottom row, right.* The constant only; *middle.* Third order and constant terms; *left.* Third order terms only. The maximum radius of the inner three-petalled figure is the magnitude of the third order coefficients, $\sqrt{(a_3^2+b_3^2)}$. The orientation of the inner figure depends on the ratio of the cosine coefficient $a_3$ to the sine coefficient $b_3$, that is, on the phase angle. It can be seen that its orientation corresponds very well with that of the chromosome. Note that $r$ is negative between the petals in odd order figures but negative parts are reflected back on the positive ones. *Middle row, right.* Constant and fourth order terms; *middle.* Fourth order terms only. They are much smaller than the third order terms, as might be expected for a trefoil shape, hence the smaller size of the figure (in even order curves the negative portions reflect in between instead of on the positive ones, hence doubling the apparent number of petals). *left.* Constant and fifth order terms. *Top row, right.* Fifth order terms only. They are larger than the fourth order terms, illustrating the phenomenon described in the note on Table 1, namely, that in trefoil figures the fifth order terms tend to reinforce the third order ones whilst the fourth and sixth order ones are small. *left.* All terms to order six. (b) A quadrifoil metacentric chromosome. *Bottom row, right.* Constant term only; *left.* The outer curve shows the constant and third order terms, the inner figure shows the third order terms only. In this case the third order contribution is almost negligible. *Top row, right.* The outer curve consists of constant and fourth order terms, the inner curve shows the fourth order terms alone. In this case they are clearly the dominant term in the expansion. *left.* All terms to order six.



(a)



(b)

make the first order coefficient zero. This is because a figure of the type $r=b\cos(\theta-\alpha)$ represents a circle whose circumference passes through the coordinate origin, the negative part being reflected back on to the positive portion. The size of this circle is a measure of the eccentricity of the outline with which it is being matched, as the coefficients can only be large if the positive products on one side are not offset by corresponding negative products on the other. There is no simply evaluated analytic expression to determine the exact point about which the coefficients will be zero, but an iterative shift to the center of the circle rapidly reduces the first order coefficients to a small value.

In a survey of 300 small and middle-sized chromosomes the root mean square discrepancy between the original $(r, \theta)$ function and its sixth order trigonometric polynomial representation only rarely exceeded 3 per cent, and in these cases most of this is taken up by seventh and eighth order terms. Certainly, therefore, trigonometric approximations are a means of condensation of information, and one might well attempt a direct shape classification based on coefficient values (after preliminary transformation to deal with orientation).

---

Let $a_k$, $b_k$ be the Fourier coefficients of polar coordinate representations of chromosome boundaries, as defined in the text; let

$$m_k=\sqrt{a_k^2+b_k^2},\quad f=\sqrt{m_3^2+m_5^2},\quad g=\sqrt{m_4^2+m_6^2}.$$

An analysis of the regression of the centromere index $I$ on the ratio $f/(f+g)$ yielded the expression

$$E=46f/(f+g)-1{\cdot}2$$

as a best linear approximation to the index, with regression coefficient 0·6 (88 chromosomes: all the small ones in 4 cells for which $m_7+m_8$ was less than $0{\cdot}05\times m_1$ and $f+g>0{\cdot}1m_1$).

The scatter of $E$ about $I$ is too large to allow direct approximation to the index. But the following results separating acrocentrics from metacentrics hold:

| $E$ | $I$ | Per cent of Chromosomes |
|---|---|---|
| <15 | <20 | 39 |
| 15–30 | Indeterminate | 53 |
| >30 | >20 | 8 |

Table 2. Centromere Index prediction from Fourier Coefficient Magnitudes

The relationship between cosine $(3\theta-\alpha)$ figures and trefoils, and cosine $(4\theta-\alpha)$ figures and quadrifoils is so suggestive as to deserve special examination.

It turns out that if the magnitude of the fourth order coefficients (i.e., $\sqrt{(a_4^2+b_4^2)}$) is substantially greater than the magnitude of the third order ones we are unlikely to be dealing with an acrocentric chromosome, and this might be used as a partial check on centromere location by other methods, or possibly for classification when size is the only other datum available. The converse does not seem to be true: although the frequency with which metacentric chromosomes such as 16s and 19–20s have third order coefficients considerably greater than fourth order ones is quite low, it is not negligible. As we are at pains to avoid misleadingly high confidence estimates being associated with wrong results, even in a small percentage of cases, it is unlikely that we could make any use of this fact.

A slightly more promising result was obtained from a linear regression analysis comparing the centromeric index with the magnitudes of the third, fourth, fifth, and sixth order coefficients and certain of their ratios. The most important term in the regression equation was the ratio of the RMS value of the third and fifth to that of the fourth and sixth, and the relationship obtained is summarized in table 2.

The trigonometric coefficients can also be used to determine axis orientation. In the case of large, elongated chromosomes the second order term plays an important role as a 'stretch factor'; dominant third order or dominant fourth order terms are, as remarked above, usually associated with trefoils and quadrifoils. If the cosine and sine combination for each wave number be re-expressed as a cosine with a phase angle, thus:

$$a_n \cos nx + b_n \sin nx = c_n \cos (nx+\alpha),$$

then the orientation corresponding to these terms is $\alpha/n$.

Given a large second order term, or a clearly dominant term of order 3 or 4, the axis inclination seems to agree very well with the corresponding $\alpha$.

In introducing the paragraph we remarked that the original motive for seeking an expression in orthonormal functions for the $r, \theta$ figures was the need for a smoothing and curve fitting procedure which reflected overall shape characteristics. In all cases so far examined, the correspondence of the curve defined by the sixth order trigonometric polynomial with the digital figures, and in particular of the concavities in the centromere region with those observed by the eye in the digital outline, is excellent. If a centromere-defining valley or shoulder is present at all it corresponds to a local minimum of the curvature of the trigonometric figure, determined by analytic formula. However, the polar coordinate calculation of curvature involves evaluating the expression

$$\left\{2\left(\frac{dr}{d\theta}\right)^2 - r\left(\frac{d^2r}{d\theta^2}\right) + r^2\right\} \Big/ \left(r^2+\left(\frac{dr}{d\theta}\right)^2\right)^{\frac{3}{2}}$$

at each point and this is rather heavy going for the timing required. Nevertheless, the results indicate that the requisite information is available in the trigonometric polynomial, and it should certainly be possible to use the smooth analytic curve where the digital boundary is not quite good enough.

**Acknowledgements**

Much of the work described in this article was done in collaboration with other members of the MRC Clinical and Population Cytogenetics Unit, especially Miss Judith Hilditch, who in particular worked out the present form of the profile analysis scheme and was solely responsible for the skeletonizing algorithms mentioned. I am obliged to Miss Ruth Goldberg and Mr Peter Smith for various statistical calculations.

## REFERENCES

Attneave, F. & Arnoult, M. D. (1966) The quantitative study of shape and pattern perception. *Pattern Recognition* (ed. L. Uhr). New York: Wiley.

Evans, O. & Sweeney, O. (1969) *A method for computer analysis of chromosome photographs, a preliminary report.* Memo 58, Information Exchange Group 3.

Freeman, H. (1961) Techniques for the digital computer analysis of chain-encoded arbitrary plane curves. *Proc. Natl Electronics Conf.*, **17**, 421.

Gallus, G. & Montanaro, G.A. (1968) A problem in pattern recognition in the automatic analysis of chromosomes: locating the centromere. *Computers and Biomedical Research*, **2**, 187–97.

Gallus, G. & Neurath, P.W. (1970) Improved computer chromosome analysis incorporating pre-processing and boundary analysis (available from author).

Hilditch, C. Judith (1968) An application of graph theory in pattern recognition. *Machine Intelligence 3*, pp. 325–47 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Hilditch, C. Judith (1969) Linear skeletons from square cupboards. *Machine Intelligence 4*, pp. 403–20 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Ledley, R.S. (1962) Visual pattern recognition by moment invariants. *IEEE Trans. Information Theory*, 179–87.

Mendelsohn, M.L., Hungerford, M.L., Mayall, B.H., Perry, B.H., Conway, T.J. & Prewitt, J.M.S. (1969) Computer-oriented analysis of human chromosomes II: Integrated optical density as a single parameter for karyotype analysis. *Ann N Y Acad Sci.*, **157**, 376–92.

Regoliosi, G. (1969) Personal communication. Instituto di Biometria e Statistica Medica.

Rutovitz, D. (1968) Data structures for operations on digital images. *Pattern Recognition.* Washington, D.C.: Thompson Book Company.

Rutovitz, D., Cameron, J., Farrow, A.S.J., Goldberg, Ruth, Green, D.K. & Hilditch, C. Judith (1970) Instrumentation and organization for chromosome measurement and karyotype analysis, *Human Population Cytogenetics Pfizer Medical Monographs 5* (eds Jacobs, Patricia J., Price, W.H. & Law, Pamela). Edinburgh: Edinburgh University Press.

Stone, P.S., Littlepage, J.L. & Clegg, B.R. (1967) *Second Report on the Chromosome Scanning Problem at the Lawrence Radiation Laboratory.* Proc. Seminar of Society of Photo-optical Engineers. New York.

# 25

# ESOTerIC II—an Approach to Practical Voice Control: Progress Report 69

David R. Hill
and
Ernest B. Wacker
University of Calgary

**Abstract**

It is appropriate, in a Workshop session, to discuss work done and in progress, and to avoid plans or speculation. Such a Workshop does, however, afford an opportunity to interpret earlier plans and speculation in terms of physical objects and current activities. The emphasis in this report is, therefore, twofold – description and interpretation. The first part is more concerned with interpretation of the ESOTERIC philosophy – what is the problem toward which the ESOTERIC philosophy is directed, and how has this emphasis arisen. The second part is more concerned with a detailed description of a real machine, ESOTERIC II, that is now implemented, and about to be tested. The justification for building hardware, when simulation might seem more appropriate, is also examined.

## ORIGINS OF THE WORK

The ESOTERIC project is a continuation of automatic speech recognition research that originally started with attempts to apply the STELLA learning machine to problems of automatic speech recognition. At the time (1963) work on STELLA, a general problem solver, or versatile learning machine (Andreae 1964, 1969) was under pressure to move into 'applications areas'. STELLA was under construction at Standard Telecommunication Laboratories Ltd., Harlow, UK, where, because of a strong corporate interest in automatic speech recognition as a means of voice dialling and for other purposes, it was suggested that automatic speech recognition would be a suitable 'applications area' for the work on STELLA. The main reason behind the suggestion was the fact that it had, by then, become apparent that speech recognition, far from being easy, was so difficult that no real solution was in sight, though a number of attempts had raised hopes of

ultimate success in solving the problems. At a time when the idea of a learning machine had gained some acceptance, it seemed reasonable to try one out in a situation in which most of the problems seemed to arise from incomplete advance information, and a need for adaptation to different speakers. Undoubtedly this attitude arose, at least in part, from the exaggerated claims that had been made for Perceptrons, a particular class of adaptive machines. STELLA differed from other adaptive machines chiefly in that she had 'actions' and could experiment with her environment in order to control it, both in the sense of direct actions, and in the sense of gathering information to assist in the control task. Also, Uhr and Vossler (1961) had enjoyed some success in applying learning to problems of pattern recognition, which was encouraging. A digital systems project had started the previous year using a zero-crossing approach to automatic speech recognition, following the success with this approach by Sakai and others (1960, 1961 and 1963). This work aimed at a non-adaptive recognizer, and the work always constituted a separate project, though some adaptive techniques were used later.

The very first scheme for adaptive automatic speech recognition was complex and over-ambitious. It involved 'analysis-by-synthesis' in that the incoming speech was to be recorded, and replayed into the system repeatedly until an acceptable match had been generated. The scheme, in simplified form, is shown as figure 1. At any given stage of the analysis, the input speech was to be compared with the output of a speech synthesizer such as PAT (Antony and Lawrence 1962). The difference signal was to be fed to STELLA, forming her I-pattern, and STELLA's task was to be that of producing actions (to select and modify rules for speech synthesis) in order to reduce this difference. When the difference became less than a specified tolerance, a reward signal was to be given to STELLA, forming a parameter

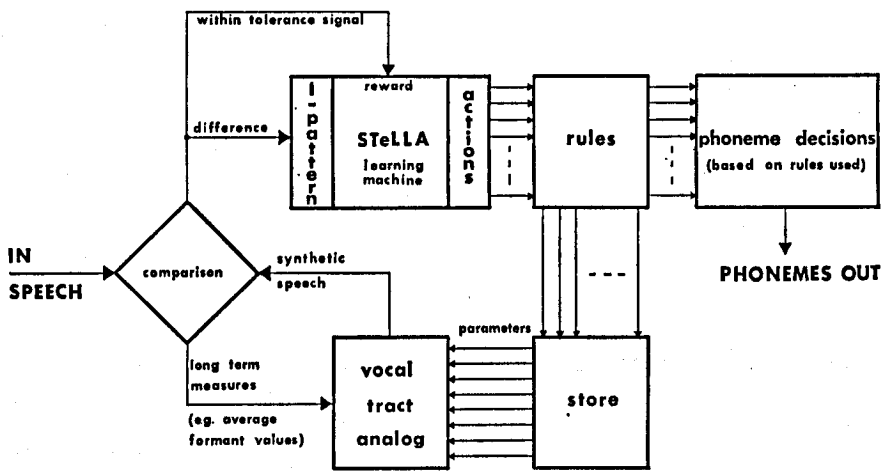



Figure 1. Simplified schematic of 'analysis-by-synthesis' ASR proposal (1963)

in her learning process. It was envisaged that the reward signal might, at first, be given by a human operator, though in the diagram it is shown resulting from the comparison process. On the basis of the rules selected, it was supposed that phoneme decisions could be made, since the rules required for synthesis would constitute a formal description of the input speech – the 'one' resulting from the 'many-to-one' transformation imposed by the learning machine. Provision was also to be made for an operator to listen to the synthetic speech, though this is not shown.

It is hardly necessary to state that this scheme never advanced beyond the study phase. It did, however, make an attempt to tackle the three problems of automatic speech recognition which seemed the greatest hindrance to further progress. These problems were: *feature selection* – what was the nature of the cues in the acoustic waveform that allowed humans to recognize utterances? *time normalization* – how could the elastic, non-linear time scale of speech events be standardized in order to allow comparisons to be made between what might be different versions of the same word? and *segmentation* – which Stevens, of MIT, has suggested is 'the problem that you can't' (segment speech into any kind of building-blocks, such as phonemes, with any degree of reliability). Feature selection was taken care of, because the description of the input speech was in terms of the rules needed for synthesis, and such rules could be found (Holmes *et al.* 1965). Time normalization was taken care of because STELLA could learn the necessary actions needed to reduce a difference due only to a time-scale mismatch. Segmentation was taken care of since comparison supposedly did not rely on segmentation.

The scheme also made three facts very clear. First, that our knowledge of learning machines and speech production was not up to the exacting requirements of such a scheme; secondly, that one could not begin to build an adaptive speech recognizer, without more experience with non-adaptive approaches; and thirdly, undoubtedly the most important lesson of all, that however one tried to hide the process, at some stage, in any recognition scheme, one would have to analyse the input speech *explicitly* in terms of just those attributes that a more direct analysis and classification scheme might require. In an analysis-by-synthesis scheme the process is hidden in the comparator, for difference signals are meaningless if they do not refer to *relevant* differences – i.e., a mismatch of just those important features needed for an analysis and classification scheme. G. A. Miller, of Harvard, has referred to this as the 'diamond problem', since comparators are frequently represented by a diamond in block schematic diagrams.

The next phase of the work was accordingly directed at the problem of feature selection alone, with little provision for adaption, but a very strong philosophy of 'keeping options open' and avoiding self-limiting lines of research. Work was started on a 'Speech processor operating on features', which drew heavily on published knowledge about cues known to be important in speech perception, particularly that resulting from work at the

Haskins Laboratory in New York (for example, Liberman 1957, O'Connor *et al.* 1957). Attention was also paid to experiments on the neuro-physiology of animal sensory systems (for example, Hubel and Wiesel 1962, Whitfield and Evans 1965). Because of the useful simplification, and under the influence of the 'distinctive features' proposed by Jakobson, Fant, and Halle (1969), analysis in terms of *binary* features was proposed. It is hardly necessary to emphasize that the relevant literature was widely scattered and one result of this phase was the publication of an abstracted bibliography of papers (Hill 1966). A more ambitious and up-to-date version, incorporating a descriptor index, is currently being generated.

The speech processor has been described elsewhere (Hill 1966a, 1967). It turned out to have a tendency towards three-state, or ternary, features as a result of identifying binary oppositions, but allowing a 'don't know' judgement for cases where the evidence was not sufficiently convincing either way. Since one object was the gathering of data for the design of further features, some of the measures were not at all binary – for example, the interval timer used an exponentially decreasing counting rate, displayed, or output, as a ring counter state. Since there was no active process, to distort and vary the time scale as required, it was necessary to specify some other means of dealing with the effect of time-scale variations. At the same time it was necessary to keep the number of bits of information handled down to a minimum, to avoid getting lost in a welter of irrelevant detail, and also for reasons of economy. It is worth noting at this point that telephone quality speech requires about 40,000 bits/second to specify it, yet – even neglecting the very real constraints which exist – 40 phonemes, at an average rate of, say, 10 per second, require only about 50 bits/second.

A sampling technique which amounted to a method of segmentation was implemented. Pulses were generated from two sources: first, from a high-frequency-energy detector, with the intention of marking the beginning and end of strong fricatives; secondly, from a 'word-shape' signal – based on any sudden change in the rate of increase or decrease of the signal. The 'word-shape' signal was a short-term mean power measure derived from the raw speech signal by rectification and smoothing. By applying a threshold to the second derivative of this measure four pulse types were produced – beginning of rise, end of rise, beginning of fall, and end of fall. These six pulse types, four relating to total energy, and two to fricative energy, could be used in any mixture to control feature sampling. During the interval between any two selected pulses, any feature detected would set the appropriate bit in a storage matrix, till ultimately the word ended. Thus each column of the matrix would represent the features detected during a given sample interval, and successive columns represented successive sample intervals during the word, the time scale being set by the rate of utterance. Other schemes have used somewhat similar measures (for example, Olson and Belar 1964, Gold 1966, Reddy 1968). Table 1 summarizes the features that were proposed. All except the

| Name of group | Composition of group |
|---|---|
| Silence | Present/absent |
| Relative energy level | Higher-than/about-equal-to/lower-than the energy in the last interval |
| Voicing | Present/absent |
| Duration | Non-linear (see text) |
| High frequency | Present/absent |
| Fricative quality | 5 selected patterns derived from H.F. spectral slope features present/absent |
| Vowel quality | 10 selected patterns derived from L.F. spectral slope features present/absent |
| Transitions | The frequency spectrum was divided into 5 regions (see text). In each region the presence/absence of peaks whose mean frequency was rising, and of peaks whose mean frequency was falling could be noted (see text). |

Table 1

transition features were implemented before the scheme was dropped in favour of computer analysis.

The spectral slope features were the result of comparing adjacent filter outputs. As a result of such a comparison the judgements POSITIVE SLOPE present/absent and NEGATIVE SLOPE present/absent were made – two binary judgements. These two features, for a given pair of filters, were mutually exclusive, so there were really three possible slopes, positive, negative, or (within certain limits) no significant slope. A 'null' slope of 6–12 dB/octave (negative) was assumed. These slope features were used as a basis for spectral quality judgements, and also for transition detection. Since formant transitions were believed important cues, though attempts to track formants proved fraught with difficulty, the frequency spectrum was divided into five regions: three were the exclusive domain of the three lowest formants, and

two represented ambiguous regions. It was intended that slope features moving up or down in frequency within each region should set the transition features RISING or FALLING, appropriately, for each region, with no attempt to assign formant labels.

The whole feature matrix resulting from the input of an utterance to the speech processor was output as a matrix of lights, a crude, but effective, form of man-machine interaction. Indeed, it turned out that the main benefit resulting from the construction of the processor was the ability to observe this matrix for many utterances. The output defied formal analysis up to the time that the processor was discontinued.

Building the processor undoubtedly provided a useful focus for gathering information concerning the speech process. It also formed a welcome umbrella during occasional management storms. Its principal value was however, as just noted, in providing an effective means for experimental interaction with large amounts of speech data in digestible form. It was in subsequent attempts to provide a formal means of analysing the output that the ESOTeric project had its direct origin.

## THE ESOTerIC PROJECT

### Introduction

There were, in reality, three different aspects to the formal analysis of the output from the processor. First, such an analysis was intended to complete the requirements of a formal description of the original input, and only on the basis of such a formal description could decisions be taken, and hence the merit or demerit of the features themselves be evaluated. It became all too obvious that any definitive test of speech processing equipment, not open to conventional testing techniques such as 'listener preference' or 'intelligibility' testing, nor yet giving a categorical output which could be judged right or wrong, was an equivocal, lengthy process, of doubtful value. The experimenter could, perhaps, see patterns, in the light display that were characteristic of particular words, but to test the feature set adequately it was essential that a machine algorithm be developed for detecting such similarities. By way of illustration, consider the fact that it has been shown that people can be trained to recognize speech on the basis of the sound spectrograph (Potter *et al.* 1948) yet attempts to build machines to emulate the performance have failed. This problem we may call the TESTING problem, and amounts to the fact that it is difficult to test any part of a recognition scheme unless the whole recognition process is automated, if for no other reason than the difficulty of handling the otherwise vast amounts of data.

The second aspect of the need to formalize the output analysis was the time aspect. It seemed impossible to achieve perfect, or even adequate segmentation on the basis of any criteria then being used. Although the segmentation circuits worked well, indeed better than might have been hoped, there was always some segmentation where it was undesirable, and

yet other segments were not separated. In any case, since thresholds were involved, segmentation was something of a compromise at the very best. Even when segmentation seemed to be the same for two similar utterances, the features did not always turn up in the corresponding segments. In particular, feature A might always be followed by feature C, and feature E by feature B, in most different versions of the same word, yet occurrences of features E and C might sometimes fall in the same segment, and sometimes in different segments, in a highly variable and unpredictable manner. More subtle sequential constraints appeared to be present. If one could describe these sequential properties without reference to (unreliable) segment boundaries, one would have caught the essential similarities in the output between different words in terms of the (supposedly) important features only. Not only would this allow the importance of the features under test to be judged (first aspect), but such an approach could form the basis of decision in any machine, using the best binary features that could be found. We may call this the SEQUENCE DETECTION problem.

The third aspect of the analysis was the fugitive nature of many of the features. The machine never seemed to be able to make up its mind about the presence of anything. This applied at the level of spectral slope features just as much as at the level of output, for a secondary display of the spectral slope features was also provided. This effect tended to confuse the output, and was an obstacle to any useful analysis. There was a clear need to embody hysteresis in the feature detection circuits. We may call this the HYSTERESIS problem.

In all of the above, and in what follows, it is important to remember that for any machine that is trying to classify patterns as humans classify them, the only test that is valid for any part of the machine is 'does it, in some way, improve the measure of agreement between machine judgements and human judgements?'. There is only an indirect link between the judgements of either human, or machine, and the acoustic waveform. Indeed, it is generally agreed that the acoustic waveform simply does not contain all the information required for 'perfect' recognition. Thus the question to be answered in judging the importance of features, or other parts of the machine system, relates to the way in which they indicate similarity between utterances which humans judge to be similar.

### Why build hardware?

At this stage, with a computer system under way, research might well have turned over entirely to computer simulation. That this did not happen is due to a variety of reasons, many of which may be summarized as answers to the question, 'Why build hardware?'. Some workers, for example Reddy (1967), have gone so far as to suggest that one should *not* build hardware prior to implementing a working simulation. In his paper, however, Reddy follows his assertion that the computer is less restrictive than hardware with

the observation that a different approach to filtering is required, and that brute force techniques such as pair-wise comparison, or a 1-out-of-*n* strategy, are too time consuming. Denes (1965) seems to support the view, which to a large extent is surely correct. However, there are some good arguments for building hardware, and the real answer lies, as always, in compromise. Even Reddy uses an A–D converter which (with a zero-crossing approach) is not essential. The answers to the question are, briefly:

(a) Research cost: Hardware studies may cost less than computer simulation.

(b) Communicability: Logic and circuit diagrams for a machine form a convenient and unambiguous medium for communication with colleagues.

(c) Feasibility: Since the end product will very likely be a special purpose piece of hardware, the early use of hardware avoids the problem of ending up with a computer solution which cannot be implemented.

(d) Real-time: The use of a serial machine for information processing of this type frequently results in many times real-time operation, even with a dedicated machine.

(e) Credibility: The computer age has not been with us long enough for managers, investors, and government agencies to accept computer solutions as readily as hardware solutions. The impact of a small machine is always greater than that of a large machine if they do the same job.

(f) Patentability: Perhaps one of the most frequently undeclared reasons. This may be the most important reason of all. Obtaining protection for computer programs is far more difficult than obtaining protection for hardware, even if it is possible.

(g) Testability: Whether inevitable or not, it certainly seems to be true that speech recognition programs only run on the home system. Clearly cost and other factors are involved, but to send a piece of hardware to other people for trials is about as acid a test of the ideas and procedures involved as one can devise.

It is perhaps as well to provide some support, or additional explanation with respect to the above points. As far as research cost is concerned, the point is well illustrated by experience in producing spectrographs. Using the Manchester University Atlas computer gave a cost, per spectrograph, in dollars: using a spectrograph machine gave a cost in fractions of a dollar. A compromise solution using a PDP–8 computer in conjunction with special purpose hardware gave a cost in pence (Hill 1969a). Of course, the Atlas allowed extreme flexibility, in theory. In practice, of course, one gets just as constrained by existing software as existing hardware. The MACRAN time series analysis programs cost thousands of dollars; and in-house software is often left unchanged, once working, because of the real effort needed for rewriting. There is also another aspect to cost – that of turnaround time.

This is using the 'computer only' man's argument against him. Real-time interaction with data aided by a powerful machine is fine. Interaction with a 24-hour turnaround program, or interaction using a system dedicated to the purpose only between the hours of 01.50 and 02.35 which may be delayed for hours in practice, is hardly useful interaction. A small computer, dedicated full time, offers the same advantages as special hardware, from this point of view, but is frequently too small to perform the complete recognition task. Reddy's program, for example, is reported as requiring 75,000 words, each of 36 bits.

Regarding communicability, many computer scientists avoid special languages in their work because widely accepted languages, such as ALGOL, are a valuable means of communicating their work to others. Some even re-write their algorithms prior to publication, if they have not used such generally accepted languages. Logic and circuit diagrams, properly executed, are equally universal and unambiguous.

It may be objected that the feasibility argument depends on a false assumption, and that the final version of a speech recognizer may still reside in core storage. The core requirements and CPU time required for some programs to date (for example, Reddy and Vicens 1968) suggest that even with time-sharing, the use of a general purpose computer for practical applications is not economical. If the object of speech recognition is to access a multi-access computer system, the abilities of the computer are likely to be overcommitted without performing recognition as well. There almost certainly has to be a reduction in the number of bits being handled before the speech input gets near the computer. This cry is echoed throughout the literature. A chief advantage of earlier 'stand-alone' devices such as AUDREY (Dudley *et al.* 1958), SHOEBOX (Dersch 1961), and DAWID I (Tillman *et al.* 1965) was that they offered an economical possibility of voice control for other machines in limited situations. It will be a long time, for example, before everyone can afford to use a multi-access computer for voice control of, say, their telephone answering machines. Even when that day comes, if special purpose hardware does a cheaper, or better job, it will be used instead.

The points regarding real-time and credibility are self-explanatory. On the subject of patentability, one should note that this is of greatest interest to commercial ventures in the field. Commercial ventures usually exist to make a profit out of their activities. Thus they find hardware appealing, as it allows protection to be obtained, and enforced in the courts under well-established patent law. Clearly protection is of little use if it cannot be enforced, and proving the formal equivalence of two computer programs is an interesting unsolved problem!

Finally, there is the point about testability. Many of the situations in which tests of speech recognition equipment are required exclude the use of a large computing system with, say, A–D conversion facilities. Some situations are those in which available computing equipment is already committed to

other processing; in others, say in limited audio-visual systems for teaching, no such computer may be available. These are important applications, and we have a great deal to learn concerning human factors in such situations (Hill 1969b). To be able to send a stand-alone system away from the highly favourable atmosphere of home territory facilitates searching tests of the procedures and equipment, as well as allowing attention to focus on human factor aspects of such trials. Field trials in selected applications can most easily be carried out using a stand-alone device: also, the travelling salesman has never found it easy to travel with more than his sample case, even though the samples may not represent his total range. If we can demonstrate that even our simplest techniques work, and have viable applications, we shall be able to progress on the two fronts of human factors and credibility. A stand-alone device, which can be used away from the magical influence of the designer, gives an acid test of its credibility, 'usability', and inbuilt assumptions.

### The work

The ESOTERIC project should, in reality, be considered as a whole, consisting in part of computer experiments, both simulation and basic research, and in part of hardware design and testing. This paper is mainly concerned with the latter subject, however, together with the underlying philosophy, and with the background. As noted above, the ESOTERIC project arose as a direct result of attempts to formalize the analysis of the output of the original speech processor, as a pre-requisite to taking decisions. Three aspects of this analysis problem were noted; the TESTING problem, which requires a complete scheme to be set up in order to test parts adequately; the SEQUENCE DETECTION problem, which requires a means of making sequential characteristics of a time pattern explicitly available; and the HYSTERESIS problem, which requires some 'stickiness' in the decisions that are implicit in the imposition of thresholds to produce binary features. It is towards solution of these three aspects in particular that the ESOTERIC project is directed. Once these are taken care of, the search for better features may continue.

*Hysteresis.* It is convenient to consider the three problems in reverse order, taking the HYSTERESIS problem first. It seems easy to solve, simply by incorporating amplitude hysteresis on the trigger thresholds of the feature detectors. Thus two levels are involved, a top trigger level, above which the output signal comes on, and a bottom level, below which the output signal goes off. Once the signal has appeared at the output, the analog input has to decrease appreciably before the output switches off again. A similar argument applies for the signal going off. Such a device exhibits a tendency to 'stick to its decisions'. Our experience was, however, that this alone was not sufficient hysteresis. Momentary short changes in the input level, quite large compared to the normal input, could occur due to noise and other factors, and cause momentary presence or absence of the output, at times when the human

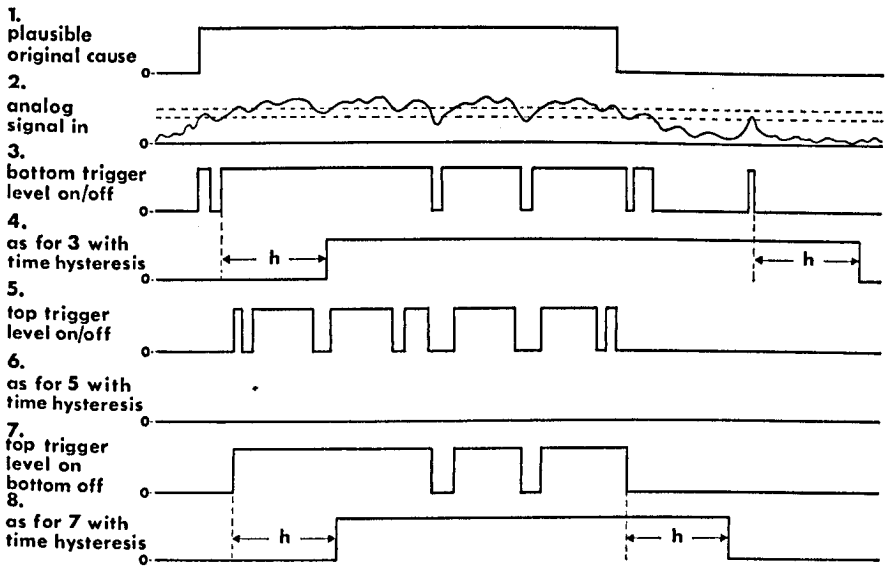



Figure 2. Illustration of the combined action of time and amplitude hysteresis

could say such outputs were spurious. In any case, some minimum duration criterion was required for any given features, to determine whether a given signal was significant or not. This second hysteresis attribute is termed 'time hysteresis', and acts to prevent detection of a feature until the binary output has been continuously present for a selected time, and then acts so as to inhibit detection of absence of the feature, until it has been continuously absent for a similar period of time. The combined effect of time and amplitude hysteresis is illustrated in figure 2. The application of hysteresis in this manner is a 'weight of evidence' technique. Mackay (1962), in his illuminating paper on theoretical models of space perception, suggests that perception is on the basis of a grudgingly modified 'null hypothesis'. The null hypothesis is not abandoned until the weight of evidence is against it, rather than merely not supporting it. The same point has been made by Arbib (1969) who suggests that, like the movie cartoon artist, perception assumes that everything remains unchanged, unless such assumptions are unsupportable. This strategy economizes effort, processing power being concentrated on the information-bearing changes. Without the ability to set up and test such minimum hypotheses, the machine's (or person's) ability to structure its input, a necessary precursor to making good decisions, is seriously handicapped. At a practical level, it is necessary in order that the processor may make a reasonable representation of the input, and in order to produce data suitable for subsequent stages in the processing. The difference between signals before and after hysteresis is quite striking.

*Sequence detection.* The problem of SEQUENCE DETECTION is of central concern in the ESOTERIC project. The object of sequence detection is to capture the elusive similarities between the parallel time series outputs of a number of binary feature extractors, for utterances of the same word, while being undisturbed by relatively gross time displacement of individual features. Instead of a 'sequence detector' we should, perhaps, more accurately refer to a 'sequence describer', since this is the function of the sequence detection part of the ESOTERIC scheme, to describe the sequence. Thus the sequence detector is the implementation of a descriptive grammar, and as such comprises primitives, and relationships between primitives. In a time series, the simplest relationship is that of precedence – which of two events occurred first. As they stand, the outputs of the binary feature extractors are not events, they are extended in time. This means that for various cases of overlap, a feature which begins first, may or may not end first, and vice versa. The real event is the beginning, or end, of a feature. This, incidentally, is in accord with our previous observation that perception depends on changes, and certainly one may be unaware of a noise, until it stops. There is one remaining problem. The ending of a feature after a short presence may be significantly different to the ending of the same feature after a longer period. As an example, consider the presence of friction due to an alveolar constriction. If this ends after less than 50 milliseconds, it is very likely the release of a /*t*/, whereas if it ends in more than 100 milliseconds, it is very likely an /*s*/. Such differences seem to be fairly crude, and perhaps two duration categories suffice for English. The events for which precedence is to be determined, then, are the beginning of a binary feature, and its ending in one or more duration categories.

The input to a sequence detector thus comprises a number of lines on which pulses may occur marking events, as above. We may call these *primitive acoustic events*, or PAEs. The output of the sequence detector consists of a number of lines carrying similar pulses, and again marking events. These events may simply be a repetition of the primitive events, but many of them will mark the detection of particular precedence relations between particular primitives. We may call these latter events *compound acoustic events*, or CAEs. A given output from the sequence detector tells something unique concerning the original input, and tells it without reliance on the occurrence of other outputs. The outputs should express all the relevant information about the important events in the input, and the order in which the events occurred, explicitly. In a simple scheme, such as is given as an example in figure 3, the outputs could set the store elements in a 'bit pattern' to be used for decision.

The notion of 'sequence breaker' arises from the question 'When is a sequence not a sequence?' to which the best answer is that prohibited events must not occur in the middle of a sequence. In figure 3 it has been assumed that any event is prohibited in any sequence of which it is not a part. Prohibited

events break the sequence, and are therefore called 'sequence breakers'. Notice that the second time PAE 'E' occurs, the CAE '(B(A(CE)))' is detected, but, although the sequence *b–a–c–e* apparently occurs a second time, it is not detected again. This is because the CAE '(E(BA))' is detected first, breaking the sequence.

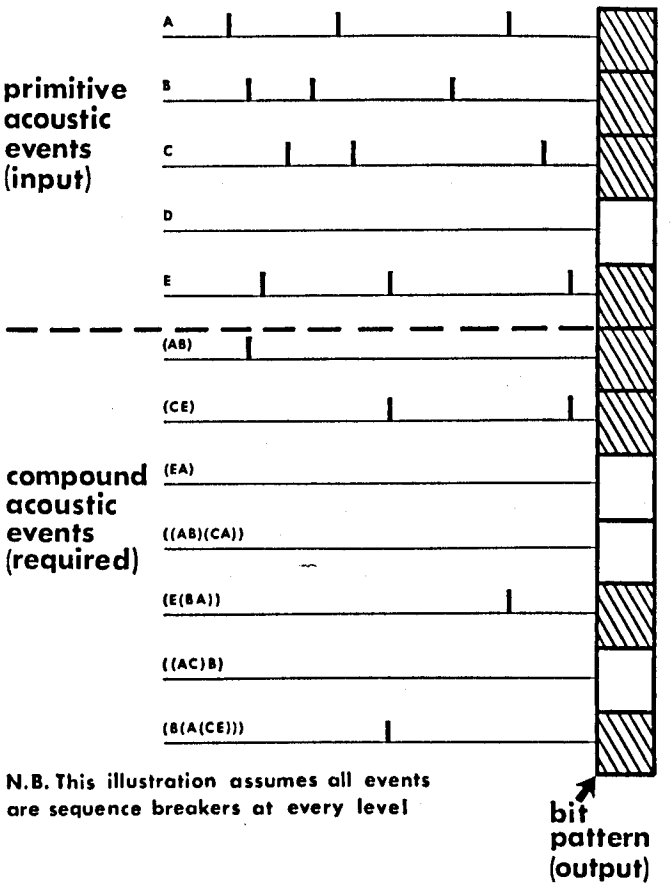



Figure 3. Static illustration of sequence detection

The basic element of the sequence detector is a precedence-sensitive switch which examines two inputs, say '*i*' and '*j*', and issues an output pulse on an output '*ij*' if '*i*' occurs followed by '*j*', or an output '*ji*' if the reverse is true, providing no input to a 'sequence breaker' input occurs in the middle. In hardware, this element is termed an *elementary sequence element*, or ESE. In the simulation the function is computed by a single recursive subroutine. The process of sequence detection, with more general reference to the problem of reducing a set of analog signals to a set of independent binary features, is covered in Hill (1969*c*). Using a number of ESEs in parallel allows a larger

number of primitives to be treated, whereas disposing them in serial arrangements allows more complicated CAEs to be detected. Clearly a compromise is required, since a sufficiently complex CAE might only occur in one word, and some primitives might occur in every word. Either extreme is informationally undesirable, though some redundancy is essential to combat noise.

Max Clowes (1969) has pointed out that relationships other than precedence may be important between primitives. This is a fair point, though it is not immediately apparent what other relationship to use, given the assumptions as to how speech may be treated that are implicit in the ESOTERIC philosophy, and which are made explicit below. If a descriptive grammar for spectrographs was developed, then there would be a much closer analogy to work on picture grammars, and many other relationships could be defined. Property attachment is a distinct possibility. Separating end events into different categories is a form of property attachment, since it represents a qualitative distinction between otherwise identical descriptions. It is, in the present scheme, considered preferable to include the step as part of the analysis, rather than as a qualification on subsequent description of sequence. Neville Moray (1965), at Sheffield University, has performed experiments with humans which suggest an ability to handle two types of information, content and order, and that the maximum information handling capacity is only realized when both forms are present. Whitfield and Evans (1965) in their study of the auditory cortex of the cat, conclude that it is the analysis of time structure, rather than stimulus frequency, that is the important property of the primary auditory cortex. Of course, some content analysis is carried out, at least in the cochlea, so the notion of an auditory pattern recognizer performing on the basis of content and order analysis does not conflict with what is known, by means of psycho-physics and neuro-physiology, about animal systems. We may consider the analyser to be concerned with the processing of content information, and the sequence detector to be concerned with the processing of order information.

Work on figure description languages has also influenced this aspect of the work, perhaps not to the extent it should have. Three papers of particular note were those of Marril and Bloom (1965), Clowes (1967), and Guzman (1967). A recent paper by Guzman (1968) on the recognition of three-dimensional bodies in terms of the regions (parts of planes) that form them, without recognizing the bodies as such, raises the interesting question as to what the analogous exercise would be for auditory pattern recognition. Presumably the recognition of sources in terms of the noises arising from them, without recognizing the sources or noises as such. This process would depend on clues such as pitch structure, for linking noises, in the same way that Guzman's program depends on clues such as T-joints, for linking regions. But this is the speculation we planned to avoid!

There is a rather important point to be made concerning sequence detection, a point concerning the intrinsic economy. The point will be illustrated

with reference to another approach to dealing with sequences in speech, typified by an RCA device (Martin *et al.* 1966). In such devices, phonemes, or phoneme-like entities, are detected during the processing stage so that the input to a decision taker consists of a string of such entities, in order, one at a time. Comparatively costly sequential logic, embodying gating and successive delays, for timing purposes, is then required for each word, since any given word will consist of a particular string of the phonemes, occurring in a certain order, with a certain relative timing. Of course, since there is the kind of variation noted above, more than one set of logic will be required for each word, for each word may have several variants, and an exact match of sequence, element by element, within certain time constraints, is required for such a 'sequence detector' to give an output. Thus several sets of costly sequential logic are required for each word in the vocabulary, and as the vocabulary increases in size, problems of discrimination arise. Selfridge and Neisser (1960) pointed out the very real problems of noise susceptibility suffered by such sequential decision-taking. In the ESOTERIC philosophy all operations are carried out with a view to parsimony and generality, at least that is the aim. Timing is carried out at the level of binary primitives. Sequence detection is carried out as a general operation, not specifically for each word, and since timing is not involved the logic is cheaper as well as less profuse. Furthermore, sequence detection is reduced, as far as it possibly can be, to a parallel process, avoiding the problems of sequential decision techniques. Finally the output is 'staticized'. A bit pattern is generated in which each bit is a piece of evidence about the input regardless of the presence or absence of any other bit. If these bits of evidence are not independent of each other, there may be duplication, which complicates the subsequent decision; but there are techniques for dealing with such dependencies, for example Samuel's 'signature table' approach (Samuel 1967) or Andreae's 'string correlator' (1965) both of which are essentially empirical approaches to grouping dependent features in logical combination to produce new, mutually independent features. Another approach would be to observe the features and discard those which duplicated other features to any great extent, but such a process is computationally very expensive indeed. However, the static bit pattern produced, as one form of output from the sequence detector in ESOTERIC, is amenable to decision methods that are not expensive in terms of hardware. A diode matrix allows correspondence matching, and a resistor matrix allows various weighted decision techniques to be implemented, both at a comparatively small cost per word.

The final point to be made about sequence detection, as embodied in the ESOTERIC philosophy, concerns the selection of those sequential characteristics, or CAES, that are to be detected. In the first machine built, there was only a very crude form of sequence detection (Hill 1969c) and no selection was involved, over and above that implicit in the intuitive selection of useful attributes of the analyser output. In ESOTERIC II, the combinations are

limited, so the numbers involved are small, and empirical selection is still feasible. It is clearly preferable, and in the long run essential, to employ a learning strategy. A grammar-based processor, such as the ESOTERIC SEQUENCE DETECTOR, is eminently suited to the application of learning strategies. At the risk of speculating a second time, we suggest that those CAES needed to discriminate words actually encountered could be constructed (on the basis of direct comparison at the level of primitives) and tried out. On the basis of performance criteria, such as actual discriminating power, and usage, a competitive situation could be set up in which only the most useful CAES, or those that were essential, would be kept. Such a scheme may be retrieved, to some extent, from the realms of speculation by referring to Uhr and Vossler's work (1961).

*Testability.* The third aspect of the problem arising from work on the speech processor was testability. The need, in testing a pattern recognition system, is to have a complete (if not final) system in order to test any part; partly because the only valid test of any part is its effect on machine judgements; and partly because testing a complete system constitutes the most efficient form of automatic data handling – an important requirement in data-rich situations such as speech recognition. The solution to this problem merely involves using a complete system, perhaps partly simulated and partly hardware, during research, and producing complete stand-alone machines, from time to time, which can be put out on field trial. A consequence of this is a requirement for a modular approach in either software, or hardware. A modular approach allows fundamental changes to be made to any part of the machine without forcing a redesign, or rewrite, of the other parts.

## Conclusion

The ESOTERIC philosophy may be summarized in the following assumptions:

(a) Speech may be analysed in terms of the presence/absence of important characteristics, at least to a first approximation. This is supported by neurophysiological experiments, though Spinelli (1967) suggests that not enough note has been taken of the fact that the response is proportional to the degree of similarity between the input stimulus and the optimum stimulus. The assumption is, however, qualitatively right.

(b) The information in speech, at the acoustical level, is coded partly in terms of content – as detected under (a) – but also partly in terms of order, so that means of handling order *explicitly* are required.

(c) The best test of any part, or modification to any part, of a pattern recognizer is whether or not it improves the agreement between the machine judgements and those of the original system to which the patterns were meaningful, The system is usually a 'standard' human being.

(d) Solutions to problems should be general. The techniques employed should not be *ad hoc* or self-limiting: for example, those used in ESOTERIC II

(an isolated utterance recognizer) should extend easily and naturally to true continuous speech.

(e) Hardware and software approaches are both vital, and should proceed together.

(f) A modular approach to either hardware or software is essential to facilitate modification, and to allow both approaches to interact in complete, hybrid systems, while avoiding problems due to lack of compatibility between the different sections.

(g) Solutions to problems should be parsimonious (Occam's razor). This leads not only to reduced complexity and cost, but to a deeper understanding of the processes involved.

(h) Segmentation and time-normalization are really two aspects of the same problem – resulting from failure to handle duration and sequence information adequately. Segmentation is only *necessary* at the level of meaning, which – for a machine – depends on the significance of the outputs. A phoneme recognizer is not necessarily desirable (to go on to recognize words still requires a further stage, to recognize the phoneme sequence, see above), and may be far more difficult than, say, a 'command' recognizer.

## ESOTerIC II – THE MACHINE

### Introduction

The stage has been set for a presentation, with the minimum of explanation, of hardware currently at the point of completion, which is the latest hardware embodiment of some aspects of the ESOTERIC philosophy. A general description of an earlier machine, ESOTERIC I, and of ESOTERIC II has appeared previously (Hill 1969c). In that paper, results from ESOTERIC I, and some discussion of the general principles involved, were presented. This section of this paper is concerned with the details of ESOTERIC II, as now constructed at the University of Calgary. There will be considerable reliance on diagrams of the actual logic of the machine. An attempt has been made to stick closely to the MIL-STD-806B symbols throughout.

One point must be re-emphasized, before launching into the description proper. That is the point about the features used in ESOTERIC II. The speech input is analysed into two frequency bands, low frequency and high frequency. The filter characteristics are shown as figure 4. Using logic, triggers (embodying hysteresis), an operational amplifier, and four comparators, the rectified and smoothed output from these two filters is transformed into four binary features. These features are HISS (high frequency predominating), HUMPH (low frequency predominating), BOTH (high and low frequency maintaining a predetermined balance), and GAP (absence of significant energy). These features are subsequently transformed into events (PAEs), which then constitute the machine's only description of the acoustic content of the input. That this analysis is inadequate for anything but the simplest of recognition goes without saying, but the ESOTERIC project is aimed at the

problem of how to handle the information resulting from binary feature detection, as noted above, and, at this stage at least, certainly not directly with feature selection. The project arose out of the difficulties experienced in dealing with the output from a far more complicated set of binary feature detectors. In fact it should be noted that the four features are actually mutually exclusive, and cannot, therefore, utilize the full flexibility of the sequence detection scheme. Some sequences are impossible, for example, if all events are sequence breakers, or, looking at things a different way, the action of the sequence detector is redundant in the sense that very real constraints, on the possible orderings of the primitives, exist.

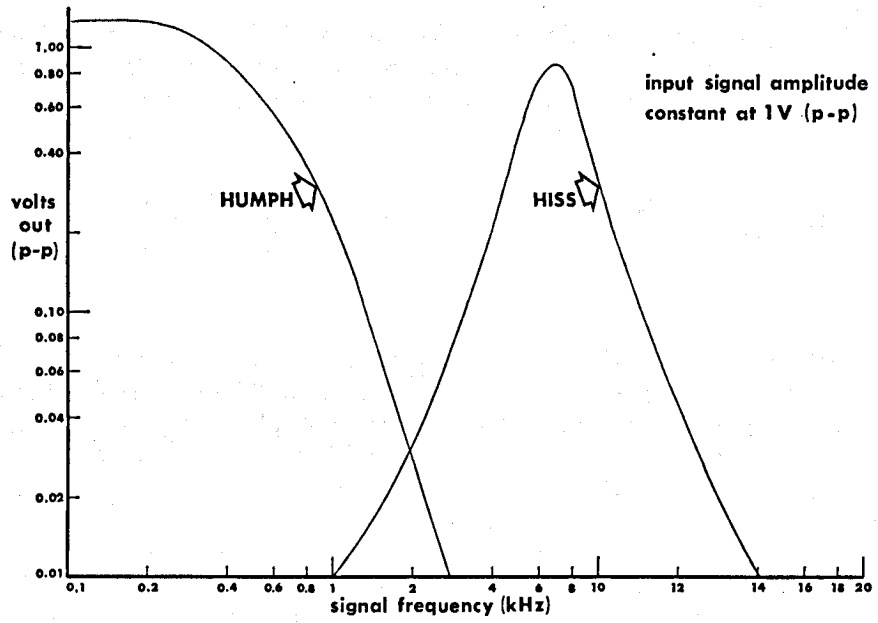



Figure 4. Filter characteristics for HISS and HUMPH

The crude analysis used in ESOTERIC II was chosen for two reasons: first because it allows a better comparison with ESOTERIC I; and secondly because the analysis is very simply performed and described, whilst providing signal forms typical of those obtained from any binary feature detector. The analysis also has some recognition potential despite its simplicity (Hill 1969c). It is, however, a real misunderstanding to believe that HISS and HUMPH have any central significance beyond convenience.

### Hysteresis

Amplitude hysteresis: figure 5 shows how amplitude hysteresis is obtained. For inputs which vary between ground and some positive value, the 6v input is connected to +6v, the input to the non-inverting inputs to the two comparators, and the reference to the inverting inputs. If the input is then

positive with respect to the level set by the 'on' potentiometer, it is also positive with respect to the 'off' level and both comparators give a logical '1' out. Thus gate 3 has two '0' inputs and gives a '1' out, while gate 4 gives '0' out. Thus flip-flop 7 is set, and the output of the amplitude hysteresis element (AHE) is '1', as required. If the input is less positive than the 'off' level, then flip-flop 7 is cleared, and the output of the AHE is '0'. If the input level lies between the 'on' level and the 'off' level both inputs to flip-flop 7 are '0' and it therefore remains in the immediately preceding state. A similar line of argument applies for the opposite polarity of input connections, and the device therefore provides amplitude hysteresis for either positive going signals, or negative going signals.

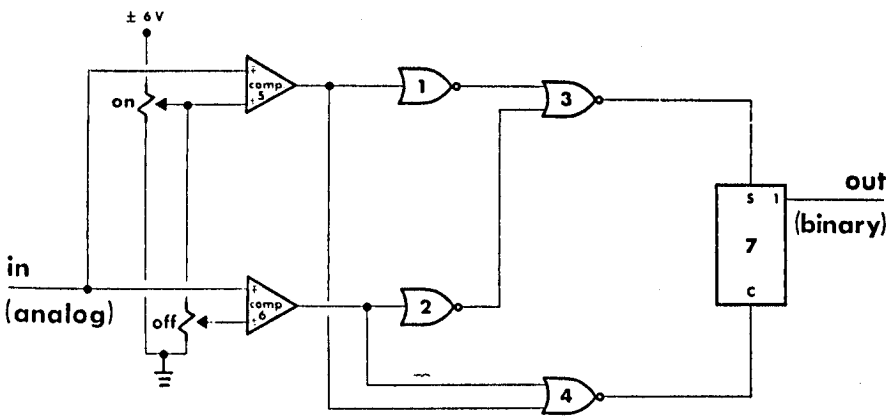



Figure 5. Amplitude hysteresis element—schematic

### Time hysteresis

Figure 6 illustrates the circuit of an integrating single-shot. It may be regarded as a monostable with zero recovery time, which would set to '1' and begin timing from every positive edge input, the output thus remaining at '1' until no further positive edge occurred within the period of the monostable. At this time, when the period expired, the output would return to its initial '0' state. Since such a monostable is not practical, two monostables having fast recovery times are used alternately and the outputs of these form the device output via a logical 'OR' function. The initial inverting gate is included since the J–K flip-flop driving the two monostables changes state on the negative-going edge of a signal.

Figure 7 illustrates the use of two integrating single-shots to provide time hysteresis. The purpose of the circuit is to provide a logic level '1' beginning when the input has been continuously present for a specified time, and continuing until the input has then been continuously absent for a similar period of time. Gates 1 and 2 provide the means of applying test pulses, to allow testing and calibration. Integrating single-shot 8 'fires' on the leading

edges of the input pulses. Thus both inputs to gate 5 are '0' only if the input is still present when the integrating single-shot has stopped firing. Consequently the output flip-flop is set only if the input is continuously present for the period of the integrating single-shot 8. The purpose of the delay is to avoid spurious setting of the output at the very first leading edge, due to the propagation delay through integrating single-shot 8, which could give a momentary condition of two '0' inputs to gate 5.

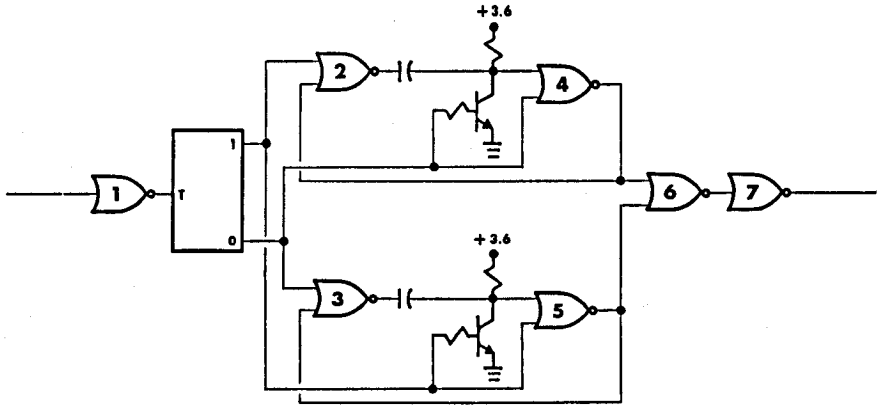



Figure 6. Integrating single-shot—schematic

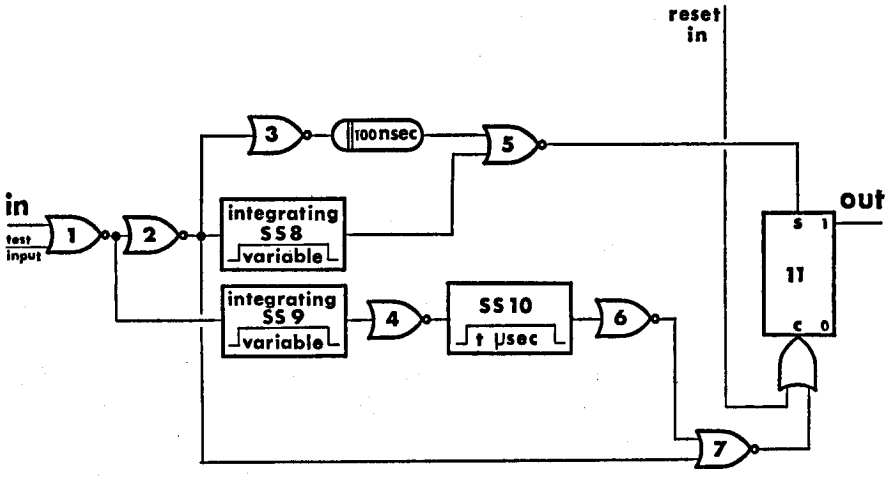



Figure 7. Time hysteresis elements—schematic

Due to the fact the inverted signal is applied to integrating single-shot 9, it will 'fire' at the trailing edges of the input pulses. Thus single-shot 10, which 'fires' at the time that integrating single-shot 9 ceases 'firing' will produce a pulse of nominal duration only when the input has been absent for the period of integrating single-shot 9. If the input is still absent at this time, gate 7 will have two '0' inputs, and will therefore clear the output

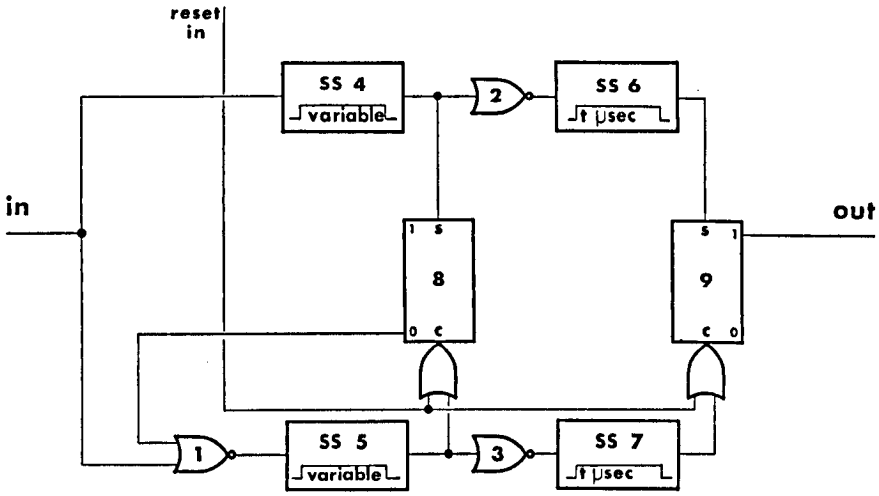



Figure 8. Delay normalization—schematic

flip-flop. The circuit thus implements time hysteresis as required. It is called a *time hysteresis element*, or THE.

Figure 8 illustrates a *delay normalization element*, or DNE. The purpose of the circuit is to adjust the total delay for all features to some standard, since the individual delays will depend on the THE settings. The present overall delay is chosen as a nominal 100 milliseconds, and for each feature, the delay in the DNE is set to (100 – THE-delay) milliseconds. Delay normalization is necessary, in general, to maintain the time correspondence of events prior to sequence detection. The action of the circuit is straight-forward. Single-shots 4 and 5 are set to the required delay. Until single-shot 4 has 'fired' and set flip-flop 8, action in the lower half of the circuit is inhibited by the input from flip-flop 8 to gate 1. Single-shot 4 'fires' when a signal starts (goes to '1'). At the end of its period, the input to gate 2 falls to '0', and a leading edge appears at the input to single-shot 6, 'firing' it, and setting the output flip-flop (9) to '1'. Thus the output rise is delayed by the required amount relative to the input rise. A similar line of reasoning applies to the lower part of the circuit, comprising gates 1 and 3, single-shots 5 and 7, and the clear input to the output flip-flop, with regard to the trailing edge of an input signal. The inhibit signal to gate 1 is simply to ensure that single-shot 5 is in a 'recovered' or ready state at the end of the input, since it would otherwise have been sitting with a '1' input up to the time the input appeared. With the delay times involved, the recovery time can be significant.

#### Event detection

Figure 9 depicts the event detector. It is called a *ternary* event detector since, for the reason given above under *sequence detection*, it produces three

outputs. The purpose of the *ternary event detector*, or TED, is to produce pulses of nominal duration, on three different lines, corresponding to the beginning of an input, the end of an input of short duration, and the end of an input of long duration. These pulses are the actual primitives for input to a subsequent sequence detector. Provision is made for test inputs. The action is obscured slightly due to a need to deal with inputs whose duration is close to the borderline between short and long. 'Close' may be defined in any sense required, but, clearly, within half the width of a nominal pulse either side of the border there is inevitable ambiguity.

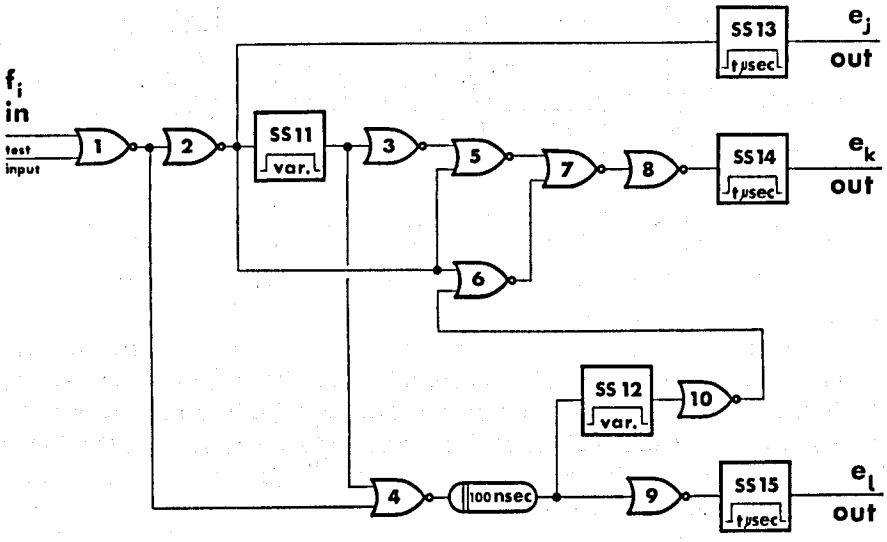



Figure 9. Ternary event detector—schematic

Single-shot 12, gate 10, and gates 6, 7, and 8 exist simply to deal with 'borderline cases'. Gates 7 and 8 provide an OR function on the outputs of gates 5 and 6, allowing a 'short' event pulse to be generated as a result either of gate 5 output going to '1' or gate 6 output going to '1'. Single-shot 11 is set to the longest duration that is to be considered 'short'. Single shot 12 is set to the interval length, following the 'definitely short' boundary, which is to be considered ambiguous. Thus gate 5 produces an output only if the input ends during the 'definitely short' period, since only then can it have two '0' inputs. Gate 6 produces an output if the input ends during the period of single-shot 12. Since single-shot 12 is triggered by the trailing edge from single-shot 11, which marks the end of the 'definitely short' period, and then only if the input is still present, gate 6 will only produce an output if the input continues into the period of ambiguity, and then ends before this period expires. Thus a 'short' event pulse is generated if the input ends short, or during the period of ambiguity. In the latter case, a 'long' event pulse will

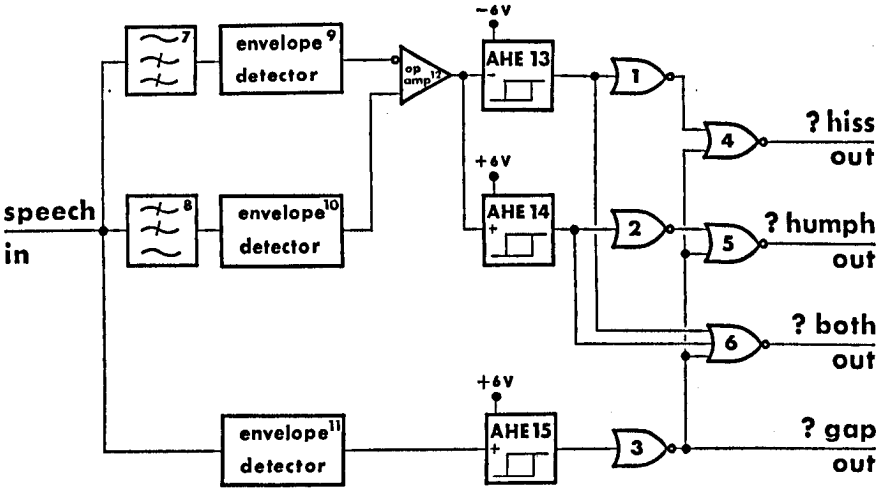



Figure 10. ESOTERIC II Analyser—Schematic (Part I) (characteristic detection and amplitude hysteresis)

also be generated (from single-shot 15) by the trailing edge from single-shot 11.

Clearly a 'beginning' event pulse is produced for the leading edge of the input, as a direct result of its firing single-shot 13. If the input exceeds the 'definitely short' period, and then runs on past the period of ambiguity, the only effect of the trailing edge will be to 'fire' single-shot 15, producing a 'long' event pulse. The 'beginning', 'short', and 'long' event pulses are thus produced (in the diagram as $e_j$, $e_k$, and $e_l$, respectively) as required. In the case of ambiguous duration, both 'short' and 'long' event pulses are produced.

We may now consider the analyser section of ESOTERIC II depicted in two parts, as figures 10 and 11. In figure 10, high pass and low pass filters 7 and 8 feed two envelope detectors, 9 and 10. Thus an analog signal from 9 represents the amount of HISS energy present, and that from 10 the amount of HUMPH energy. These two analog signals are combined by means of operational amplifier 12 to produce as output a measure of the balance between HISS energy and HUMPH energy. This balance signal varies between + and −6V, 0V representing the selected 'in balance' condition. Amplitude hysteresis elements 13 and 14, which follow, allow detection of 'possible HISS' and 'possible HUMPH' for deviation greater than the selected amount away from the balance condition, in either a positive or negative sense, as appropriate.

Of course, the balance measure is produced whatever the amplitude of the input signal, and one of the three states, possible HISS, possible HUMPH, or possible BOTH would always be indicated if provision were not made to detect possible GAP. The raw speech wave is fed to envelope detector 11, which feeds the GAP detecting AHE 15. A criterion for silence is thus set,

independently of the detection of other features. When silence is detected, output from other feature detectors is inhibited. The application of this inhibition is the sole function of gates 1, 2, 4, and 5. Gate 6, when not inhibited, generates the possible BOTH signal, if neither possible HISS, nor possible HUMPH is present. It may be seen that the output of part 1 of the analyser comprises four mutually exclusive possible features, ?HISS, ?HUMPH, ?BOTH, and ?GAP. They represent an intermediate stage in the processing.

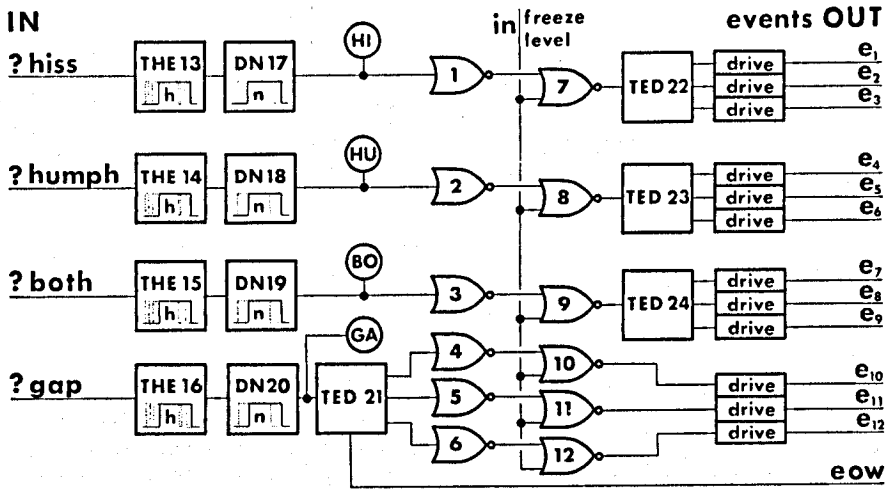



Figure 11. ESOTERIC II Analyser—Schematic (Part II) (time hysteresis, freeze, and event detection)

Figure 11 shows part 2 of the analyser which, by applying time hysteresis to each possible feature, rejects non-significant occurrences, and produces a 'clean' occurrence signal for the subsequent stages in the processing, as noted above (*hysteresis*). Delay normalization restores the time registration of the processed features.

Gates 1 to 12 are included to allow the analyser output to be inhibited completely, in a state that has been called 'frozen'. The machine is frozen when an *end of word* (EOW) decision is made, on the basis of the GAP feature. A GAP exceeding, say, 300 milliseconds is taken as a word-terminating symbol, though other arrangements for decision could be made which required neither a frozen state, nor isolated words.

It is also necessary, to complete the analysis, that the features be processed into events by means of event detectors (see above). These are provided, but, in the case of GAP, the event detector precedes the point at which inhibition occurs, since spurious 'end of gap' events would otherwise be generated. Finally drivers on each event output provide the drive necessary to operate subsequent circuits (purely a fan-out problem).

### The sequence detector

Figure 12 depicts the *elementary sequence element* (or ESE) which forms the basic element of the sequence detector. The device is symmetrical, so the operation of only half will be considered. Flip-flop 7 provides memory as to whether the $i$th event ($e_i$) has occurred since the last sequence breaker, reset, or $e_{ij}$ pulse. If the flip-flop is set, it means that the $i$th event has occurred, and should lead to an output saying event $i$ occurred followed by event $j$ (which constitutes the event $e_{ij}$) if event $j$ occurs. The flip-flop is cleared by a sequence-breaker pulse, a reset pulse, or an $e_{ij}$ output pulse, the connections to gate 3 providing for positive resetting, by inhibiting the set input during resetting. It is set by an $e_i$ input pulse occurring when the set input is uninhibited. There is a slight possibility of uncertain operation, which is acceptable.

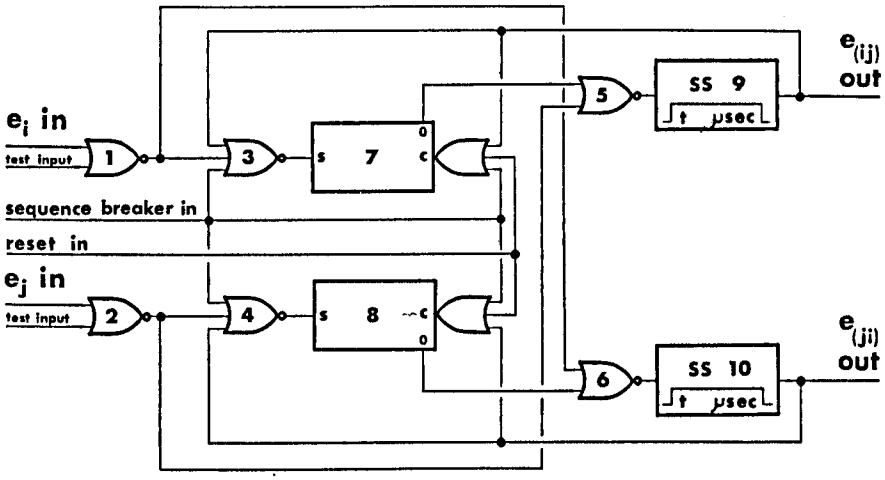



Figure 12. Elementary sequence element—schematic

If flip-flop 7 is set, and $e_j$ occurs, then gate 5 will have two '0' inputs, and will produce a '1' out, triggering single-shot 9, which generates an $e_{ij}$ pulse representing the occurrence of the $i$th event followed by the $j$th event.

In ESOTERIC II it is required to note simply the occurrence of certain of the various events that can be detected. The ESEs are, therefore, connected together by means of buffers (not shown) as necessary, and the pulses representing the occurrence of events of interest are fed to a set of output flip-flops which, together, comprise the output bit pattern (see below, *overall schematic*).

### The controller

Figure 13 shows the controller for ESOTERIC II. The controller determines which of four states the machine is in, on the basis of two signals, 'end of long GAP' and 'start of long GAP'. The latter signal is derived from the ternary

event detector for GAP, in fact from the 'period of ambiguity' single-shot for GAP (single-shot 12, figure 9). This provides an end of word indication, and will be referred to as the EOW signal.

Consider an initial condition, with both the start flip-flop and the control flip-flop in the reset state. Only gate 2, of the four gates 2, 3, 4, and 5, will have two '0' inputs, and the state 'ready' will therefore be indicated. As soon as the end of long GAP signal occurs, which it must for the start of a word, the start flip-flop is set, and the machine is now in the 'computing' state. So far all the electrical outputs remain inactive.

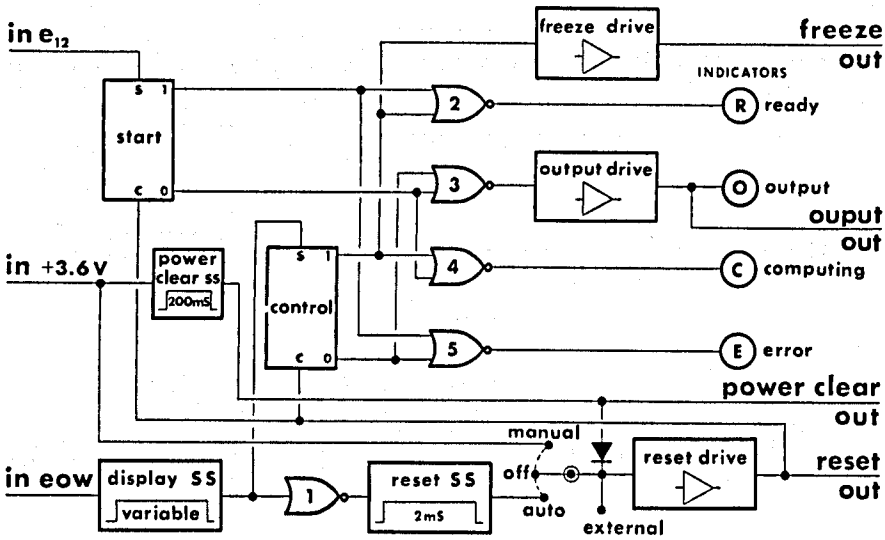



Figure 13. ESOTERIC II controller

When the EOW signal occurs, the display single-shot is triggered. The leading edge of the display single-shot sets the control flip-flop so that (assuming the start flip-flop was set, as suggested) gate 3 now has two '0' inputs, and the output drive and indicator are activated, and a 'freeze' signal appears on the appropriate output line. This freeze signal inhibits the output from the analyser section as noted above. If a new microphone input occurs to the machine, before the freeze signal is removed, of course, events will be missed. In particular, $e_{12}$, end of long GAP, will be missed. If this occurs, the machine will still be in the ready state when the EOW signal occurs, with the start flip-flop clear. Thus gate 5 would have two '0' inputs, and an error would be indicated. Also, although the freeze signal would still be produced (since the control flip-flop would be set by the EOW signal) no output drive would be generated, as gate 3 would be inhibited by the state of the start flip-flop.

At the end of the display period, the reset single-shot is triggered, so that the reset signal simultaneously returns the machine to the ready state, and

clears the memory. Two final points should be mentioned. A power-clear signal is provided, as a result of power coming up on the +3.6 v line. Besides providing an extended reset signal, the power-clear is taken directly to the memory elements of the GAP delay normalizer on the assumption that there is silence when the machine is first switched on. These elements are not normally reset. Secondly, a switch is provided to allow the reset mechanism to be disconnected, or the reset signal to be provided manually. If the reset switch is in the OFF position, external reset signals may be introduced via the external connection.

### Overall schematic

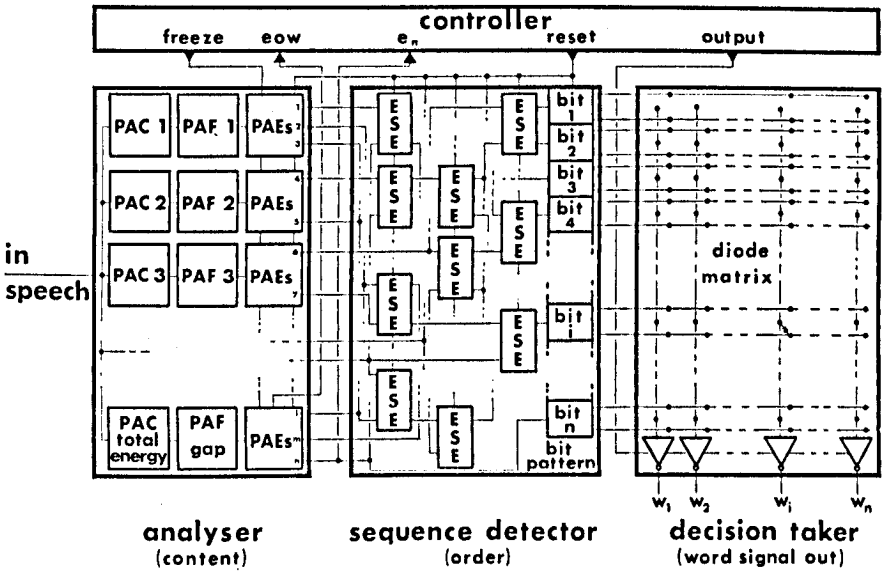



Figure 14. ESOTERIC II—overall schematic

Figure 14 illustrates the overall block schematic of ESOTERIC II. Speech enters the analyser where, by detecting characteristics, and transforming them to events, outputs are produced which represent the significant content of the input. As a result of the process of sequence detection, which analyses the sequence and thereby represents order information, a bit pattern is produced in which both content and order are represented explicitly. A given bit in the pattern is regarded as evidence to be used as a basis for decision, and the information contained in it does not depend on any relationship to other bits. The value of the information, however, will be reduced if the particular bit does not occur independently of the other bits. (Thus if every bit gave the same information, we could dispense with all but one of them.)

### The decision-taker

In figure 14 a very simple decision-taker is illustrated. The bit pattern is regarded as a code, and a diode matrix allows the occurrence of given

patterns to stimulate given outputs. Each different form of a word requires a column in the matrix, except where particular features may be ignored entirely. This is the type of decision used in ESOTERIC I, and also in ESOTERIC II, as it presently exists. In hardware terms, there is not a great deal of difference between the diode scheme, and the theoretically far superior scheme of a weighted decision. In a matrix board (e.g., a 'SEALECTRO' board, such as was used in ESOTERIC I) the diodes may be replaced by resistors, and the buffers by operational amplifiers. Figure 15 illustrates such an alternative decision-taker.

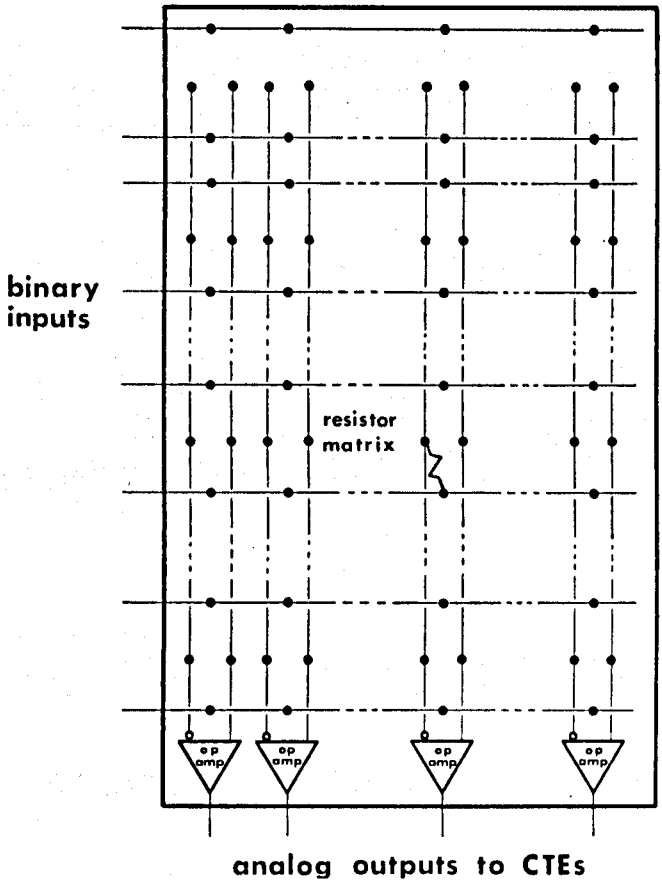



Figure 15. Alternate version of the decision taker

This alternative version is essentially a hardware form of the decision suggested in Hill (1967), and referred to in Hill (1969c). The binary inputs are in pairs, and represent the outputs of bit-pattern flip-flops. One state or the other of a flip-flop will be significant (mere absence of a feature is hardly evidence) and may indicate that a given word is more likely than not to have occurred, or the opposite. A connection, corresponding to the significant

state, should be taken therefore to either increase or decrease the final output for the word. Thus each cell in the matrix, which consists of four alternative connections, will have one connection made by means of a resistor. The form of the decision is

$$\log R_{\text{post}_j} = \log R_{\text{prior}_j} + \sum_{V} \log W_{ij} + \sum_{\bar{V}} \log \bar{W}_{ij}$$

$$R_{\text{post}_j} = \frac{P(E_j/X)}{P(\bar{E}_j/X)}: \quad R_{\text{prior}_j} = \frac{P(E_j)}{P(\bar{E}_j)}$$

$$W_{ij} = \frac{P(e_i/E_j)}{P(e_i/\bar{E}_j)}: \quad \bar{W}_{ij} = \frac{P(\bar{e}_i/E_j)}{P(\bar{e}_i/\bar{E}_j)}$$

$X$ being the set of events which occurred, $E_j$ the input whose likelihood of occurrence we wish to evaluate, and $e_i$ the occurrence of the individual events.

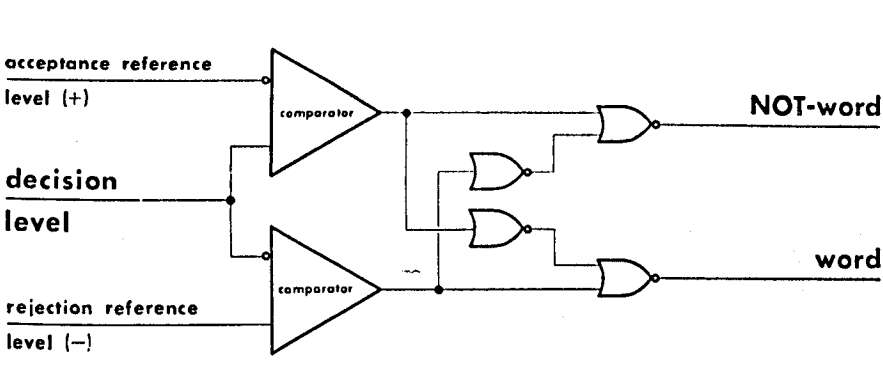



Figure 16. Confidence threshold element

The analog output produced varies between positive and negative extremes, representing extreme likelihood, to extreme improbability. Applying this signal to a variation of the amplitude hysteresis element shown as figure 16, and called a *confidence threshold element* (CTE), allows three regions to be defined. If the signal exceeds the top threshold, we may confidently assert that the word is more likely to have occurred than not. If the signal fails to exceed the bottom threshold, we may confidently assert that the word is more likely not to have occurred than to have occurred. If the signal amplitude falls between these limits, we are unable to make a decision either way. This approach to decision taking is as suggested in the hypothetical recognizer of Hill (1967), and would be one of the next steps in the ESOTERIC project.

**Acknowledgements**

The support provided by the National Research Council of Canada for the continuation of this work is gratefully acknowledged.

## REFERENCES


Andreae, J.H. (1964) STELLA: a scheme for a learning machine. *Proceedings of the 2nd IFAC Congress, Basle 1963*, pp. 503–6. London: Butterworths.

Andreae, J.H. (1965) Private communication.

Andreae, J.H. & Cashin, P.M. (1969) A learning machine with monologue. *International Journal of Man-Machine Studies*, **1**(1), 1–20.

Antony, J. & Lawrence, W. (1962) A resonance analogue speech synthesiser. *Proceedings of the 4th Int. Congress on Acoustics*. Copenhagen.

Arbib, M.A. (1969) Computer metaphors for mental development. *Preprint for Symposium 'Cognitive Studies and Artificial Intelligence Research'*. Univ. of Chicago.

Clowes, M.B. (1967) Perception, picture processing and computers. *Machine Intelligence 1*, pp. 181–97 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver & Boyd.

Clowes, M.B. (1969) Private communication.

Denes, P. (1965) On-line computing in speech research. *Proceedings of the 5th International Congress on Acoustics*, **A23**, Liege.

Dersch, W.C. (1961) Decision logic for speech recognition. *IBM Technical Report* 16.01.106.018, 1 December. San Jose: IBM.

Dudley, H. & Balashek, S. (1958) Automatic recognition of phonetic patterns in speech. *J. acoustical Society of America*, **30**(8), 721–32.

Gold, B. (1966) Word recognition computer program. *MIT Technical Report 452*. Cambridge, Massachusetts: MIT.

Guzman, A. (1967) Scene analysis using the concept of model. *Computer Corporation of America Contract Report AFCRL-67-0133*, January.

Guzman, A. (1968) Decomposition of a visual scene into three-dimensional bodies. *Proc. AFIPS Fall Joint Computer Conf.*, **33**(1), 291–304. Washington: Thompson Book Co.

Hill, D.R. (1966) An abstracted bibliography of some papers relative to automatic speech recognition. *Standard Telecommunication Laboratories Technical Memorandum* 522. Harlow, Essex: STL Ltd.

Hill, D.R. (1966a) STAR – a machine to recognise spoken words. *Proceedings of the IFIP 65 Congress*, p. 357. New York: Spartan.

Hill, D.R. (1967) Automatic speech recognition – a problem for machine intelligence. *Machine Intelligence 1*, pp. 199–226 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver & Boyd.

Hill, D.R. (1969a) Some applications of a small computer (PDP-8) to automatic speech recognition research. *Decuscope*, **8**(3), 2–6.

Hill, D.R. (1969b) Some practical steps taken towards a man-machine interface using speech. *Department of Mathematics, Statistics and Computing Science Research Report No. 73*. Univ. of Calgary, Alberta.

Hill, D.R. (1969c) An ESOTERIC approach to some problems in automatic speech recognition. *International Journal of Man-Machine Studies*, **1**(1), 101–21.

Holmes, J.N., Shearme, J.N. & Mattingly, I.G. (1965) Speech synthesis by rule. *Language and Speech*, **7**(3), 127–43.

Hubel, D.H. & Wiesel, T.N. (1962) Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.*, **160**, 106–54.

Jakobson, R., Fant, C.G.M. & Halle, M. (1969) *Preliminaries to speech analysis*. Cambridge, Massachusetts: MIT Press.

Liberman, A.M. (1957) Some results of research on speech perception. *J. Acoustical Soc. of America*, **29**(1), 117–23.

Mackay, D.M. (1962) Theoretical models of space perception. *Aspects of the Theory of Artificial Intelligence*, pp. 83–103. New York: Plenum Press.

Marril, T. & Bloom, B.H. (1965) The CYCLOPS-2 system. *Computer Corporation of America Report number TR65-RD1.* Cambridge, Massachusetts: Computer Corporation of America.

Martin, T.B., Zadell, H.J., Nelson, A.L. & Cox, R.B. (1966) Recognition of continuous speech by feature abstraction. *RCA Technical Report number TR-66-189.* Camden, New Jersey: Radio Corporation of America.

Moray, N. (1965) Private communication.

O'Connor, J.D., Gerstman, L.J., Liberman, A.M., Delattre, P.C. & Cooper, F.S. (1957). Acoustic cues for the perception of initial /*w,j,r,l*/ in English. *Word*, **13**, 24–37.

Olson, H.F. & Belar, H. (1964) Performance of a code operated speech synthesiser. *Proc. 16th Annual Meeting of the Audio Engineering Society*, October.

Potter, R.K., Kopf, G.A. & Green, H. (1948) *Visible Speech.* New York: Van Nostrand.

Reddy, D.R. (1967) Computer recognition of connected speech. *J. Acoustical Soc. of America*, **42**(2), 329–47.

Reddy, D.R. & Vicens, P.J. (1968) A procedure for the segmentation of connected speech. *J. Audio-Engineering Soc.*, **16**(4), 404–11.

Sakai, T. & Inoue, S-I. (1960) New instruments and methods for speech analysis. *J. Acoustical Soc. of America*, **32**(4), 441–50.

Sakai, T., & Doshita, S. (1962) The phonetic typewriter. *Proceedings of the IFIP 62 Conf.* (ed. Popplewell, C.M.), New York: Humanities Press.

Sakai, T. & Doshita, S. (1963) The automatic speech recognition system for conversational sound. *IEEE Trans. on Electronic Computers*, **EC-12**(6), 835–46.

Samuel, A.L. (1967) Machine learning using the game of checkers II: recent progress. *IBM J. Research & Development*, **11**(6), 601–17.

Selfridge, O.G., & Neisser, V. (1960) Pattern recognition by machine. *Scient. Am.*, **203**(2), 60–8, *also* (1963) in *Computers & Thought* pp. 237–50 (eds Feigenbaum, E.A. & Feldman, J.). New York: McGraw-Hill.

Spinelli, D.N. (1967) Receptive field organisation of ganglion cells in the cat's retina. *Exptl. Neurology*, **19**(3), 291–315.

Tillman, H.G., Heicke, G., Schnelle, H. & Ungeheuer, G. (1965) DAWID I – ein Beitrag zur automatischen 'Spracherkennung'. *Proc. 5th Int. Congress on Acoustics.* Liege.

Uhr. L. & Vossler, C. (1961) A pattern-recognition program that generates, evaluates, and adjusts its own operators. *Proc. Western Joint Computer Conf.*, **19**, 555–70. Revised version appears in *Computers and Thought*, pp. 251–68 (eds Feigenbaum, E.A. & Feldman, J.). New York: McGraw-Hill.

Whitfield, I. C. & Evans, E.F. (1965) Behaviour of neurones in the unanaesthetised auditory cortex of the cat. *Report of the Neurocommunications Research Unit for March 1964 to February 1965 on Contract DA-91-591-EUC-2803 of the U.S. Army.* Birmingham, UK: Univ. of Birmingham.

# 26

# On Imitative Systems Theory and Pattern Recognition

P. A. V. Hall
City University
London

A general theory is formally developed to tackle the problem of designing one system to imitate functionally a second given system: the theory is applicable to scientific modelling, control process identification, and to intelligent machine design, and the application of the theory to visual pattern recognition is given in some detail. Information about the given system is obtained experimentally as samples of the input–output mapping of the system; and from these samples parameters in the model or design are estimated. Algorithms result which can be iteratively automated, thus leading to machine learning. Particular attention is given to convergence of the methods, whereby increasing the number of samples and number of parameters systematically leads to arbitrarily close matching of the model and given system.

## 1. INTRODUCTION AND SUMMARY

In this paper we are concerned with general problems of science, of modelling and knowledge, and how these relate to specific engineering problems.

Modelling in the broad sense of scientific theory is fundamental to pure science and thus to engineering, but modelling theory is also of direct interest to engineering. In Automatic Control it is always necessary to describe by means of a suitable model the functioning and behaviour of the manufacturing plant or other system to be controlled, and often one goes further and actually produces a physical model of the plant (in predictive and adaptive control). Considerable effort at present is spent in trying to automate many of the more menial tasks performed by humans: the task of sorting letters in the Post Office, and data preparation for computers, are examples. Often also there is a desire to mechanize where humans cannot agree, or the task is too complex: for example, in weather forecasting and medical diagnostics.

These problems are all essentially similar, in that a given system is to be imitated by some physical model or artifact, with the stipulation that the

input–output mappings must be similar (but we do not care if the means whereby the mappings are produced are not the same). That the control example is of such a type is clear: in letter sorting the given system is a human sorter with input the envelope and output the particular action to sort the envelope appropriately; in data preparation the given system is a human, with input the character to be punched on paper tape or card, and output the corresponding code of holes: in weather forecasting and medical diagnostics the given system is a part of Nature, with inputs the measurements made, and outputs the prediction, wet or fine, healthy or diabetic, or what have you. These problems are all exceedingly complex and defy a naive solution: thus we seek some generally applicable method whereby they may be systematically solved. It is with this solution that we are concerned primarily here. In the latter half we apply the theory to a data preparation type of problem, to illustrate the appropriateness of the theory: in Automatic Control the use of Wiener's theory of non-linear systems (Cuenod and Sage 1968, Wiener 1958) is a manifestation of a part of our theory (the use of orthogonal functions) and there is scope for the application of the general techniques (Hall 1968).

Before we can design the artifact, we must of course first find out some facts concerning the thing we wish to model. Knowledge concerning the given system can come from two sources, termed here 'direct' and 'indirect'. Direct knowledge is acquired by experimentation directly upon the given system to acquire samples from the input–output mapping; indirect knowledge is often called 'a priori', and includes knowledge of the intuitive kind, as well as constraints like dimensional analysis and conservation theorems (Kline 1965), and microscopic knowledge concerning the inner workings of the system. It is with direct knowledge that we shall be primarily concerned here.

When designing a machine of any kind, one is necessarily constrained to consider only some limited set of variables: these could be component types and values and tolerances, and their interconnexions, in an electronic circuit; or rotor diameter, number of commutator segments, number of turns per segment, brush type, bearing type, etc., in an electric motor. Thus we shall view design as the task of estimating a finite set of parameters.

Then the complete design method can be crudely described as follows. In our model we build in 'adaptability' by arranging that the model can do one thing if the parameters have one set of values, can do another different thing if the parameters have another different set of values, and as the parameters are varied a whole infinity of possible behaviours of the model are generated. The greater the number of parameters, the more degrees of freedom in the model, and the more adaptable it becomes. The task is to choose, from among all these possible behaviours, the behaviour most similar to the given system (we assume that we can measure similarity). To make this choice one observes the given system and learns about its behaviour, and as one learns more and

more, the parameters of the model are adjusted to force the model to replicate more and more of the given systems' behaviour; and this is done in such a way that on the average the selected behaviour for the model is closest to the given behaviour (i.e. is most similar).

In fact the modelling process can be completely automated to make a general purpose model producing machine. This machine can then be viewed as imbued with 'intelligence': it learns, adapts, generalizes on its experience. Furthermore such a machine would be failure tolerant – for if hardware failure is viewed as the loss of a parameter and its associated amount of adaptability, then hardware failure does not mean total loss of function, but rather only a degradation in performance, in the accuracy of imitation.

This formulation of modelling from direct knowledge (samples) has much akin to the problems of induction and inference considered by Meltzer and by Popplestone (this volume), except that here we consider many samples, and real numbers and the powerful structure they induce. In these other approaches, indirect knowledge appears as a doctrine of simplicity, the order in which hypotheses are tested, and is loosely equivalent to our choice of functions $\{\phi_i\}$ (see later text for details: perhaps a better correspondence appears in methods which seek to find the best $\{\phi_i\}$ by 'evolutionary' methods).

In section 2, definitions are given to formalize the problem in general terms – this is important in order to avoid confusion concerning heuristic notions such as 'normal operating conditions', etc. In section 3, parameter estimation algorithms are derived for a special case, where the criterion of similarity is the mean square difference, and the input and output sets are vector spaces. These algorithms mostly appear in a form iterative with increasing number of samples, and hence can be viewed as 'learning'. This special case is, however, of universal applicability by virtue of the fact that one can compound devices from fixed outer stages and an adaptable central stage, thus realizing any type of mapping. In section 4 an example is worked through to illustrate the theory and the relative efficiencies of the various techniques discussed. In section 5 the theory is applied in some detail to optical character recognition (the data preparation problem) to provide both an overview of the subject useful as a theoretical guideline, and specific techniques similar to those existing, but here based on secure theoretical foundations.

## 2. FORMAL STATEMENT OF THE PROBLEM

We shall now consider, formally and precisely, the various aspects of the problem.

*Definition* 1. A *system* is the set $S=(X, Y, f)$,

where $X$ is the set of possible *inputs*,

$Y$ is the set of possible *outputs*,

$f$ is a general mapping of $X$ into $Y$.

(This definition is intended to embrace the usual definitions of machines used

in algebraic machine theory.)

*Definition* 2. An *operating system* is the set $S' = (S, p)$, where $S$ is a system and $p$ is a probability measure defined on the input set $X$.

This is intended to formalize the notion of normal operating conditions in a stationary environment, and is of vital importance when comparing two systems. Note that the operating distribution may not be known explicitly, but can be observed implicitly in the normal operation of a system.

*Definition* 3. A *sampled system* is the set $S'' = (S, Q)$, where $S$ is a system and $Q$ a sampling procedure which generates a sequence of samples of the input–output mapping, $f$, of $S$, $\{(x_k, y_k) : k = 1, \ldots, N, \ldots\}$ according to some rule and without limit. A system can be sampled either *actively*, by choosing an input and recording the corresponding output, or *passively*, by observing the system while operating normally, and recording both inputs and outputs as they appear. Most commonly sampling will be random with respect to some probability measure $q$, and passive so that $q$ coincides with the operating distribution $p$: sometimes it may be appropriate to use other techniques. (Clearly it may be relevant to consider continuous sampling, where sampling functions (of time) are observed, but this will not be considered in this paper.)

*Definition* 4. A *parametrized system* is a system $(X, Y, \phi(a))$ in which the input–output mapping takes the form $\phi(a)$, where $a$ is a sequence of parameters which may be unbounded. As the parameters are varied a whole family of systems is created. [We shall mostly deal with linearly parametrized systems, where $\phi(a) = \sum_{i=1}^{M} a_i \phi_i$, and $Y$ is a vector space over the real numbers. An interesting example of non-linear parameters is afforded by the contribution to the Workshop by Michie and Ross, where a distance function on the set of possible states $S$ of the GRAPH TRAVERSER (i.e., a mapping of $S \times S \to R$) is to be estimated.]

This formalizes the notion of design constraints discussed in section 1.

It is of interest to be able to compare two operating systems $S_1 = (X, Y, f_1, p)$ and $S_2 = (X, Y, f_2, p)$, and to measure the extent to which the two systems are similar. Since the systems only differ in their input–output mappings, this reduces to a measure of difference between $f_1$ and $f_2$.

*Definition* 5. Let there exist on the space of functions $F = \{f : X \to Y\}$ a functional $e(f_1, f_2)$ having the following properties:

(i) $e(f_1, f_1) = 0$

(ii) $e(f_1, f_2) = e(f_2, f_1) \geq 0$.

We shall call $e(f_1, f_2)$ the *error* between $S_1$ and $S_2$. In the particular case where $F$ is a metric space with distance function $d$, $d$ can be used to measure the error.

The situation with which we shall be predominantly concerned is the mean square error, with $F$ the space of square integrable functions (that is, $L^2$),

when $e(f_1, f_2) = \sqrt{(\int_X |f_1 - f_2|^2 \, d\mu)}$. Usually the error would depend upon the operating probability measure $p$; thus $\mu = p$ in the integral above.

Using the preceding concepts, we shall now formally state the problem.

*The problem.* Given an operating system $S_1 = (X, Y, f, p)$, and an operating parametrized system $S_2 = (X, Y, \phi(a), p)$, can we determine the values of a finite set of parameters $(a_1, a_2, \ldots, a_M)$ (where the remaining parameters in the sequence $a$ are zero or arbitrary) using only a finite set of samples ($N$ in number) of $S_1$, such that the error between systems $S_1$ and $S_2$ is less than some prespecified amount $\varepsilon$? As an intermediate stage we must set up a sampled system $S_3 = (X, T, f, Q)$, where $Q$ may or may not be related to $p$: part of the problem is to find suitable sampling procedures $Q$.

Clearly a basic requisite to a solution to the problem must be a convergent methodology, that in the limit of increasing number of parameters ($M$) and number of samples ($N$), exact matching of the two systems will be achieved. In practice, exact matching would not be achieved, and the error function would have some proper interpretation as a cost or similar. Solution schemes might then involve fixing $M$, successively estimating the parameters with more and more samples until some estimate of the error indicates success or otherwise. If otherwise, more parameters would need to be considered and the process repeated: if success, the number of parameters could perhaps be reduced to economize the solution.

When $X$ is a subset of the $n$-dimensional real vector space and $Y$ is the real line $R$, this problem reduces to the well-studied problems of numerical approximation (Handscomb 1966) and stochastic approximation (Chien and Fu 1967). In the next section I shall give a brief account of the techniques available to us to estimate linear parameters with least mean square error. I shall refer to this as the *problem in canonical form.* Of course there are techniques available for different input and output spaces and other error measures (notably the minimax error measure), but in this paper these will be passed over.

When the input and output spaces are not of the canonical form, we can convert the problem to that form by compounding the imitating system of three 'layers', preceding and succeeding a parametrized system (or systems) in canonical form by appropriate fixed systems. This idea should become clear when applying it to the example and to visual pattern recognition in the latter part of this paper.

## 3. SOLUTION TO THE PROBLEM IN CANONICAL FORM

We shall restrict ourselves here to the operating system $S_1 = (X \subset R^n, R, f, p)$ and parametrized system $S_2 = \left(X \subset R^n, R, \phi(a) = \sum_{i=1}^{M} a_i \phi_i, p\right)$. Thus the inputs are $n$-dimensional real vectors, outputs are real numbers, and parameters appear linearly.

Further, let us restrict ourselves to the mean square error criterion. Thus the error between $S_1$ and $S_2$ is given by

$$e(f,\phi(a))=\int_X\left(f(x)-\sum_{i=1}^{M}a_i\phi_i(x)\right)^2 \mathrm{d}p \tag{3.1}$$

(note that we are dropping the more usual square-root on the right) and the problem is to estimate the parameters $a_i$ so as to minimize the integral (3.1), given samples from $f(x)$, $\{(x_k,f(x_k)): i=1,\ldots,N\}$. These samples are obtained by some yet to be determined procedure $Q$.

### 3.1. Orthonormal functions

Suppose that the $\phi_i$s are drawn from some complete orthonormal sequence of square integrable functions. Then if $f$ is also square integrable, we know (Sz.-Nagy 1962, for example) that the parameters which give the least squares matching are:

$$a_i=\int_X f\phi_i\,dp\colon i=1,2,\ldots,M \tag{3.2}$$

where

$$\int_X \phi_i\phi_j\,\mathrm{d}p=\delta_{ij}=\begin{cases}1 \text{ if } i=j\\ 0 \text{ if } i\neq j\end{cases}. \tag{3.3}$$

Further, we know that as the number of parameters are increased without limit,

$$\lim_{M\to\infty}\left(f(x)-\sum_{i=1}^{M}a_i\phi_i(x)\right)=0$$

almost everywhere, and thus the error can be reduced to less than any prescribed $\varepsilon$.

Since we do not know $f$ explicitly, but have at our disposal samples from $f$, we can estimate the integrals (3.2) (and thus the parameters) using standard numerical techniques. However, most techniques are specific to one dimensional integrals, and the only generally applicable technique available is the Monte Carlo method (Hammersley and Handscomb 1964). Thus we shall estimate the $a_i$ by random sampling, setting

$$a_i^{(N)}=\frac{1}{N}\sum_{k=1}^{N}f(x_k)\phi_1(x_k) \tag{3.4}$$

The variance of this estimate is proportional to $(1/N)$.

Equation (3.4) can be rewritten in iterative form as

$$a_i^{(N+1)}=\frac{N}{N+1}a_i^{(N)}+\frac{1}{N+1}f(x_{N+1})\phi_1(x_{N+1}) \tag{3.5}$$

and adding a new sample can be interpreted as learning by weight adjustment.

The sampling procedure $Q$ is either to draw actively samples from $X$ randomly with respect to the probability measure $p$, the normal operation of $S_1$ to generate the samples.

The use of orthonormal functions has the attractive property that the terms in the approximating expansion $\phi(a)$ are independent: terms can be added or removed without affecting any other terms. But there is also the severe disadvantage that the orthonormality condition (3.3) rests upon the measure $p$, which must thus be known before starting. To overcome this, we can either sample with respect to some other measure (Hall 1968), or remove the orthonormality constraint. The latter is considered in the following section.

### 3.2. Linearly independent functions

To meet the orthonormality requirement (3.3) we must explicitly know the probability measure $p$ in order to be able to choose suitable functions $\{\phi_i\}$. This is not in general the case, so let us now only restrict the functions $\{\phi_i\}$ to some complete linearly independent sequence.

Since we are dealing with mean square error and square integrable functions, again we know (Achieser 1956, Sz.-Nagy 1962) that there exists a unique set of parameters $(a_1, a_2, \ldots, a_M)$ which minimize the mean square error (3.1). Differentiating (3.1) with respect to the parameters we obtain

$$\frac{\partial}{\partial a_i}(e(f,\phi(a))) = 2\int_X \phi_i(\sum a_j\phi_j - f)\,\mathrm{d}p.$$

Setting this to zero for the extremum, and changing the order of integration and summation, we obtain the optimum parameters as the solution of the following system of simultaneous linear equations:

$$Ta = c \tag{3.6}$$

where

$$\left.\begin{aligned} t_{ij} &= \int_X \phi_i\phi_j\,\mathrm{d}p \\ c_j &= \int_X f\phi_j\,\mathrm{d}p \\ a_i &= \text{unknown parameters} \end{aligned}\right\} i,j = 1, 2, \ldots, M.$$

If we estimate the elements of $T$ and $c$ from the samples by the Monte Carlo technique, we then obtain, after cancelling common factors, the system of equations:

$$T^{(N)}a^{(N)} = c^{(N)} \tag{3.7}$$

where

$$\left.\begin{aligned} t_{ij}^{(N)} &= \sum_{k=1}^{N} \phi_i(x_k)\phi_j(x_k) \\ c_j^{(N)} &= \sum_{k=1}^{N} f(x_k)\phi_j(x_k) \\ a_i^{(N)} &= N\text{-sample estimate of } a_i \end{aligned}\right\} i,j = 1, 2, \ldots, M.$$

Samples are drawn randomly from $X$ with respect to $p$. Clearly $T^{(N)}$ is symmetric; it can also be shown that $T^{(N)}$ is positive semi-definite.

The system (3.7) could have been found by minimizing the mean square error over the sample points, when (3.7) appears as the normal equations of Numerical Analysis (Hall 1968).

While the system (3.7) permits a unique solution because the $\{\phi_i\}$ are linearly independent, this is not guaranteed for (3.7). However it can be shown that (3.7) is always consistent (Lanczos 1957), and thus there is always at least one solution and this is unique only if $T^{(N)}$ is non-singular: this is not in general guaranteeable though for many function systems this will be the case for $N>M$. Because of this possibility of singularity, we shall consider a biased, though consistent (Kendall and Stuart 1961) estimate of $T$

$$U^{(N)}a^{(N)}=c^{(N)} \tag{3.8}$$

where $U^{(N)}=T^{(N)}+I$ and $I$ is the $M\times M$ identity matrix. $U^{(N)}$ is non-singular and positive definite for all $N$.

One possible technique for generating an imitating system is then to sample the given system some predetermined large number of times and form the matrix $T^{(N)}$ and vector $c^{(N)}$, and then solve (3.7) by a standard numerical technique – such techniques if properly performed would handle the case for $T^{(N)}$ singular and thus the linear independence of the $\{\phi_i\}$ could also be waived. But we would also like to be able to consider 'learning' type techniques.

Adding another sample means updating the vector $c^{(N)}$ and the matrix $T^{(N)}$ (or $U^{(N)}$): successively considering more and more samples would mean at each stage solving (3.7) or (3.8) and throwing away the preceding solution: to avoid this there are two ways out.

Firstly, we can solve the equations iteratively, using

$$a^{(N,k+1)}=A^{(N)}a^{(N,k)}+B^{(N)}c^{(N)} \tag{3.9}$$

where $a^{(N,k)}$ is the estimate of $a$ with $N$ samples, after $k$ iterations, started with $a^{(N,0)}=a^{(N-1,K)}$ where $K$ is the total number of iterations with $N-1$ samples. The matrices $A^{(N)}$ and $B^{(N)}$ are related to $T^{(N)}$ (or $U^{(N)}$) in a way which depends upon the particular iteration scheme considered – each iteration scheme has its own convergence conditions (Faddeeva 1959). For our case the Seidel technique can be shown to be always convergent for $U^{(N)}$ (because $U^{(N)}$ is real, symmetric, and positive definite). In the Seidel technique, $U^{(N)}$ is split into an upper triangular (including the diagonal elements) matrix $Q$ plus a strictly lower triangular matrix $V$ (thus $U^{(N)}=Q+V$): then $A^{(N)}=-Q^{-1}V$ and $B^{(N)}=Q^{-1}$. It is interesting to note that gradient hill-climbing techniques for minimizing the mean square error over the sample points are essentially of the form (3.9).

Secondly, if we examine in detail the process of adding a sample, we find the iterative scheme suggested by Kashyap and Blaydon (1966):

$$a^{(N+1)}=a^{(N)}+(T^{(N+1)})^{-1}\phi(x_{N+1})(f(x_{N+1})-\phi^T(x_{N+1})a^{(N)})$$

$$(T^{(N+1)})^{-1}=(T^{(N)})^{-1}-\frac{(T^{(N)})^{-1}\phi(x_{N+1})\phi^T(x_{N+1})(T^{(N)})^{-1}}{1+\phi^T(x_{N+1})(T^{(N)})^{-1}\phi(x_{N+1})} \tag{3.10}$$

where $\phi^T(x_N)=(\phi_1(x_N), \phi_2(x_N), \ldots, \phi_M(x_N))$.

An analogous formula holds for $U^{(N)}$ substituted for $T^{(N)}$ which is equivalent to starting from $T^{(0)}=I$. Since $U^{(N)}$ is non-singular for all $N$, the modified formula holds for all $N$ and the scheme is well behaved.

### 3.3. Stochastic approximation techniques

Iterative schemes for estimating parameters, such as those of (3.5) and (3.10) above, can be viewed as stochastic approximation sequences (Chien and Fu 1967). (3.5) and (3.10) converge as a consequence of the way in which they were derived and do not require the general theory, but the class of algorithms developed by Aizerman, Braverman and Rozonoer (1964a,b,c, 1965) and extended by Kashyap and Blaydon (1966) and Devyaterikov, Propoi, and Tsypkin (1967) rest upon this general theory. These algorithms are mostly of the form below, and estimate least squares parameters, thus minimizing (3.1).

$$a^{(N+1)}=a^{(N)}+\gamma(N)\phi(x_{N+1})(f(x_{N+1})-\phi^T(x_{N+1}(a^{(N)}). \tag{3.11}$$

or, to minimize mean absolute error;

$$a^{(N+1)}=a^{(N)}+\gamma(N)\phi(x_{N+1})\ \text{Sign}\ (f(x_{N+1}-\phi^T(x_{N+1})a^{(N)}). \tag{3.12}$$

where $\{\phi_i\}$ are linearly independent and square integrable, and $\gamma(N)$ are positive real numbers such that

$$\lim_{N\to\infty}\gamma(N)=0, \quad \sum_{N=1}^{\infty}\gamma(N)=\infty, \quad \sum_{N=1}^{\infty}\gamma(N)^2<\infty \tag{3.13}$$

Intuitively one would expect (3.10) (and (3.5)) to converge faster than (3.11) or (3.12) for it uses more 'information': this has been found in practice – see the example of section 4: some analytical ranking seems desirable, though it is not yet available: convergence rates for (3.7) and (3.10) are known (Wagner 1968).

### 3.4. Error of the estimates and estimates of the error

With a finite set of parameters, the imitating system will not in general be able to match exactly the given system, but will exhibit some error. Further, since the parameters are estimated from samples they will not be least squares coefficients, but 'noisy' versions of them; this will add to the mean square error between imitating and given systems.

The mean square error with least squares parameters $a$ is:

$$e=\int_X (f-\sum a_i\phi_i)^2\,\mathrm{d}p \tag{3.14}$$
$$=\int_X f^2\,\mathrm{d}p-a^TTa$$
$$=\int_X f^2\,\mathrm{d}p-a^Tc,$$

where $T$ and $c$ are as defined in section 3.2.

The mean square error with estimated parameters $a^{(N)}$ is then

$$e_N=e+(a^{(N)}-a)^TT(a^{(N)}-a) \tag{3.15}$$

thus showing, since $T$ is positive definite, that $e_N$ is greater than $e$.

If we wish to assess the goodness of our imitation, it is $e_N$ that we must estimate. (We recall that the strategy is then to either accept the imitation as adequate if $e_N$ is small enough, or otherwise continue sampling or consider more parameters or both in order to reduce $e_N$ to an acceptable level.) Estimating $e_N$ could be done directly using an independent set of $L$ random samples and a Monte Carlo estimate of the integral (3.14) (with $a_i^{(N)}$ instead of $a_i$), so 'testing' the system.

$$e_N^{(L)}=\frac{1}{L}\sum_{j=1}^{L}(f(x_j)-\sum_{i=1}^{M}a_i^{(N)}\phi_i(x_j))^2. \tag{3.16}$$

However, the number of samples $N$ required to estimate the parameters, and the number of samples $L$ required above to 'test' the imitation, may be very large (perhaps in excess of $10^5$ if the required matching must be close) and it is reasonable to ask, could we not also estimate the mean square error while estimating the parameters, and thus elicit a considerable economy? Of course we can, for $M$ the number of parameters will be much less than $N$ and thus we will have extracted only a fraction of the potential 'information' from the samples.

Consider firstly orthonormal functions. The expected value of $e_N$ (i.e. the average when considering all possible sets of $N$ samples) is then

$$E(e_N)=e+\sum_{i=1}^{M}E((a_i^{(N)}-a_i)^2). \tag{3.17}$$

The expectations on the right are variances. An adequate estimate of (3.17) would be:

$$e_N^{(N)}=\frac{1}{N}\sum_{k=1}^{N}f^2(x_k)-\sum_{i=1}^{M}\left((a_i^{(N)})^2+2V^{(N)}(a_i^{(N)})\right), \tag{3.18}$$

where $V^{(N)}(a_i^{(N)})$ is an estimate of the variance of the samples. For this we could use the sample variance

$$V^{(N)}(a_i^{(N)})=\frac{1}{N-1}\sum_{k=q}^{N}\left(f(x_k)\phi_i(x_k)-a_i^{(N)}\right)^2 \tag{3.19}$$
$$=\frac{1}{N-1}\sum_{k=1}^{N}f(x_k)^2\phi_i(x_k)^2-\frac{N}{N-1}(a_i^{(N)})^2. \tag{3.20}$$

(For pattern classifications discussed in section 5, $0 \leqslant f(x) \leqslant 1$ and we can show that Variance $(a_i^{(N)}) \leqslant \frac{1}{N}$ and thus use this as a crude estimate, with some economy.)

For general non-orthogonal sequences of functions the theory is more complicated and we consider rather the following estimates which are consistent (Kendall and Stuart 1961) but which may be biased towards too low a value:

$$e_N^{(N)} = \frac{1}{N-M}\left(\sum_{k=1}^{N} f(x_k)^2 - (a^{(N)})^T T^{(N)} a^{(N)}\right) \tag{3.21}$$

$$e_N^{(N)} = \frac{1}{N-M}\left(\sum_{k=1}^{N} f(x_k)^2 - (a^{(N)})^T U^{(N)} a^{(N)} - 2(a^{(N)})^T a^{(N)}\right). \tag{3.22}$$

The preceding analysis of error is strictly speaking of first order value only for we have said nothing of the distribution of the estimates and cannot apply confidence limits. This would require knowledge of higher order moments and is here neglected as being computationally prohibitive: rather in practice one should treat the estimates with some sceptism, but as an invaluable guide to the design process.

## 4. AN ILLUSTRATIVE EXAMPLE

### 4.1. Given system

We are given a system which classifies pairs of numbers, each in the range $(-1, +1)$, according to the following rule:

$$f(x_1, x_2) = 1 \text{ if } 0.2 \leqslant x_1 \leqslant 0.5, \text{ or } (x_1+0.5)^2 + (x_2+0.3)^2 \leqslant 0.16$$
$$= 0 \text{ otherwise.}$$

Input pattern space is shown in figure 1. Under normal operating conditions pairs of numbers arrive distributed uniformly in the rectangle $-1 \leqslant x_1 \leqslant +1$; $-1 \leqslant x_2 \leqslant +1$.

### 4.2. The problem

We wish to use a digital computer to replace the given system with as few errors as possible.

### 4.3. Relating to theory

Input space $X = \{(x_1, x_2): -1 \leqslant x_1 \leqslant +1; -1 \leqslant x_2 \leqslant +1\}$.
Output space $Y = \{0, 1\}$.
Mapping $f$: $X \rightarrow Y$ assumed unknown (but as defined above). Operating measure $p$ is given by probability density function

$$p(x_1, x_2) = \tfrac{1}{4}.$$

We need to approximate $f$ by some function $g: X \rightarrow Y$. We shall realize $g$ as a two-stage mapping $h(\phi(a, x_1, x_2))$ where $\phi(a, x_1, x_2)$: $X \rightarrow R$ (i.e., maps $X$ to the real numbers) and $h(r)$ is a threshold function setting $h(r)$ to 1 if

$r \geqslant \frac{1}{2}$ and to 0 otherwise. In a sense $h$ 'cleans up' $\phi$ and can be viewed as a decision procedure. Also, since $Y \subset R$, we can view $f$ as a function of the same type as $\phi$ – and shall attempt a least squares matching of $f$ and $\phi$.

The parametrized system mapping $\phi(a)$ will be linear in the parameters – thus $\phi(a, x_1, x_2) = \sum_{i=1}^{M} a_i \phi_i(x_1, x_2)$; the function system $\{\phi_i\}$ used will be polynomials, since these are the easiest (and cheapest) to program in the language used (FORTRAN IV) – hence

$$\phi_i(x_1, x_2) = \sum_{k=0}^{k} \sum_{j=0}^{j_i} b_{jk} x_1^j x_2^k.$$

The problem is then, having fixed the $\phi_i$s (i.e., the $b_{jk}$s) to determine the $a_i$s so that $\phi$ approximates $f$ in the least squares sense.

A polynomial system orthonormal in $X$ w.r.t. $p$ can be easily formed from the Legendre polynomials. Suppose $\{P_i(x)\}$ is the system of normalized Legendre polynomials in one dimension, then we form in two dimensions the system $\{P_i(x_1, x_2) = P_{j_i}(x_1) P_{k_i}(x_2)\}$, which is clearly orthonormal if the original normalization in one dimension was with respect to probability distribution (weighting function) $p(x) = \frac{1}{2}$.

### 4.4. Analytical solution

Since we know everything about the problem we can solve it analytically to find, for any given polynomials, the correct least squares coefficients. This was done (with the aid of the computer) and the mean squares error (MSE) and error-rate (ER) are plotted in figure 2 as a function of number of parameters (using normalized Legendre polynomials). This illustrates how, with increasing number of parameters, performance improves, and for any fixed set of parameters represents the limit for the learning techniques.

The curve $P$ shows the error-rate predicted under the hypothesis that errors are Gaussian distributed with variance equal to the mean square error. The lack of agreement demonstrates the inadequacy of this hypothesis.

### 4.5. Learning the parameters

The parameters were fixed to a set of 64 (all polynomials with powers of $x_1$ and $x_2$ less than or equal to 7) and the various techniques of section 3 were programmed. Performance was assessed by independent sets of 10,000 samples each – the results are plotted in figures 3 and 4, mean square error and error-rate respectively, where the curves are:

A – Orthonormal functions (Legendre) using (3.5).

B – Exact solution of normal equations (3.8) (using biased estimate $U$) by Gaussian elimination (Faddeeva 1959). Function system used was Legendre – other systems would be the same, except for round-off noise which could be appreciable. ((3.10) would converge at same rate, but was not programmed.)

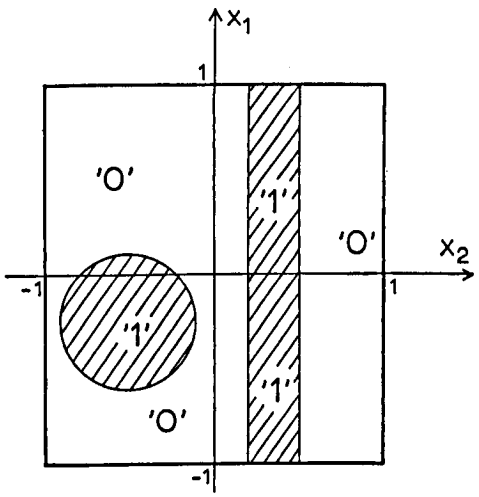



Figure 1. Input 'Pattern' Space is area inside large square: points within input set classified as '0' or '1' as indicated (shaded area is '1').

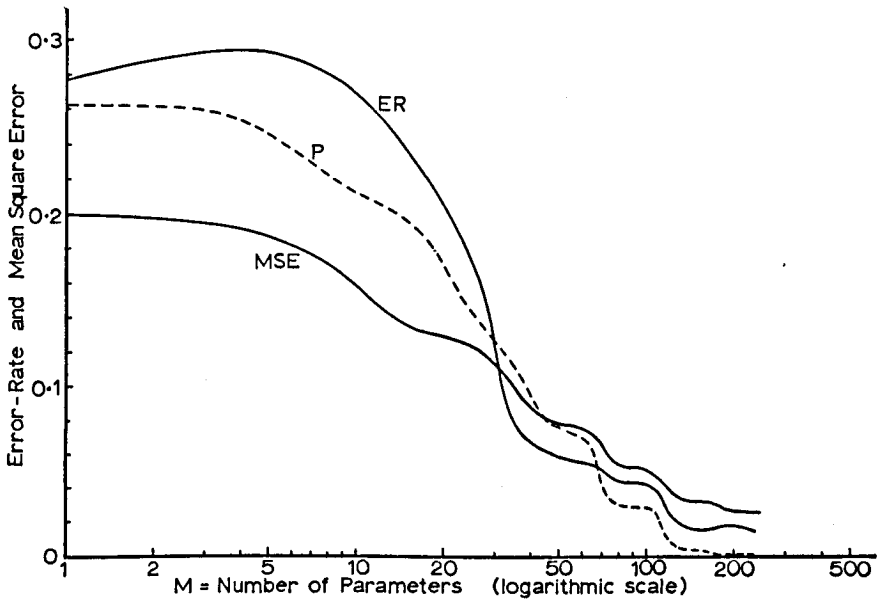



Figure 2. Analytical solution. Mean square error (MSE) and Error-Rate (ER) as function of number of least square parameters (M).

C – Seidel iteration of normal equations (3.9), using 1 cycle per sample ($K=1$) and Legendre polynomials, which represent the limit of well-behaved functions. (With $K=6$ the performance was indistinguishable from B.)

D – Seidel iteration (3.9), using 6 cycles per sample ($K=6$) and simple powers of $x_1$ and $x_2$, which represent the limit of ill-conditioned equations.

E – Stochastic approximation: best case (3.11) with Legendre polynomials; other function systems or (3.12) would be worse.

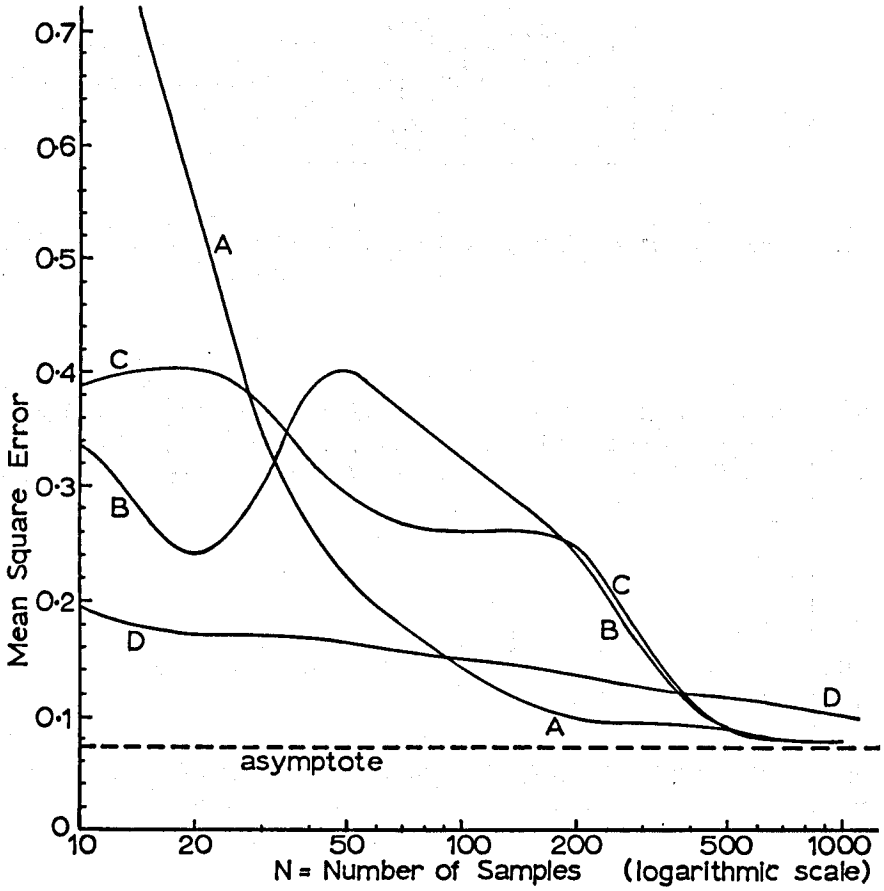



Figure 3. Learning, 64 parameters. Mean square error as a function of number of samples (N) for parameter estimation, for various techniques (see text: E-curve unplottable, MSE > 1·5).

We see how all the techniques converge with increasing number of samples (except for the initial period where random fluctuations are significant). We see how much better the 'exact' techniques (A and B) are, compared

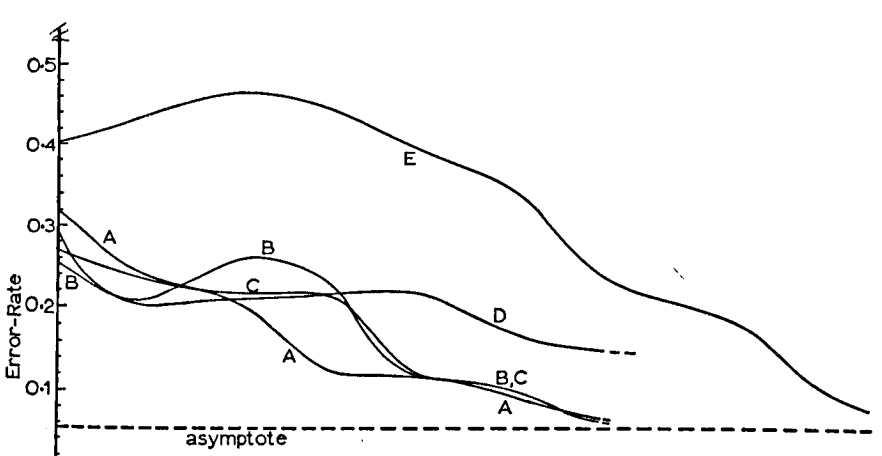



Figure 4. Learning, 64 parameters. Error-rate as a function of number of samples (N) for parameter estimation, for various techniques (see text).

to the stochastic approximation techniques (E), with the Seidel methods (C and D) coming in between.

In Table 1 a comparison is made between the various techniques in terms of run time and storage requirements (for 1,000 samples, 64 parameters and all tests). This gives some indication, though crude, of the relative economies. The requirements of the Kashyap and Blaydon technique (3.10) (not programmed) would be somewhat worse than the Seidel technique (C).

| Technique | Run time | Storage |
|---|---|---|
| A | 0.194 | 0.138 |
| B | 0.514 | 1.0 |
| C | 0.578 | 0.557 |
| D | 1.0 | 0.557 |
| E | 0.250 | 0.138 |

Table 1. Relative economies of learning techniques in example of section 4

## 5. APPLICATION TO VISUAL PATTERN RECOGNITION

The general desire is to make a machine which will duplicate the human ability to recognize visual patterns. Let us suppose here that the problem is

to build a machine which will perform alphanumeric classifications of visual patterns: that is, the data preparation task is to be automated.

### 5.1. The system to be imitated

The given system is a human being or perhaps a collection of cooperating human beings, which signals the nature of a given pattern – what alphanumeric character it represents, if any. (Note that no ambiguity is allowed, the pattern classes do not intersect.) In a sense the given system is the 'teacher'.

The set of possible inputs are light distribution over some 'visual field'. Thus

$$X = \{d(\xi_1, \xi_2) : (\xi_1, \xi_2) \in V \subset R^2\}. \tag{5.1}$$

$X$ is a function space with elements $d(x, y)$ which describe the intensity of light at all points within the two-dimensional visual field $V$. $X$ is in general of infinite dimensionality.

The outputs are the set

$$Y = \{A, B, C, \ldots, Z, 1, 2, \ldots, 9, 0, \pi\} \tag{5.2}$$

where $\pi$ indicates the null classification, to which patterns which are not alphanumeric characters belong.

The operating probability distribution is not known *a priori* and will be discovered implicitly while designing the artifact.

A realistic criterion of error is the error-rate, which is the probability that the artifact makes a classification in disagreement with the human standard. It can be shown from the Tchebyshev inequality that error-rate is bounded by four times the mean square error. I will presume the design objective is to make the error-rate less than some prespecified $\varepsilon$.

### 5.2. The artifact

As indicated at the close of section 2, the intention is to realize an artifact by a three-stage process, the central one of which is in canonical form, where the learning takes place. This means preceding a canonical system by a fixed system which maps $X$ into $R^n$, and succeeding the canonical system by a fixed system which maps $R$ onto $Y$. In fact we shall use a central section consisting of $m$ (here $m=36$, the number of non-null categories) canonical systems in parallel, and thus the final stage will map $Z=R^m$ onto $Y$. This is schematized in figure 5 and is the usual form (though by no means the only possible form) that pattern recognition devices take (Nagy 1968, Nilsson 1965, Sebestyen 1962).

The first stage comprises the measurement devices (transducers) which characterize the pattern by $n$ numbers – most often the physical form is a lens system followed by a photomosaic, though it could be a coherent optics 2D Fourier analyser or similar. Mathematically this can be viewed as an approximation in which $d(x, y)$ is projected into a subspace $W$ of $X$.

$$d(\xi_1,\xi_2) = \sum_{i=1}^{n} w_i \psi_i(\xi_1,\xi_2) \tag{4.3}$$

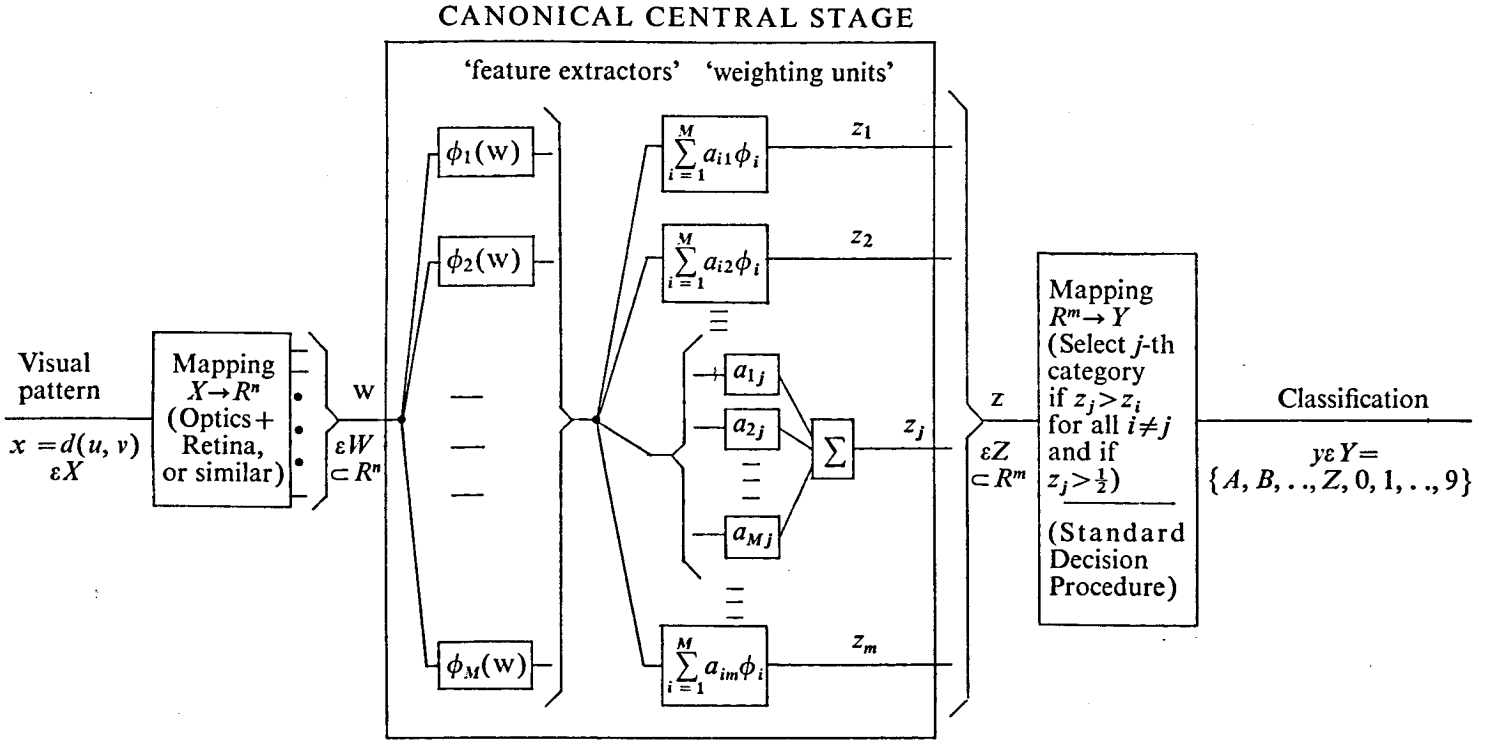



Figure 5. Visual pattern recognition. (Recognizer block diagram)

$\{\psi_i\}$ is the system of measurement functions which spans $W$: the coordinate vectors of $W$ are elements of the $R^n$. We shall suppose that $n$ is large enough, that there is a negligible error due to the mapping from $X$ into $R^n$ (Hall 1968).

The papers by Paton, by Rutovitz, and by Searle (in this volume) all predominantly discuss the representation problem, transformations (perhaps non-linear) on the space $W$; while van Emden shows a way of reducing $n$, the dimensionality of $W$, or $M$, the number of features, using $p$, but independently of any particular classification of points of $X$ or $W$.

The final stage can be viewed as a scheme whereby $Y$ is embedded in $R^n$. Several ways of achieving this are possible. Sebestyen (1962, Chapter 3) suggested using $m=1$, imbedding $Y$ in the real line: unfortunately this imposes a critical ordering in $Y$ which would mean that a continuous deformation of a 0 to a 9 not passing through any intermediate character, would in the artifact with continuous $\{\phi_i\}$ signal all intermediate characters, 1 to 8. Thus here we choose a more standard approach, combining threshold and maximum amplitude selections (Nagy 1968, Nilsson 1965).

Let us impose an arbitrary ordering on the set $Y$, with the null-classification $\pi$ as $y_0$. Then we shall imbed $Y$ in $Z=R^m$ by the correspondence

$$y_0=\pi\leftrightarrow(0, 0, \ldots, 0)$$
$$y_i \quad \leftrightarrow(z_1, \ldots, z_m)$$

with $z_j=\delta_{ij}$: that is, $z_i$ is 1 and all other components are zero. The final stage will perform an 'inverse' of this mapping by:

$$y=y_i \text{ if } z_i\geqslant\tfrac{1}{2} \text{ and } z_i>z_j \text{ all } j\neq i.$$

The pattern belongs to the $i$th classification if $z_i$ is the largest component of $z$, provided that $z_i$ also exceeds $\frac{1}{2}$.

The central stage computes $m$ numbers (the $z_i$) from the $n$ characterizing measurements of the pattern. In the ideal situation these $m$ numbers would be such that if the pattern belongs to the $i$th class, the $i$th number is one and all the rest are zero. Because the central stage will only be an approximation to the ideal, the $m$ numbers are 'cleaned up' by the final stage.

The parameters of the central canonical stage can be estimated using the techniques of section 3, sampling the human performance randomly under normal operating conditions. In data preparation this might be done as follows: a prototype paper handling machine with first stage (optics plus retina) would read typical data sheets, the resulting set of $n$ measurements being relayed to the learning equipment (figure 6) together with correct classification [this could be determined by some human operator of the prototype: faulty (noisy) human classification here would not be critical, for the algorithms learn the average of noisy functions (Aizerman, Braverman and Rozonoer 1964a, Kashyap and Blaydon 1966)]. Each component of $z$ is treated independently and the parameters are estimated for minimum mean square error from the ideal.

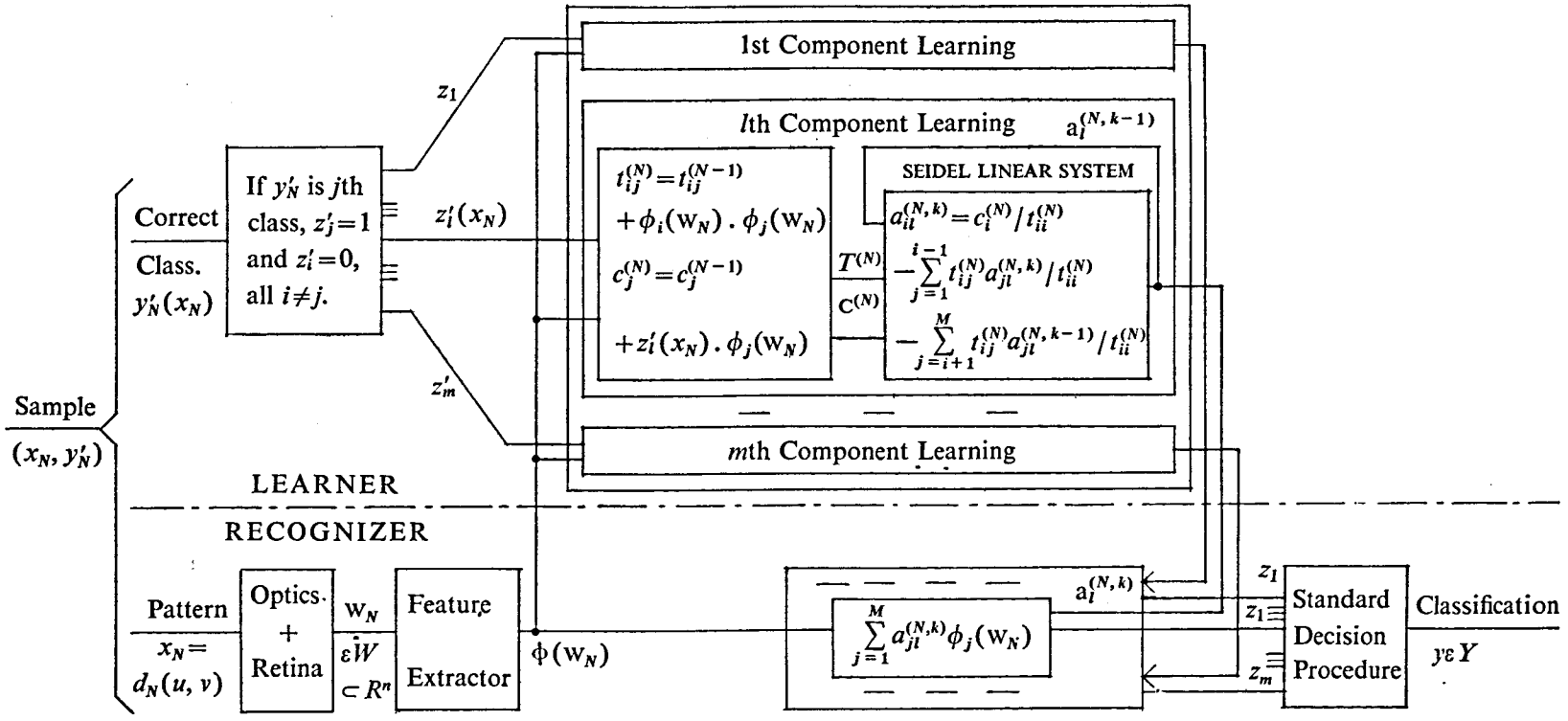



Figure 6. Visual pattern recognition. (Learning, Seidel method).

The functions $\{\phi_i\}$ appearing in the approximation in the canonical central stages can be interpreted as 'features' in the way that is standard practice in Pattern Recognition (Hall 1968, Nagy 1968, Nilsson 1965, Sebestyen 1962). These feature functions need not necessarily be of the intuitive kind: they can simply be any functions that are easy to realize physically, such as polynomials, or Walsh functions (Wiener 1958), in digital realizations.

Having fixed the functions $\{\phi_i\}$ to be used, and having estimated the parameters, we must then assess the performance of the machine. A crude bound to the error-rate is given by four times the mean square error – which can be obtained at the same time as learning (section 3.4): if this indicates success, we are all right. If this indicates failure, a more refined test using separate samples may indicate success – if failure, we must then consider more training samples or more parameters or both.

## SUMMARY LIST OF PRINCIPAL SYMBOLS USED

$S$ – system defined by an input–output mapping, $f$.
$X$ – input set or space.
$Y$ – output set or space.
$f$ – mapping of $X$ into $Y$.
$p$ – operating distribution, probability measure defined on $X$.
$Q$ – sampling procedure for generating samples $x_k$ from $X$.
$(x_k, y_k)$ – sample from input–output mapping $f$. $y_k = f(x_k)$.
$\phi(a)$ – parametrized mapping of $X$ into $Y$, parameters $a_i$.
$e(f_1, f_2)$ – error functional to measure difference between two functions mapping $X$ into $Y$.
$M$ – total number of parameters.
$N$ – total number of samples.
$\varepsilon$ – prescribed error criterion, below which error $e(f, \phi)$ must come.
$a_i^{(N)}$ – $N$-sample estimate of parameter $a_i$.
$T$ – $M \times M$ matrix with elements $t_{ij} = \int_X \phi_i \phi_j \, dp$.
$c$ – $M$-vector of 'experience', $c_j = \int_X f \phi_j \, dp$.
$T^{(N)}$ – $N$-sample estimate of $T$.
$c^{(N)}$ – $N$-sample estimate of $c$.
$U^{(N)}$ – biased $N$-sample estimate of $T$; $U^{(N)} = T^{(N)} + I$.
$e$ – error between $f$ and $\phi(a)$ with correct least squares parameters.
$e_N$ – error when using $N$-sample estimates of the least squares parameters.
$e_N^{(N)}$ – estimate of $e_N$ using same $N$ samples as for $a_i^{(N)}$.
$d(\xi_1, \xi_2)$ – light intensity at a point in a 2 dimensional visual pattern.
$\pi$ – null-classification.
$n$ – number of retinal measurements (mosaic size, or number of functions in an approximation of $d(\xi_1, \xi_2)$).
$m$ – number of categories, excluding null-category.
$Z$ – $m$-dimensional vector space $R^m$ in which $Y$ is imbedded.
$z_i$ – output of classifier (before threshold) for $i$th catgeory.

**Acknowledgements**

I should like to record my thanks to Professor C.A. Rogers of the Department of Mathematics, University College London, and to Dr W.E. Taylor of the Department of Electrical Engineering, University College, London, for discussions and criticisms which have helped in the formulation of this paper; and to the Science Research Council for support during the period in which this research was carried out.

## REFERENCES

Achieser, N. I. (1956) *Theory of approximation.* New York: Ungar.

Aizerman, M.A., Braverman, E.M. & Rozonoer, L.I. (1964a) The theoretical foundations of the method of potential functions in the problem of teaching automata to classify input situations. *Automation and Remote Control*, **25**, 821–38.

Aizerman, M.A., Braverman, E.M. & Rozonoer, L.I. (1964b) The probablistic problem of teaching automata to recognize classes and the method of potential functions. *Automation and Remote Control*, **25**, 1175–91.

Aizerman, M.A., Braverman, E.M. & Rozenoer, L.I. (1964c) The method of potential functions for the problem of restoring the characteristics of a function converter from randomly observed points, *Automation and Remote Control*, **25**, 1546–54.

Aizerman, M.A., Braverman, E.M. & Rozonoer, L.I. (1965) The Robbins – Monro process and the method of potential functions. *Automation and Remote Control*, **26**, 1882–6.

Chien, Y.T. & Fu, K.S. (1967) On Bayesian learning and stochastic approximation. SSC-3, 28–38.

Cuenod, M. & Sage, A.P. (1968) Comparison of some methods used for process identification. *Automatica*, **4**, 235–69.

Devyaterikov, I.P., Propoi, A.I. & Tsypkin, Ya.Z. (1967) Iterative learning algorithms for pattern recognition. *Automation and Remote Control*, **28**, 108–17.

Faddeeva, V.N. (1959) *Computational methods of linear algebra.* New York: Dover.

Hall, P.A.V. (1968) Pattern classification as interpolation in $N$ dimensions. *Computer J.*, **11**(3), 287–92.

Hall, P.A.V. (1969) Generalisation of Wiener's Theory of Non-linear Systems for Process Identification. *IEEE Trans. on Automatic Control*, AC/**14**, 312–13.

Hammersley, J.M. & Handscomb, D.C. (1964) *Monte Carlo Methods.* London: Methuen.

Handscomb, D.C. (1966) *Methods of numerical approximation.* Oxford: Pergamon Press.

Kashyap, R.L. & Blaydon, C.C. (1966) Recovery of functions from noisy measurements taken at randomly selected points, and its application to pattern recognition. *Proc. IEEE (Letters)*, **54**, 1127–9.

Kendall, M.G. & Stuart, A. (1961) *The advanced theory of statistics, 2, Inference and Relationship.* London: Charles Griffin and Co.

Kline, S.J. (1965) *Similitude and Approximation Theory.* New York: McGraw-Hill.

Lanczos, C. (1957) *Applied Analysis.* London: Pitman.

Nagy, G. (1968) The state of the art in pattern recognition. *Proc. IEEE*, **56**, (5), 836–62

Nilsson, M. J. (1965) *Learning Machines.* New York: McGraw-Hill.

Sebestyen, G.S. (1962) *Decision making processes in pattern recognition.* London and New York: Macmillan.

Sz.-Nagy, B. (1962) *Introduction to real functions and orthogonal expansions.* New York: Oxford University Press.

Wagner, T. (1968) The Rate of Convergence of an Algorithm for Recovering Functions from Noisy Measurements Taken at Randomly Selected Points. *IEEE Trans. on Sys. Sci. and Cyb., SSC* **4**, 151–4.

Wiener, N. (1958) *Non-linear problems in random theory.* Cambridge, Mass.: MIT Press.

# PRINCIPLES FOR DESIGNING INTELLIGENT ROBOTS

# 27

# Planning and Robots

James Doran*
Department of Machine Intelligence and Perception
University of Edinburgh

## Introduction

This paper is a survey and discussion of research work relevant to the task of constructing some kind of reasoning robot. The emphasis is entirely on the organization of the reasoning processes, in particular planning, rather than on hardware. In practice the reasoning would most probably be carried out within a digital computer.

My objective is to clarify the relationship between some superficially rather disparate approaches to this task, and simultaneously to indicate what seem to be the key problem areas.

No new experimental results are presented, but the approach to the subject which I have adopted is a consequence of earlier experimentation with a simple computer simulation of a robot (Doran 1968a, 1969).

## Heuristic problem-solving programs

Our starting point is heuristic problem-solving by computers. The following outline of the present state of the subject is very brief, but sufficient for our purposes. For more detailed information the reader may refer to the valuable survey by Sandewall (1969).

Heuristic problem-solving has been dominated over the last decade by what may be called the General Problem Solver (GPS) tradition (Newell, Shaw and Simon 1959, 1960, Hormann 1965, Ernst and Newell 1969, Quinlan and Hunt 1968). The various versions of the GPS program, its descendants, and the associated proposals and speculations, form an impressive body of research. However, although a great deal has been learnt from the GPS work about how to write a single computer program to solve a variety of problems, actual performance has always been somewhat disappointing, and the work has sometimes been marred by a certain vagueness of presentation. Typically the GPS can, with a substantial amount of human aid, solve quite a wide range of puzzles of the 'party-game' variety, for example the 'Tower of Hanoi'

* Present address: S.R.C. Atlas Computer Laboratory, Didcot, Berkshire.

and 'Missionaries and Cannibals' problems, together with very simple problems in potentially much more complex task areas such as the integral calculus.

Another substantial body of work on general problem-solving is that associated with the Graph Traverser program (Doran and Michie 1966, Doran 1967, Michie 1967, Doran 1968, Michie, Fleming and Oldfield 1968, Michie and Ross 1970). Work on the Graph Traverser has tended to be rather more concrete, and less ambitious, than that on the GPS, and has not claimed to have psychological significance. The performance of the program in practice is rather better. In the next secion I shall outline the basic Graph Traverser algorithm and indicate where the GPS differs from it.

Important projects devoted to much more specific applications of heuristic problem-solving are those of Evans (1964) and Moses (1967). The task areas are geometric-analogy problems and symbolic integration respectively, and the performance level attained is quite high.

At the present time several new trends are appearing:

(a) heuristic problem-solving of the GPS or Graph Traverser type is sufficiently familiar to be embedded in more complex programs or programming languages (for example, Burstall 1968, Popplestone 1970),

(b) the theory, rather than the practice, of heuristic problem-solving is being explored (for example, Ernst 1968, Nilsson 1969, Banerji 1969, Sandewall 1969a, Pohl 1970),

(c) the problem of automating the process of finding a good problem representation is being explored (for example, Amarel 1968, 1969),

(d) automatic theorem-proving procedures are being applied to problem-solving (for example, Green 1969).

All these trends have some relevance to the remainder of this paper.

#### Problem-solving and planning by robots

In this section and the next we shall consider the transition from heuristic problem-solving as exemplified by the Graph Traverser, to planning by a robot as exemplified by my own work and that of Marsh (Doran 1967, 1967a, 1968a, 1969; Marsh 1970; Michie 1967, 1968a; Popplestone 1967). I do not suggest that the Graph Traverser is the only starting point for such work. I use this example because I am particularly well acquainted with it.

The primary components of the Graph Traverser 'schema' are as follows. The problem to be solved must be capable of representation as a graph, in the mathematical sense, where the nodes of the graph correspond in some sense to possible *states* of the problem and the arcs to possible operations or *operators* which transform one state into another. This problem graph is defined to the program by specifying the set of possible states and the set of possible operators and their effects. It must then be possible to interpret the original problem as either the task of finding a path across the graph from one particular node (the starting state) to another (the goal state), or the

task of finding a node with some desired property. In this paper we shall always be concerned with the former task.

The solution procedure of the Graph Traverser is actually to 'grow', in the computer store, a larger and larger portion of the problem graph in the form of a *search tree* rooted at the starting node. Growth continues until the node or path sought is discovered. In order to do this efficiently the program uses, in general, a heuristic state evaluation function and heuristic operator selection techniques to grow the search tree in the most promising direction. Full details of this process can be obtained from the references quoted.

The GPS diverges substantially from the Graph Traverser by laying great stress on the difficulty that may arise in applying a selected operator to a particular state, and the way in which a recursive call of the program can be used to deal with this difficulty. The converse of this is that the GPS pays less attention to holding in store a ramified search tree and to the use of a state evaluation function.

Obviously, there is more to the Graph Traverser than I have described, and the interested reader is again referred to the source papers. We shall, however, need here one further concept, that of a *partial solution path.* When the program fills the available computer store with its search tree without finding a solution path, then it commits itself to all or part of a particular branch of the tree as the first part of the required path. This enables it to discard part of the search tree and to re-use the store thus freed.

A typical application of the Graph Traverser is to some kind of sliding-block puzzle, where the states correspond to possible configurations of the pieces of the puzzle, and the operators to possible movement of the pieces.

The transition from a problem-solving context to that of planning by a robot is a simple one, at least at first sight. By planning we mean the robot's need to consider possible courses of action in the light of some desired goal, and to select and carry out the most promising. Hence all one need do is to identify the concept of a state, in Graph Traverser terminology, with some state of the robot and its environment, and to identify the concept of an operator with some possible act on the part of the robot. A solution path, once discovered, is then a plan of action which the robot must execute rather than simply exhibit to the outside world as in the problem-solving case.

My own work, which developed out of some of the earlier Graph Traverser work, simulated a robot living in a very simple 'cage' environment. A state, better called a *perceived state* in this case, corresponded to the little that the robot could perceive of its surroundings at any time, and an operator to one of a set of simple movements (stepping, turning to the left or right). A simple motivational system provided goal states.

The *planning tree*, corresponding to the Graph Traverser search tree, was grown by a simple depth first procedure with heuristic cut-off, partly because a state evaluation function is much less useful as a means of directing the growth of the tree in a robot situation. However, a simple evaluation function

was used, with other heuristic procedures, to select partial solution paths, that is to say partial plans, when a full plan could not be found.

Rather more important was the incorporation into the planning tree of a form of 'expectimaxing' (*see* Michie and Chambers 1968). The need for this arose because of ambiguity in the robot's perceptual input. Specifically, any particular perceived state might well be the outcome of a variety of possible relationships between the robot and its environment. This led to a degree of uncertainty on the part of the robot as to what the outcome of any particular act might be. The robot tried to cope with this uncertainty by evaluating plans in terms of their predicted average benefit, using an expectimaxing procedure.

### Planning and learning

The simulated robot described in the previous section could have had built into it detailed information about the effects of its possible movements upon its perceptions, in the same way that the Graph Traverser program is told what is the immediate effect of any operation on any configuration of pieces of a sliding-block puzzle. In fact, it was expected to collect such information from experience. Thus, in this case, planning was very closely associated with learning. The following types of learning occurred in the system:

(a) learning of the relationship between acts and perceptions by noting the effects of individual acts, by making generalizations about the effects of acts, and by noting that certain complicated transitions from one perceived state to another can always be achieved,

(b) learning which acts to employ in particular situations and the benefits to be expected – a kind of habit formation.

These learning capabilities can be regarded as a rather complex form of the rote learning employed by Samuel in his checkers programs (Samuel 1959, 1967). They by no means exhaust the possibilities for learning in such a system. Much more powerful forms of generalizations and abstraction are needed, and some degree of self-optimization of the variety of parameter settings controlling the planning would be possible. Any work on learning in heuristic problem-solving systems is potentially relevant (Quinlan 1969, Michie and Ross 1970). Nor can strategies for discarding information, for 'forgetting', be ignored. They may be crucial to success.

The whole question of learning in such a Graph Traverser based robot control system has been tackled from a slightly different angle by Marsh (1970). Here the learning aspects of the system have been clarified and isolated by the use of 'memo-functions' (Michie 1967a, 1968, Popplestone 1967). Marsh has also obtained experimental results quantifying the benefit to be obtained by using memo-functions at different points in the planning process.

Before leaving this section we should note two other areas in which such

a system will ultimately have to show learning ability. These are time and attention.

A robot, if it is to exist in a dynamic environment, must be able to estimate the passage of time, to allow for the time it takes to form and execute plans (*see* Toda 1962), and to estimate how a complex dynamic situation (for example, traffic at a road junction) will develop in time.

A robot must also be able to decide which sensors to read when. The point here is that a system with any degree of serial reasoning cannot be attending to the input from all of its sensors all of the time. It follows that the robot's attention strategy is a highly important part of its planning process. Too little attention to the outside world could lead to sudden disaster, confusion, or, more subtly, to a growing misconception by the robot of just what its situation actually is! Too much attention could waste time and hinder action.

#### Parallel processing

When a planning system of the type which we have been discussing is required to learn from experience the effects of the acts available to it, then it will build up a network of state–act–state transitions in its memory. This network is the equivalent of the problem graph in the problem-solving situation, but unlike the problem graph, it is actually kept as a network of data-structures and pointers in the computer store, at least until specific compression mechanisms go to work.

Planning is then not a question of reconstructing a fragment of this memory network, but of tracing out the planning tree on the already constructed network. This means that any planning algorithm which uses state evaluation followed by 'backing-up' of values, for example, expectimaxing, can plausibly be reinterpreted as a network flow process of the following general type:

(1) 'excitation' is inserted into the network at the goal state, and
(2) continuously transmitted by the arcs,
(3) continuously redistributed by the nodes, and
(4) detected by a receiver at the node corresponding to the robot's current state, which
(5) implements the act corresponding to the incoming arc carrying the greatest excitation.

A system of this type, the STELLA learning machine, has been explored in detail by Andreae (*see* Andreae and Cashin 1969) though in practical simulation work the parallel aspects of the machine had to be simulated on a serial computer. The model of animal learning behaviour put forward by Deutsch (1964) also has this general form, but is less precisely specified than the STELLA automaton.

In each of these two examples the networks through which excitation

flows are formed by receptor–motor units which automatically link up in accordance with the behaviour of the system's perceptual environment.

Network flow systems are attractive partly because they seem 'natural' in some sense, and partly because once one postulates a parallel processing capability of this type, the problems associated with the computational load and organizational complexity of major tree searches promise to disappear. However, it is far from clear how such network systems can be persuaded to yield really complex behaviour. The results of the work on perceptrons and other such self-organizing systems do not encourage optimism (Minsky and Papert 1969).

We should also note two important research projects which involve parallel processing and robot control, though not planning. These are the simulation studies on instinctive behaviour by Friedman (1969) and on the function of the reticular formation in the vertebrate by Kilmer and others (Kilmer, McCulloch and Blum 1969).

#### Complex planning

So far I have used the word 'planning' to refer to a robot's ability to consider alternative sequences of acts and to select the most promising among them. A close parallel has been drawn with the operation of a particular heuristic problem-solving program, the Graph Traverser.

However, in the context of heuristic problem-solving the word 'planning' has almost invariably been used for some more complex procedure *added* to the basic heuristic search in order to make it more effective. I shall employ the term *complex planning* for such additional procedures.

One form of complex planning is the use of *macro-operators*. What happens is that the original problem graph has added to it new arcs corresponding to concatenations of the original operators. These macro-operators are then used in the usual way to help grow the search tree. Clearly one needs some way of automatically constructing such operators, and equally some way of decomposing them when they appear in a solution path or plan.

The use of macro-operators in heuristic problem-solving has been tried out by, among others, Travis (1964), Hormann (1965) and Michie (1967). The concept is closely related to the mathematical concepts of a lemma and a theorem. Examples of the use of macro-operators in robot simulation work are provided by Nilsson and Raphael (1967) and Doran (1969). In each of these two latter examples the situation is complicated by the fact that the definitions of the macro-operators, or better macro-acts, are not available at plan execution time, so that when such an act is to be carried out a complex sub-process is set in motion.

The use of macro-operators is only one rather simple form of complex planning. The planning procedure proposed for the GPS (Newell, Shaw and Simon 1959) envisaged the abstraction both of the states and of the operators of the original problem formulation, in order to create a new simplified

version of it. The solution to this simple problem would then guide the solution of the original problem. Minsky(1961) has discussed complex planning of this 'homomorphic model' type, and has stressed the potential reduction in total search effort to be won. In the same paper he has also considered the use of 'semantic models' as a form of complex planning in a mathematical context. The successful geometry theorem-proving program of Gelernter (1959), which used a 'diagram' to test the validity of propositions, is a well-known example of this form of planning.

Recently Sandewall (1969) has defined a Planning Problem Solver (PPS). This is an attempt to explore in detail complex planning of the 'homomorphic model' type as applied to the GPS. His lengthy discussion covers the use of what I have called 'expectimaxing' at the complex planning level (compare Minsky's remarks, 1961), and indicates how concepts drawn from board game-playing programs such as $\alpha$-$\beta$ pruning can be carried over to this situation. He also proves certain optimality results for the PPS.

Sandewall's optimality results take us some way towards answering the very general question: 'When is complex planning beneficial, in the sense that it reduces the total amount of computation needed to solve the problem (or find a plan), and when is it more trouble than it is worth?' The answer to this question is clearly bound up with the problem of characterizing and actually finding good complex planning models for particular problems. And this problem itself is very closely related to the problem of problem representation as treated by Amarel (1968, 1969) and others.

#### Planning and formal systems

The present Stanford Research Institute (SRI) hardware robot project (Nilsson 1969a), not to be confused with the preliminary simulation work mentioned earlier, has experimented with more than one *ad hoc* heuristic search method for planning. These have made use of the 'map' or 'grid model' with which the robot is provided.

Of greater interest is the use of a first-order predicate calculus resolution theorem-prover to control the robot, as described by Green (1969). In barest outline, the procedure followed in order to have the robot carry out some task is to require the theorem-prover to prove the theorem that a situation can exist in which the task has been completed. During the proof procedure the theorem-prover constructs the sequence of acts, that is the plan, which the robot must execute to perform the task. Everything which the robot has to know in order to perform the task is formulated by the experimenter as a set of axioms (the 'axiom model') within the formal language of predicate calculus. Achieving this formulation may itself be far from straightforward, and at the present time the system can cope only with very simple tasks. In his paper Green discusses such complications as acts with more than one possible outcome, and tasks such that the robot's plan must include making some observation and using its result.

Green's work falls broadly within the Advice Taker tradition. The Advice Taker was proposed by McCarthy (1959) as a program capable of 'common-sense' reasoning, and capable of accepting and using advice whilst solving a problem. The program was to reason by manipulating sentences in formal languages, and therefore would have a very powerful means of storing and using its knowledge of the world. As a means to this end, McCarthy advocated the formalization of such everyday concepts as situation, causality, ability, and knowledge.

From this initial impetus a great deal of valuable work has resulted. However some of the more philosophical problems at the heart of the Advice Taker project still seem to defy any very coherent solution (McCarthy and Hayes 1969).

Since the Advice Taker work is much concerned with reasoning about actions and what they can achieve, it is not surprising that the gulf between this work and, say, the GPS work is less wide than at first appears (Hubermann 1965). For example, the concepts of situation, fluent, action, and strategy discussed by McCarthy and Hayes (1969) correspond roughly to the concepts of a state, a property of a state, an act, and a plan as used in this paper. It remains to be seen whether the attempt to formalize such concepts within a coherent logical system, rather than merely to incorporate them within some actual computer program or hardware device, is the only or even the best way to make progress.

### Plans and programs

McCarthy and Hayes (1969) explore in some detail the parallel between a plan or course of action and a computer program. Their aim is partly to have procedures which prove results about computer programs do the same for plans.

Almost all programs involve loops, where a certain sequence of operations is repeated until some test is satisfied. One is led to consider plans with a similar property. The 'San Diego' problem, attributed to McCarthy in this context, is illuminating. The problem is to formulate a plan by which a motorist can get from San Francisco, say, to San Diego given the following information:

(a) there is no map obtainable with both San Francisco and San Diego marked on it,

(b) any filling station will provide a map of its local area, which will both indicate the direction of San Diego and show the location of at least one other filling station in that direction,

(c) the motorist is now at a filling station in San Francisco.

The solution to the problem is the following plan:

'Collect a map from the filling station you are at and drive to the filling station shown on the map which is furthest in the direction of San Diego. *Repeat* this procedure *until* you arrive in San Diego.'

Clearly the plan is one big loop and test. Equally clearly no planning procedure of the Graph Traverser type is going to generate such a plan (Michie and Popplestone 1969).

A different approach to this general topic is that of the psychologists, Miller, Galanter, and Pribram (1960), in their well-known book *Plans and the Structure of Behaviour*. They also point to the parallel between a computer program and a plan (*loc. cit.*, p. 16). Their TOTE unit (Test–Operate–Test–Exit unit), which they propose as the fundamental building block of plans, is just the 'loop and test' combination we are considering. For example, they exhibit a plan for knocking a nail into a plank which, simplifying somewhat, is '*repeat* the operation of hitting the nail with the hammer *until* its head is flush with the surface of the plank'.

Unfortunately, the discussion of Miller and his colleagues stops well short of an automatic procedure for generating such plans. It does imply, however, that plan generation should involve loops from the outset. Does this mean that we should abandon informal problem-solving methods for planning, and instead turn to the work concerned with the automatic generation of computer programs (*see*, for example, Green 1969) ? The prospect is not too inviting for those with an interest in human as well as machine intelligence.

What might be a 'natural' way to solve the San Diego problem – that is, a way which common sense and introspection suggest might be the way in which a person might solve it? By trying to answer this question we may obtain some productive new ideas. Consider the following procedure:

(1) visualize a very crude outline map of California with San Diego and San Francisco marked, and also with the motorist's location marked (initially at San Francisco),

(2) consider possible sequences of 'relevant' acts for the motorist – driving to filling stations, buying maps, and so on – manipulating his location on the image map as seems appropriate,

(3) observe that one possible sequence of acts has two interesting properties – that the acts form a repeating pattern and that the sequence corresponds to a roughly linear motion of the motorist's location on the image map,

(4) by extrapolating the motion on the map observe that if the repetition of acts is continued, then the motorist will reach San Diego,

(5) convert this observation into the required plan.

The reader may or may not feel that this solution procedure is plausible or in any way illuminating. He should compare Amarel's analysis of the 'mutilated chequer-board' problem (Amarel 1969). It does suggest that the key steps in the solution of such a problem may be abstraction followed by a rather simple form of extrapolation, in this case extrapolation of motion in a visual image.

### Models and percepts

A robot must acquire or be given knowledge about its environment and itself.

This knowledge is often called its 'world model'. Examples of different kinds of world model which we have encountered are:

(a) the SRI robot's *axiom model* for use by the resolution theorem-prover (Nilsson 1969a),

(b) the SRI robot's *grid model* which can loosely be described as a 'map' of its surroundings (Nilsson 1969a),

(c) *perceptual cause and effect models* which store perceived states and the way in which they are modified by acts (Deutsch, 1964; Doran, 1968, 1969; Andreae and Cashin, 1969).

We can reasonably add to this list:

(d) *semantic memory systems* of one kind or another (for example, Quillian 1969, Becker 1969),

(e) *belief systems* such as those developed by Colby and his co-workers (Colby and Smith 1969).

The precise relationships between these superficially very different ways of storing and using knowledge about the world have not been much explored. I have two minor comments:

(1) Colby's term 'belief system' should perhaps be rather more widely used. It has advantages compared with 'world model' in that it avoids any association with physical 'scale' models, and correctly implies that a robot's 'knowledge' is bound in practice to include approximations to the truth as well as outright errors. It also encourages consideration of the 'conviction' with which a robot should hold a belief.

(2) A key heuristic problem for the robot is that of selecting the beliefs *relevant* to the prediction of the outcome of any particular course of action. This problem has been discussed by, for example, Nilsson and Raphael (1967, p. 243) and McCarthy and Hayes (1969 – the 'frame problem').

I shall end with some speculative remarks to be associated with entry (c) in the foregoing list – perceptual cause and effect models. These remarks concern reasoning processes which operate in terms of perceptual images or 'percepts'. We are all familiar with subjective perceptual images and, for example, some people seem to 'think' in terms of visual images far more than others. What does reasoning in terms of perceptual images mean, if anything, when we are talking about a robot?

Suppose that a robot is observing, through a TV camera, two cylinders standing upright on a flat surface. We may suppose that, as the result of a complex piece of picture processing (possibly involving the robot's expectation of what is to be seen, as well as what is actually there) a data structure is generated, representing in some sense the important information in the observation. This abstracted picture we may reasonably call a visual percept. If the robot now stores away this percept as it is, then it has also implicitly stored away such beliefs as: 'the tall object is wider than the short object'. However the robot may continue the processing and formulate and store

derived beliefs in some quite different coding, for example, a predicate calculus axiom model. Since this coding does not arise naturally from the picture processor, it is reasonable that it should not be called perceptual.

In human begins we can point to the distinction between, on the one hand. a fuzzy abstracted *visual* image of a tall wide object beside a short narrow object and, on the other hand, a fuzzy abstracted *aural* image of the words: 'the tall object is wider than the short object'. This is a rather special example because the coding into which the visual percept is translated itself involves perceptual (aural) images.

What form might reasoning in terms of visual percepts take? One possibility has already been indicated in connection with the San Diego problem. The key process might be as follows. Given two such percepts, and constraints which effectively specify what constitutes a valid visual percept (compare Clowes 1969), it will often be possible to merge them to form a third. The information implicit in the new percept will then have been deduced, in some sense, from the information implicit in the originals. Experimental psychologists have studied examples of such reasoning (Huttenlocher 1968, Handel, London and DeSoto 1968).

I am not suggesting that what I have called perceptual reasoning could not be encompassed, in theory, within some suitable formal system (compare Hayes 1970). What I am suggesting is that it might have practical advantages for robot systems operating in the real world.

**Acknowledgements**

This paper has been written during my tenure of a Science Research Council Fellowship. I have benefited from many discussions with Professor Donald Michie and Dr R. Burstall, both of the Department of Machine Intelligence and Perception, University of Edinburgh

## REFERENCES

Amarel, S. (1968) On representations of problems of reasoning about actions. *Machine Intelligence 3*, pp. 131–72 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Amarel, S. (1969) Problem solving and decision making by computer: an overview. Paper distributed at the Symposium on Cognitive Studies and Artificial Intelligence Research, 2–8 March 1969. University of Chicago Center for Continuing Education.

Andreae, J.H. & Cashin, P.M. (1969) A learning machine with monologue. *Int. Journal of Man–Machine Studies*, **1**, 1–20.

Banerji, R. B. (1969) *Theory of problem solving*. New York: Elsevier.

Becker, J. D. (1969) The modelling of simple analogic and inductive processes in a semantic memory system. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 655–68 (eds Walker D. E. & Norton, L.M.). New York: Association for Computing Machinery.

Burstall, R.M. (1968) Writing search algorithms in functional form. *Machine Intelligence 3*, pp. 373–86 (ed. Michie D.). Edinburgh: Edinburgh University Press.

Clowes, M.B. (1969) Pictorial relationships – a syntactic approach. *Machine Intelligence 4*, pp. 361–84 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Colby, K.M. & Smith, D.C. (1969) Dialogues between humans and an artificial belief system. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 319–24 (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Deutsch, J.A. (1964) *The Structural Basis of Behavior*. Cambridge: University Press.

Doran, J.E. (1967) An approach to automatic problem-solving. *Machine Intelligence 1*, pp. 105–23 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Doran, J.E. (1967a) Designing a pleasure-seeking automaton. *Research Memorandum MIP–R–28*. Department of Machine Intelligence and Perception, University of Edinburgh.

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.

Doran, J.E. (1968a) Experiments with a pleasure-seeking automaton. *Machine Intelligence 3*, pp. 195–215 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Doran, J.E. (1969) Planning and generalization in an automaton/environment system. *Machine Intelligence 4*, pp. 433–54 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Doran, J.E. & Michie, D. (1966) Experiments with the Graph Traverser program. *Proc. R. Soc. A*, **294**, 235–59.

Ernst, G.W. (1968) Sufficient conditions for the success of GPS. *Memorandum SRC–68–17*. Systems Research Center, Case Western Reserve University.

Ernst, G.W. & Newell, A. (1969) *GPS: a Case Study in Generality and Problem Solving*. ACM Monograph Series. New York: Academic Press.

Evans, T.G. (1964) A heuristic program to solve geometric-analogy problems. *AFIPS*, **25**, 327–38. SJCC, Baltimore: Spartan Books.

Feigenbaum, E.A. & Feldman, J. (eds) (1963) *Computers and Thought*. New York: McGraw-Hill.

Friedman, L. (1969) Robot control strategy. *Proceedings of the International Joint Conference on Artificial Intelligence* (eds Walker, D.E. & Norton, L.M.) pp. 527–40. New York: Association for Computing Machinery.

Gelernter, H. (1959) Realization of a geometry-theorem proving machine. *Proceedings of an International Conference on Information Processing*, pp. 273–82. Paris: UNESCO House. Reprinted in Feigenbaum and Feldman (1963).

Green, C. (1969) Application of theorem proving to problem solving. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 219–39 (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Handel, S., London, M. & DeSoto, C. (1968) Reasoning and spatial representations. *J. of Verbal Learning and Verbal Behavior*, **2**.

Hayes, P.J. (1970) Robotologic. *Machine Intelligence 5*, pp. 533–54 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Hormann, A. (1965) Gaku: an artificial student. *Behav. Sci.*, **10**, 88–107.

Hubermann, B. (1965) The Advice Taker and GPS. *Research Memorandum No. 33*. Stanford Artificial Intelligence Project, Stanford University.

Huttenlocher, J. (1969) Constructing spatial images: a strategy in reasoning. *Paper distributed to the Annual Conference of the British Psychological Society*.

Kilmer, W.L., McCulloch, W.S. & Blum, J. (1969) A model of the vertebrate central command system. *Int. J. Man–Machine Studies*, **1**, 279–309.

Marsh D. (1970) Memo functions, the Graph Traverser and a simple control situation. *Machine Intelligence 5*, pp. 281–300 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

McCarthy, J. (1959) Programs with common sense. *Proceedings of a Symposium on the the Mechanization of Thought Processes*, pp. 75–91. London: HMSO. Reprinted in Minsky (1968).

McCarthy, J. & Hayes, P.J. (1969) Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, pp. 463–502 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Michie, D. (1967) Strategy-building with the Graph Traverser. *Machine Intelligence 1*, pp. 137–54 (eds Collins, N.L. & Michie D.). Edinburgh: Oliver and Boyd.

Michie, D. (1967a) Memo functions: a language feature with 'rote-learning' properties. *Research Memorandum MIP–R–29*, Department of Machine Intelligence and Perception, University of Edinburgh.

Michie, D. (1968) 'Memo' functions and machine learning. *Nature*, **218**, 19–22.

Michie, D. (1968a) Information and behaviour: a commentary on a note by R.L. Gregory. *Research Memorandum MIP–R–37*, Department of Machine Intelligence and Perception, University of Edinburgh.

Michie, D. & Chambers, R.A. (1968) BOXES: an experiment in adaptive control. *Machine Intelligence 2*, pp. 137–52 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.

Michie, D., Fleming, J.G. & Oldfield, J.V. (1968) Comparison of heuristic, interactive and unaided methods of solving a shortest route problem. *Machine Intelligence 3*, pp. 245–55 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Michie, D. & Popplestone, R.J. (1969) Freddy's first three years. *Experimental Programming 1968–1969*. Department of Machine Intelligence and Perception, University of Edinburgh.

Michie, D. & Ross, R. (1970) Experiments with the adaptive Graph Traverser. *Machine Intelligence 5*, pp. 301–18 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Miller, G.A., Galanter, E. & Pribram, K.H. (1960) *Plans and the Structure of Behavior*. New York: Holt, Rinehart and Winston Inc.

Minsky, M. (1961) Steps towards artificial intelligence. *Proc. Inst. Radio Engineers*, **49**, 8–30. Reprinted in Feigenbaum and Feldman (1963).

Minsky, M. (ed.) (1968) *Semantic Information Processing*. Cambridge Mass. and London: MIT Press.

Minsky, M. & Papert, S. (1969) *Perceptrons*. Cambridge Mass. and London: MIT Press.

Moses, J. (1967) *Symbolic Integration*. Ph.D. dissertation, MIT Mathematics Department.

Newell, A., Shaw, J.C. & Simon, H.A. (1959) Report on a general problem-solving program. *Proceedings of an International Conference on Information Processing*, pp. 256–64. Paris: UNESCO.

Newell, A., Shaw, J.C. & Simon, H.A. (1960) A variety of intelligent learning in a general problem solver. *Self-organizing Systems*, pp. 153–89 (eds Yovits, M.C. & Cameron, S.). London: Pergamon Press.

Nilsson, N.J. (1969) Searching problem-solving and game-playing trees for minimal cost solutions. *Proceedings of the IFIP Congress* 1968. Amsterdam: North Holland.

Nilsson, N.J. (1969a) A mobile automaton: an application of artificial intelligence techniques. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 509–20 (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Nilsson, N.J. & Raphael, B. (1967) Preliminary design of an intelligent robot. *Computer and Information Sciences II*, pp. 235–59 (ed. Tou, J.). New York: Academic Press.

Pohl, I. (1970) First results on the effect of error in heuristic search. *Machine Intelligence 5*, pp. 219–36 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Popplestone, R.J. (1967) Memo functions and the POP-2 language. *Research Memorandum MIP–R–30*. Department of Machine Intelligence and Perception, University of Edinburgh.

Popplestone, R.J. (1970) Experiments with automatic induction. *Machine Intelligence 5*, pp. 203–16 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Quillian, M.R. (1969) The teachable language comprehender: a simulation program and theory of language. *Comm. Ass. comput. Mach.*, **12**, 459–76.

Quinlan, J.R. (1969) A task-independent experience-gathering scheme for a problem-solver. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 193–7 (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Quinlan, J.R. & Hunt, E.B. (1968) A formal deductive problem-solving system. *J. Ass. comput. Mach.*, **15**, 625–46.

Samuel, A.L. (1959) Some studies in machine learning using the game of checkers. *IBM J. of Res. and Dev.*, **3**, 211–29. Reprinted in Feigenbaum and Feldman (1963).

Samuel, A.L. (1967) Some studies in machine learning using the game of checkers, 2 – recent progress. *IBM J. of Res. and Dev.*, **11**, 601–17.

Sandewall, E.J. (1969) Concepts and methods for heuristic search. *Proceedings of the International Joint Conference on Artificial Intelligence* (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Sandewall, E.J. (1969a) A planning problem solver based on look-ahead in stochastic game trees. *J. Ass. comput. Mach.*, **16**, 364–82.

Toda, M. (1962) The design of a fungus-eater: a model of human behavior in an unsophisticated environment. *Behav. Sci.*, **7**, 164–83.

Travis, L.G. (1964) Experiments with a theorem utilizing program. *AFIPS*, **25**, 339–58. SJCC. Baltimore: Spartan Books.

# 28

# Robotologic

P. J. Hayes
Metamathematics Unit
University of Edinburgh

## INTRODUCTION

A robot, in order to act intelligently, must be able to reason from facts which its sensors detect to conclusions which govern its actions. This reasoning process is so central to human intelligence that it seems immediately relevant to the problems of robot design to consider its properties, how it might be analysed and imitated.

Strawson (1959) defines *descriptive metaphysics* as the study of the

> 'massive central core in human thinking which has no history – or none recorded in histories of thought; there are categories and concepts which, in their most fundamental character, change not at all. Obviously these are not the specialities of the most refined thinking. They are the commonplaces of the least refined thinking; and are yet the indispensable core of the conceptual equipment of the most sophisticated human beings. It is with these, their inter-connexions, and the structure that they form, that a descriptive metaphysics will be primarily concerned.'

This paper may be looked upon as an attempt at the beginnings of a *synthetic* metaphysics; an attempt at an initial description and analysis of some aspects of the choice of a language and construction of an axiomatic theory suitable for the robot's central reasoning agency.

The reader will find that many problems are described, but that almost no solutions are offered. To some extent, this is inevitable at the present time; but in any case I feel that an analysis of what the problems *are*, should precede attempts to solve them.

In the first part of the paper, some general considerations on the construction of a suitable theory are outlined; in the second part, the choice of a formal language is discussed; and in the third part is an exposition of some results and possibilities for dealing with *time*.

My approach to robotology is firmly in the tradition of the advice-taker and SRI robot projects (McCarthy 1968, McCarthy and Hayes 1969, Green 1969), and I owe a large debt of gratitude to the workers at Stanford and

SRI, especially to Professor John McCarthy and Dr Cordell Green, for their advice and encouragement, and for creating an ambience in which I feel at home.

## 1. CONSTRUCTING A THEORY

In constructing a theory one consideration stands out as paramount. The robot must use its reasoning abilities to decide what to do – to decide upon courses of action. It must be possible for it to reach an appropriate decision in a fairly short time: in particular, sufficiently short that the environment has not changed so much since the last observation that the action is no longer appropriate. This need for 'almost-decidability' is the first and most severe requirement which the theory must satisfy.

It is possible to render any theory decidable in a trivial way by invoking a time cutoff on reasonings and having a default mechanism for deciding the values of any expressions still not decided. Such a mechanism will undoubtedly have to be present as a safety precaution. But if the basic theory is so unmanageable that the default mechanism is invoked frequently, then the robot will be making inappropriate responses more often than not, since of course the standardized response is likely not to be appropriate for more than a very small class of situations. It will probably be analogous to the 'freeze!' response which mammals give to threatening and uncontrollable environments. There does not seem to be any way of avoiding the conclusion that the basic theory must admit an efficient theorem-proving procedure which is close to being a decision procedure.

This conclusion is also not escaped by the idea of using special-purpose routines for certain classes of situations, since then the *decision to enter a certain routine* must be made by computations within the central theory. The necessity of avoiding long delays in decision making is still apparent.

In order to achieve this computational efficiency a compromise is necessary between two conflicting requirements. One is that the *expressive power* of the theory should be as great as possible, so that proofs are not of exorbitant length. The other is that the *search space for proofs* should be as *small* as possible.

Both of these are clearly related in a direct way to efficiency. If the proof desired is excessively long, then there will not be time to generate it even if the search algorithm is perfect. This is dramatically illustrated by McCarthy's first-order axiomatization of the mutilated checkboard problem (McCarthy 1964). If the search space is large, filled with proofs of irrelevant theorems, then even short proofs will be hard to find: there will be no efficient search algorithm.

Unfortunately these two requirements are in direct conflict. A rich, expressive theory is, by definition, one in which many facts can be phrased and in which many theorems can be proved. Thus the search space resulting will be correspondingly rich and difficult to search. If restrictive conditions

are imposed upon the form of allowable proofs, then the space is cut down but the theory is weakened and proofs are lengthened. This is common lore in theorem-proving: all the restrictive strategies (hyper-resolution (Robinson 1965), set of support (Wos, Carson and Robinson 1965) for instance) have this effect of at once reducing the size of the search space and in general increasing the length of proofs. The need for compromise has been noted in the set-of-support context.

The lack of expressive power of first-order logic has been noted on several occasions, notably by McCarthy in the reference cited above. The use of richer languages, for instance higher-order logic, has been put forward as a desirable goal to overcome the resulting difficulties. But such rich languages suffer from the other defect above, namely, the extremely large search space thus created for a theorem-proving program.

Consider for instance the question of instantiation, the logical deduction of an instance of an expression from the expression itself. In first-order logic the only expressions which need be substituted for variables during a search are the members of the Herbrand universe, which has the simple structure of a free algebra on a small number of generators. Even so, a direct search through all Herbrand instances yields a hopelessly inadequate theorem-proving algorithm. The giant step in theorem-proving research was the construction of a method whereby the process of instantiation could be controlled so that only those instances were automatically selected which could potentially be used in a proof. This is what the well-known unification algorithm achieves (Robinson 1965, Prawitz 1960). It is now clear that this is effectively *all* the control that *can* be had over instantiation, if completeness is to be preserved. One does not therefore hope for a further major breakthrough; this feeling is borne out by the lack of significant improvements in theorem-proving efficiency since the first Resolution programs, in spite of the greatly expanded research effort. What progress has been made has been due to use of more sophisticated searching and retrieval algorithms within the context of the unification and matching process.

When one comes to higher-order logic the instantiation process is vastly more difficult to control. If most of the power of the language is not to be sacrificed, then a potential instance of a higher-order variable is any lambda-expression of the appropriate type. The set of all such expressions has a far more complex structure than the first-order Herbrand universe. Moreover, no method is known for controlling the instantiation process in higher-order logic: indeed, it is known that no matching algorithm exists (Gould 1966). Without such a device the search is hopeless. Consider a particularly simple example, namely, the induction axiom

$$\forall P(P(0) \wedge (\forall x P(x) \supset P(x+1)) \supset \forall x P(x)). \qquad (1)$$

Searching for instances of (1) involves enumerating *all properties of numbers* and trying them in some order: clearly a hopeless task. It is precisely this job of selecting the appropriate instances of the induction axiom which is often

held to constitute the 'creative' element of a proof: herein lies the 'concept-formation'.

This is one illustration of the antagonism noted above between expressive power and search efficiency, and there are others. I do not feel that there is any way of avoiding the dilemma, but there have been some suggestions. The most notable ones are that theorem-provers will continue to improve in efficiency at the same dramatic rate as they did during the last decade – I have indicated above why I think this is a vain hope – and that the problem is relieved by using heuristics. Thus one is supposed to use a rich, expressive language and control the resulting gigantic search by the rigorous use of powerful pruning heuristics. But where are the heuristics to come from? This can only have two answers – either we are really reduced to some weaker theory again or else we are cheating by invoking an 'intelligent subroutine'. The argument has some validity in applying theorem-proving to highly specialized problem domains, where powerful heuristics are available, and also in interactive applications where the human is the intelligent subroutine (Guard *et al.* 1969, Allen and Luckham 1970), but for the robot, generality is all-important and powerful – problem dependent – heuristics just will not be available.

This is not to say that *no* heuristic power can be used: syntactic heuristics of the kind described by Quinlan and Hunt (1968) of course will be useful. But these are essentially *weak*, since they have no information available to them other than the syntactic form of the expressions themselves. Powerful heuristics are notoriously application-dependent.

### 1.1 Ontology

These rather general considerations can be applied to a concrete problem in theory construction which, although of fundamental importance, has not received any explicit attention in the literature on robots. This is the question of what ontology is to act as the foundation of the theory; or, to put it in less philosophical terms, the question of what entities the robot is to believe exist. Another way of phrasing it in the (present) context of formalized languages is as the problem of what entities are to be regarded as being values of bound variables (this is Quine's dictum: 'to be, is to be the value of a bound variable') and (although there are some technical complexities here) of individual constants. This rendering of the statement of the problem makes clear the relationship between ontological commitment and the sort of formal – and hence computational – complexity of theories discussed above.

The question of ontology has been much discussed by modern philosophers. I will outline two distinct approaches to it, and discuss them with a view to their use by a robot. Of course there are others; and my necessarily brief remarks will not do justice to the arguments used by the philosophers in question; and their motivations in constructing these ontological systems are quite different; nevertheless some useful points seem to emerge.

The first view may be caricatured as *set-theoretic platonism*. It is the view that *sets* exist. By Quine's dictum, anyone who advocates the inclusion of set theory in his theory must admit to the view that sets exist: and set theory is widely held to be at the basis of all of mathematics.

Now, set theory is the example *par excellence* of a rich, expressive theory. It is also one of the formally most intractable theories ever devised, as the huge literature concerned with elucidating its structure testifies. It seems quite impossible that a robot could be made to use the whole of axiomatic set theory in an economical way.

All the objections detailed above to higher-order logic apply with greater force to set theory. In particular, the instantiation problem reappears in the guise of the axiom schema of comprehension which allows us to assume the existence of a set corresponding to any* well-formed formula of the language with one free variable. Many of the axioms simply give one a licence to construct sets: thus the axioms of union, pairing and power-set all have this character. Axiomatic set theory provides a rich field *within which* mathematical creativity may be exercised; but it is far too large a field to *search*. It is, of course, not intended to be otherwise.

It is true (cf. below) that certain features of set theory are attractive and that certain weaker theories which include some sets in their ontologies (weak second-order theories for instance) may be useful: but that is another question.

An opposite extreme from platonism is *nominalism*, of which one of the most distinguished exponents is Nelson Goodman (1966). This is the view that the only things that exist are individuals, in some fairly restrictive sense intended to capture *concrete* things as opposed to *abstract* entities like sets. Basic to Goodman's conception of an individual is the notion of a part and the idea of combining parts to yield wholes. This might smack rather of set theory with its hierarchical collections of objects, but in fact is far more restrictive. Consider for instance three objects, which we will suppose indivisible for the moment: a CUP, a SAUCER and a TABLE. For Goodman there are seven individuals here: they comprise the three objects taken singly, the three pairs such as CUP+SAUCER, and the triple CUP+SAUCER+TABLE. Set theory on the other hand tells us that there are a huge infinity of individual entities present: they include for instance the sets {{}, {CUP, SAUCER}, CUP, {TABLE}} and {{{CUP}}}. Indeed, set theory has no need at all of concrete individuals to start on its transfinite hierarchy of sets; for we can imitate the integers starting from the null set: {}, {{}} etc., and the whole of mathematics follows.

Goodman's wholes – sums of parts – have none of the internal hierarchical structure which set theory insists must be present in every collection, and of

* To avoid the paradoxes, this needs to be restricted in some way. One takes one's pick from the many possibilities. There are always a sufficiently large number of sets to cause trouble however.

which its notation provides a ready analog. To insist on the presence of this irrelevant structure seems futile. Thus in this light Goodman's 'calculus of individuals' seems preferable to set theory. But it also has its faults. *Everything* is an individual, for Goodman: this is the nominalists' basic ontological commitment. Whatsoever individuals A and B may be, A+B is another. Even such unlikely combinations as CUP+THE STATE OF UTAH are allowed. (Of course the same objection applies in greater degree to set theory: but let us leave set theory for a while.) For Goodman's purpose this is entirely appropriate: but for the robot it is most surely not. Some constraints are essential. The robot needs to be able to consider a certain whole as an individual when it is more useful to do so than to regard it as a collection of interacting parts: for some combinations this will never be the case.

This idea of an individual is inspired by Simon's definition of a 'complex system' (Simon 1969). In Simon's phrase, the robot needs to be a *pragmatic holist*: when a certain whole – sum of parts – behaves in a way which is more economically described in terms of it as a unit than in terms of its parts and their interactions, then the robot needs to be able to regard it as indeed a unit, a single individual, an unstructured whole. The importance of this idea for the robot's theory is its entailing that what is considered to be an individual may vary depending upon the robot's circumstances and motivation: individuals come apart and recombine in different ways at different times. I know of no ontological system which faces this problem squarely.

The other major fault, for us, in Goodman's system, is its total lack of mathematical expressive power. Since *every* totality is unstructured, and there are *no* abstract objects – no sets, functions, numbers, relations – it is all but impossible to express facts of number, indeed to do even very elementary mathematics. This is intolerable: some minimal level of mathematical expressiveness is essential for everyday reasoning.

This points back to set theory – but not all the way. It seems anticlimactic to suggest a compromise at this point, but compromises are common in engineering. The robot needs some – I think a *very* little – of the expressive power of set theory (or higher-order logic). I have in mind something like the weak second-order theories of linear order (Lauchli 1968, Siefkes 1968), which have considerable expressive power, but are so weak as to admit decision procedures. It also needs some restriction of the pragmatic kind described above on what constitutes an individual. In general, following McCarthy (1968) and Minsky(1968), we can agree that the robot cannot reason about anything unless it can be told it: but, we can add, it must not be told too much.

There are other important and difficult ontological questions which have been widely debated in modern philosophy and which cut across the platonism–nominalism dispute; most notably perhaps the question of tense. Individuals (in the common sense of material objects) are created and destroyed within time. Consideration of past and future existents raises problems

which have been grappled with since Aristotle, with but scant success until fairly recently. I will take up this point again later.

The above may have left the reader pessimistic; but one can make some positive suggestions towards a suitable ontological system for a robot.

The chief fault of both platonism and nominalism is that they are reductionist in tendency: they strive to explain the whole world in terms of a small collection of primitive ideas. The stress is on *conceptual* economy (cf. Quine (1960) on 'the ordered pair as philosophical paradigm'): *structural* economy is left by the wayside.

Now the robot needs quite the opposite. As I have argued above, structural economy is of the greatest importance: the space through which the theorem-prover is to search must be kept to a manageable size. Conceptual economy however is not at all important: on the contrary, conceptual *richness* is, within fairly precise limits, a desirable property of the robot's ontology. Briefly, its individuals should be classified into different kinds, should in fact be *sorted.*

Not only does this simplify the job of its sensors, it helps to reduce the size of the search space still further. Placing a sort structure on the theory amounts to giving a semantically based classification of the syntax, which can be used to control the search. It makes available more information to the syntactic heuristics, for instance. This observation is again familiar to those working in theorem-proving (Guard *et al.* 1969).

This does not contradict what was said earlier about the need for economy in ontological commitment. It is a 'horizontal' rather than a 'vertical' richness which is wanted; many different kinds of individual; but not many of each kind. This is precisely the opposite type of economy from that which reductionist ontologies offer.

There are other reasons for finding a sort structure desirable. Somehow the phenomena which its sensors detect must lead the robot to believe facts about its environment; and it seems likely on the face of it that this mysterious process will inevitably give rise to conceptual classifications of the type being discussed. Thus, perhaps an *event* is distinguished from a *thing* by being intangible and of short duration; a *sound* is reported by a different sense organ than a *colour.* Both physical and phenomenological entities are automatically rendered into sorted language by reason of their different relationships to the sensory input. (This also takes care of the Dreyfusian objection that by inventing and supplying to the robot a sort structure we are somehow cheating, by building in our own insights. The sorts arise from the engineering requirements, not from a synthetic *a priori.*)

The interesting problem is now open to elucidate the sort structure which the robot is to use. Some initial candidates for useful classifications come immediately to mind: *things* (in the ordinary sense of material objects with a comparatively long life span), *people* [*things* capable of executing strategies. The distinction is important because *people* are potential antagonists, and

one may need to use game theory against them. *See* (Braithwaite 1955)], *actions* (Davidson 1967) argues for their inclusion. Consider 'he flew to the moon/quickly/in a rocket/last Tuesday/...' Qualifications may be added endlessly. The only way to make logical sense of this is to introduce an entity which has all these properties: thus the above becomes something like $\exists x(flying(x) \wedge subject(x) = him \wedge destination(x) = moon \wedge quick(x) \wedge vehicle(x) = rocket \wedge date(x) = last\ Tuesday \wedge \ldots)$. What is $x$ here but an action – 'his flying to the moon'? Also consider the statement 'he did it quickly'. Did what? – the answer must be, an action.) *events* (again argument due to Davidson (1967). Statements of causality – 'the stabbing caused Caeser's death' – are naturally represented as statements of relationships between events.) *strategies* [a generalizations of *actions*; of especial use in dealing with unknown quantities and with *people*, *see* above and McCarthy and Hayes (1969)]. Other possibilities include *time-instants* and *places*, and of course one will think of others.

The collection of these sorts will have a certain structure: some sorts are included in others, for instance. If the sorts are disjoint, then present-day theorem-provers will deal with them without modification: if they are partially ordered by inclusion, the necessary modifications are not difficult (Guard *et al.* 1969). If however more complicated sort structures are envisaged [as for instance have been devised the better to model natural languages (Bar-Hillel 1950)], then the problem is open, and appears quite difficult.

One other sort which has been used is that of a *possible world* or *situation* (McCarthy 1968, Rescher and Garland 1968, Lewis 1968). The resulting class of theories is very large and contains theories mimicking in their structure almost all the known modal logics. These latter seem to be extremely useful in constructing real-world theories containing statements about tense, knowledge, etc. (see below for instance), and the situation calculus is therefore similarly useful. It might be thought that the ontological commitment involved here was far greater even than in set theory – a 'possible world' is apparently a far more intractable object than a set, and the methods of constructing new worlds from old correspondingly difficult to describe. But the situation calculus is in fact quite tractable (Green 1969), on present form. The reason is that the ontological commitment is not what it seems, since in present-day applications of the situation calculus very little structure is imposed on situations. There are perhaps a few partial orderings defined by means of functions representing actions. Thus the *actual* commitment is only to certain entities having this minimal amount of structure. If one were to construct, within the situation calculus, an axiomatic theory in which many complex interactions between situations were described, then the commitment would be to more complex entities and the resulting theory would indeed be intractable in precisely the sense described above. Attempts to construct more complex theories within the situation calculus have run into just this problem.

One can illustrate this by the process of translating modal logics into the situation calculus. The basic idea is simple and well known. Propositions map into fluents with a single free situation variable: $p \rightarrow p(s)$. The translation is direct ($p \wedge q \rightarrow p(s) \wedge q(s)$ etc.) until one reaches the modal operators. The statement of necessity, $\Box p$, maps onto $\forall t(R(s, t) \supset p(t))$ where $R$ is a binary relation ('alternativeness') between situations. The axioms of the modal logics translate into conditions on $R$.

So far all is fairly simple. But suppose we try to translate a modal *predicate* calculus into the situation calculus. Then the quantifiers translate as follows:

$$\forall x P(x) \rightarrow \forall x(\text{IN}(x)(s) \supset P'(x)(s))$$

where $P'$ is the translation of $P$ and IN is a new monadic fluent (intuitive meaning: $x$ exists in situation $s$) for which many axioms must be supplied.

Of course analogous problems arise in considering the modal logics directly, but hidden in the very complicated semantics. The translation into first-order logic via the situation calculus has the merit of bringing them out into the open, so to speak; but it does not eliminate them. There has been lively debate among philosophical logicians as to which approach to the (very real) problems of referential opacity is most rewarding: *see*, for instance, Quine (1953) and more recently Massey (1969) for some broadsides against modal logic.

We will take up this matter again later.

## 2. THE CHOICE OF LANGUAGE

A theory must be phrased in some formal language for which the robot has available an efficient proof procedure. It is natural to ask: which such language is the most appropriate?

Up to now the most successful robot has used the Resolution formulation of classical first-order logic (Green 1969). One may therefore question the need to look beyond first-order logic: and one would be supported in this scepticism by some eminent philosophers, notably Quine.

However it does seem that there are some good reasons for so doing even though we have no mechanizable inference systems available yet for other languages. In this section I will try to outline three of the most pressing of these reasons. They are concerned with problems respectively of *time*, *perception*, and *ambiguity*.

### 2.1 Time and tense

The robot, like ourselves, lives in time: it has a future and a past. Like ourselves, its memory will be imperfect and its predictive power even less perfect. Like ourselves, it must have a coherent way of reasoning about the contingent future. This is extremely difficult in classical truth-functional two-valued logic.

The difficulty is not simply the problem of constructing a logic of tenses or

time-intervals. The construction of such logics has been a major task in philosophical logic and a huge amount of work has been done (*see*, for instance, Prior 1968). Tenses turn out to be modal operators because they are not truthfunctional (*p*s being true in the future in no way depends upon its being true in the past). It is therefore possible to translate them into the situation calculus roughly as described above, thus rendering everything back once more into classical first-order logic. This has, as noted above, been urged by several authors. However it seems to me that it would be a mistake to conclude that modal logics were useless. The modal logic formulation seems more likely to admit an efficient proof procedure. The reason is that when the modal formulae are translated into the situation calculus, formulae may be inferred from them which are not the translation of any statement of the modal logic. The mapping of search spaces induced by the translation is an imbedding, not a homomorphism.

The difference between the two approaches can also be understood as the difference between using *tenses*, or using quantification over *time-instants*, to handle time. The former are definable in terms of the latter, as above: and in fact the reverse is also true, provided that a clock is available (Prior 1968).

My feeling is that the economy achieved by using tensed language more than compensates for its lack of expressive power, especially in view of this fact that the power can be regained by an artificiality. For what it is worth, one can point to the fact that people are built this way: we seem to have a direct primitive intuition of pastness and presentness (tense), and can only reconstruct time instants by using clocks. Natural language also uses tenses rather than time-instants as its basic way of dealing with time.

There is a deeper difficulty in handling time, however. Any assertion in the future tense is a prediction, and may turn out, in view of the robot's limited predictive ability, to be mistaken. Moreover some statements in the future tense are open – the robot has no opinion because it is unable to find sufficient evidence either for or against them. (This phenomenon occurs of course without introducing tenses, since the robot has limited deductive power in general. But, as argued above, it should in general happen infrequently: otherwise the robot is simply inadequate to its assigned tasks. With regard to *time*, however, the robot cannot help but have an inadequate theory in this sense. The interactions of everyday environments are too complex for any fast predicting mechanism to be adequate.) All these considerations point to the need for a many-valued logic. A minimum of three values – true, false, indeterminate – seems a necessity. There is no efficient way of translating many-valued logics into two-valued logic.

This is argued (with reference to people, not robots) in greater depth by Lukasiewicz (1967). Indeed it was his considerations of future contingency which led to the first construction of a many-valued logic.

The logic of three truthvalues has been put into a mechanizable form

[which could however bear some improvement (Hayes 1969)], so we are a little way forward on this front.

However, the major problem here may be an extra-logical one, namely, describing precisely how the truth values change as the robot moves through time. This is closely related (thinking takes time) to what has come to be called the frame problem (McCarthy and Hayes 1969, Minsky 1968) and to the philosophical problems of counterfactual conditionals (*see* McCarthy and Hayes 1969). It remains a central difficulty, but a subproblem which has been partially solved is that of determining which statements are valid in such an environment, i.e., which statements need never be re-examined to see if they have changed their truthvalue.

It might be thought that every valid formula of first-order logic was such a law: but if one wants the set of all such statements to be closed under deduction (so that they form an 'inner logic'), and if one insists – as is extremely natural – that the ordinary connectives are truth-functional, then this is no longer the case.

I will go into this in more detail below: for the present it seems clear that the problems of dealing with time take one beyond first-order logic, and to establish this was my intention.

### 2.2 Perception

Facts about the robot's environment must be obtained through its sensors. This perceptual process is probably the least understood of all, at present. However one thing stands out in all work which has been done on robot construction, that being the need for conceptual feedback during the perception process.

This need has been noted by workers at Stanford (Feldman *et al.* 1969) and Edinburgh (Murphy 1969). It is also well known in experimental psychology (Averbach and Coriell 1961, Denes 1967, Yarbus 1967) that human perception is not a purely passive process, but is guided by attention mechanisms which select, on the basis of concepts formed from a partial analysis of the input, which parts to attend next.

I will not attempt to describe this process in any detail. Its effect on the language is however profound. Sensors do not deliver perfect information, but rather guesses together with an estimate of their reliability. Implementing the feedback process mentioned above requires that these guesses, and the estimations as to their truth, must be capable of being represented in the language. Moreover the robot must be able to perform inferences on the basis of these facts, inferences which direct the attention of the sensors to the vital parts of the input. These deductions must not be such as to propagate errors in an uncontrolled way: the proof procedure must deal properly with the guesses and the reliability estimates. This implies using a many-valued logic.* Thus we are again forced away from classical logic.

* The SRI group have toyed with this idea, apparently for a similar reason (*see* Green 1969).

It may be objected that the feedback process should be thought of in isolation from the deductive theory, as a sort of complex preprocessing. And in fact this is what has been suggested to date. But there are two reasons why this seems wrong. One is methodological: the division is artificial, and I venture to predict that the generality and power of theorem-proving methods will greatly advance cognitive research when they are applied to it, as has happened to other fields in the past. The other is more important. In a fast-moving (compared to its operating speed) environment the robot will need to *act* on the basis of perceptual half-truths, will need, for instance, to have protective reflexes when it detects severe threats, even though the threat may be only partly verified by the sensors. This requires that the whole deductive apparatus be intimately associated with the sensory feedback at every stage.

There are other problems which arise when one deals with percepts, however, by whatever process they are produced.

Consider the assertion $P(a)$: it says that the individual $a$ enjoys the property $P$. What does it mean to assign a truthvalue of (say) 0·5 to this statement? It could mean that the individual $a$ has the property $P$ to this degree – that $a$ is '*half-P*'; or it could mean that $P$ is *definitely* true of an entity which is '*half-a*'; which has been but partially identified and can only be called $a$ with limited confidence. The former interpretation is the usual one in many-valued logics, but the latter – which is far more damaging to the formal semantics and the conceptual basis than the other – seems to be what is wanted here.

Similar difficulties arise in interpreting modal assertions (is $\Diamond P(a)$ an assignment of *possibly-P* to $a$, or an assignment of $P$ to a '*possible-a*', whatever that means?) and modal logicians have been active in investigating them. It seems that the only way to construct a coherent semantics for modal predicate logic is to delineate precisely what the criteria for identifying individuals are. It is impossible to summarize the whole of this work here, but it is clearly directly relevant to the perceptual difficulties being discussed.

The distinction between individual constants and variables, relatively minor in classical logic, is crucial in modal logic. Variables, being simply cross-referencing devices for quantification, are not susceptible to these interpretive difficulties. But constants are encumbered by their referential apparatus and their freedom of use is thereby restricted. Thus such classically valid inferences as $\forall x P(x) \rightarrow P(a)$ typically fail in quantified modal logics.

These sort of complications have led some authors, notably Quine, to attack quantified modal logic as requiring unsavoury philosophical assumptions; but for the present purpose this is precisely the sort of behaviour one would want, and this research into the philosophical foundations of modal logic seems certain to produce valuable results for the robot's cognitive difficulties.

### 2.3 Ambiguity

In natural language usage one continually uses nouns which mean slightly different things depending on the context of their use. To take a singularly trivial example, 'pass the salt' usually is a request to pass a container of salt rather than the salt itself: the context is sufficient to force the reinterpretation of the word 'salt'.

That this sort of ambiguity does not cause problems is due to the fact that the recipient of such a message is himself capable of disambiguating it: the receiver is an intelligent creature.

Disregarding the question of communication for now, it seems likely that the robot's planning facilities will comprise a hierarchy of some sort. There will be higher-level plans defining strategies which are interpreted by lower-level planning devices whose responsibility it is to deal with the tactics of the situation. Now it seems reasonable that the sort of ambiguity being discussed should be present in the plans formed by (and hence, in the cogitations of) the higher-level control, since these plans are to be interpreted by an 'intelligent' device of some sort.

I have not yet, however, said why such ambiguity would be desirable. It is, as usual, a matter of efficiency. Suppose one undertook never to commit the unFregean sin of using one name to refer to different objects in the sort of way indicated. Then his theory must contain many distinct names for closely related entities, and in order that their close relationship be apparent, it must also contain axioms concerning these interrelationships. Thus his theory will be cluttered up with what might be called the micro-physics of the situation. It is *essential* that this kind of close analysis be left to the tactical – lower-level – planning stage.

Now, the presence of this ambiguity strikes at the very root of classical logic. It is closely related to the modal difficulties described above: in both cases we have a crucial weakening of the *denotation* relationship holding between the objects of reference (individuals) and their names (individual constant symbols). What seems to be necessary is that some flexibility is available in ontological commitment: in deciding what comprises an individual. Again we are brought back to the nominalistic difficulties described earlier.

### 2.4 Modal versus classical

The two kinds of nonclassical logic which arise naturally are seen to be modal logics and many-valued logics. As mentioned briefly above, it is possible to translate both of these back into classical first-order logic in various ways. It is therefore not clear whether one has to extend classical logic in constructing robot-oriented theories.

In the case of many-valued logics the mapping is clearly unattractive. One introduces constants $t_1, t_2 \ldots$ which are supposed to denote the various truthvalues; and then instead of asserting a proposition $p$ with truth-value

$t_n$, one asserts the classical (two-valued) truth of '$p=t_n$'. This essentially amounts to constructing a first-order axiomatic metatheory of the many-valued logic. Although an interesting theoretical exercise, the resulting theory is less intuitive, far clumsier, and is likely to be hopelessly less efficient than the original logic.

In the case of modal logic the matter is less clear-cut, however. I have touched upon this famous dispute several times already. The full story seems to be as follows. The situation-calculus modelling of modal logic is sometimes intuitively clearer than the modal logic itself (tense logic), sometimes less so (epistemic logic). It is likely that efficient proof procedures for languages involving modal notions will have to treat modal operators – or the first-order constructions which mimic them – in a special way. This seems, in view of the imbedding of the search spaces mentioned above, easier to do in the context of the modal logic itself. The situation-calculus approach, on the other hand, has the methodological advantage of providing a unified framework within which to relate different modal notions, but it does not eliminate the real difficulties of elucidating the referential structures implicit in the semantics. Indeed the construction of a suitable first-order theory and the construction of a formal semantics for the modal calculus seem isomorphic problems (see Montague (1968) for an example of the latter).

It seems clear that each approach has its advantages and snags. The only real sin is dogmatism.

There are other reasons for considering nonclassical logics, especially the problems of reasoning about *knowledge*, and connexions between the robot's reasonings and his actions. I have not discussed the former since it was aired at some length in an earlier paper (McCarthy and Hayes 1969) and I have nothing new on the subject; and I do not understand the latter, but expect that problems analogous to those of perception will arise.

Of the three reasons outlined, the first – the difficulty with time and tense – seems the most fundamental. To some extent it subsumes the frame problem and the perceptual difficulties, since both of these arise in the context of a discovery process which takes place in time. It is also the area where most progress has been made, due probably to its philosophical interest. In the next section I will outline some recent work in philosophical logic which is relevant to it.

## ASSERTIONS IN TIME

The results described in this section are due to Kripke (1963, 1965) and Grzegorczyk (1964, 1968). Also much of the underlying philosophical discussion is taken from these papers. There will be complete confusion between use and mention.

The robot's life can be thought of as a process of discovery. It finds out new facts as time passes. It is capable of perceiving, and choosing from,

alternative courses of action. Moreover when it has acted upon such a choice it cannot retrace its steps. Disregarding for the moment the purpose of such actions, we will isolate those which result in the acquisition of new data about the environment.

The natural formal representation of this process is as a directed graph, or *digraph*. At the nodes of the graph will be collections of statements representing the state of the robot's knowledge at successive time-instants. The arcs of the graph correspond to experiments which yield new information about the environment. The directedness of the graph reflects the irreversibility of the robot's decisions.

Let $\leqslant$ denote the ordering relation on such a digraph, and let $N$ be a finite or infinite set of nodes. At each node $n \in N$, there are sets $I_n$ of *individuals* and $A_n$ of ground atomic statements (*atoms*). $I_n$ is a set of individuals which the robot has met at that time, and $A_n$ is a set of observational facts which the robot has verified.

In order to make this quite precise we must specify the language which the robot is to use to represent this knowledge. This will be, for the moment, the classical first-order predicate calculus augmented by the modal operator $G$. The intended meaning of $G$ is '*it will always henceforth be true that . . .*'. We may define the dual modal operator $F$ as $\neg G \neg$, and then $F$ means '*it will be true that . . .*', i.e., $F$ is the future tense operator. These modal operators comprise the robot's language for speaking of time.

The digraphs, with their associated structure, act as interpretations of this language in the way described in detail below and for this interpretation Kripke gives a completeness result:

*Theorem I.*

*The set of formulas true at all nodes of all digraphs is the set of theorems of the modal logic CS4.* (See appendix.)

The interpretation mapping is defined as follows:

The truthvalue of an atom $A$ at a node $n$ is $T$ if $A \in A_n$; otherwise the truthvalue of $A$ at $n$ is $F$. Notice that we assign $F$ to $A$ even though neither of $A$, $\neg A$ is true at $n$. This asymmetry between truth and falsity will be important in what follows. We can now define the truthvalues of more complicated formulae by induction on their structure. Let $[P]_n$ be the truthvalue of $P$ at $n$. Then:

C1. $[\neg R]_n = \neg [R]_n$

C2. $[P \vee Q]_n = [P]_n \vee [Q]_n$

C3. $[\exists x P]_n = T$ if $[P(a/x)]_n = T$ for some $a \in I_n$, otherwise $= F$.

C4. $[GA]_n = T$ if $[A]_m = T$ for every $m \geqslant n$, otherwise $= F$.

The first three conditions mean that the interpretation of formulae not containing $G$ is the usual classical one, based on the evidence available. Condition C4 captures the intended meaning of the modal operator $G$: $GA$ is true at $n$ if and only if $A$ is true throughout $n$'s future (which includes $n$ itself).

It seems to me that this result is of central importance for robot research. The digraph semantics generalizes the common notion of 'look-ahead tree' of classical economic theory and much AI research, but the above result does not in any way depend on the digraph's structure being accessible to the robot. This assumption of perfect information is the main weakness of the use of the 'look-ahead' paradigm in these fields (Simon 1967).

This result can be extended in several ways. The construction of the digraphs has so far been perfectly free: no conditions have been placed upon the sets $I_n$ and $A_n$. Thus individuals may be created and destroyed, or may change their properties in arbitrary ways, as time progresses. This seems appropriate for everyday reasoning in a changing world. However, one may ask for a similar construction which would be appropriate for the discovery of laws, that is facts, about the world which are not expected to change.

To achieve this we impose the following restrictions on the digraph construction:

R1: if $n \leqslant m$ then $I_n \subseteq I_m$

R2: if $n \leqslant m$ then $A_n \subseteq A_m$.

These say that the universe is growing, that is, that individuals once discovered never disappear; and that atomic facts never change. These seem to be the appropriate conditions for law discovery, but one can make out a case for their being true in general. This requires the reinterpretation of the atoms and it will be discussed more fully later.

A digraph obeying R1 and R2 will be called a *law-graph*. Using the same definition of the truthvalue of an atomic formula as above, we need a suitable collection of induction clauses to define the truthvalues of more complicated formulae. Bearing in mind the asymmetry between truth and falsity mentioned earlier, it is easily seen that the clause for negation must be strengthened in order to be sure that negative statements shall also be law-like. Also, since we want *modus ponens* and instantiation to be correct rules of inference in the resulting logic, we must pay particular attention to the clauses for implication and the universal quantifier. In the latter case, for instance, the assertion of $\forall x P(x)$ must only be allowed when it is guaranteed that it cannot be falsified in the future – even though new individuals are discovered – for otherwise it cannot be regarded as a law.

The resulting collection of induction clauses is then the following:

L1. $[\neg P]_n = T$ if $[P]_m = F$ for every $m \geqslant n$, otherwise $= F$.

L2. $[P \vee Q]_n = [P]_n \vee [Q]_n$.

L3. $[P \wedge Q]_n = [P]_n \wedge [Q]_n$.

L4. $[P \supset Q]_n = T$ if $[P]_m \supset [Q]_m$ for every $m \geqslant n$, otherwise $= F$.

L5. $[\exists x P]_n = T$ if $[P(a/x)]_n = T$ for some $a \in In$.

L6. $[\forall x P]_n = T$ if $[P(a/x)]_m = T$ for every $m \geqslant n$ and every $a \in A_m$, otherwise $= F$.

L7. $[GA]_n = T$ if $[A]_m = T$ for every $m \geqslant n$, otherwise $F$.

(Notice that L2=C2, L5=C3 and L7=C4.)

With this semantics Kripke (1965) proves the following result (*see* also Grzegoroczyk 1964, 1968):

*Theorem* 2

*The set of statements (strictly: the set of statements not containing G) true at all nodes of all law-graphs is the set of theorems of the intuitionistic predicate calculus IPC.*

As remarked earlier, it is perhaps surprising that one gets a weaker logic than the classical predicate calculus. Consideration of a simple counter example to the law of excluded middle however shows why this must be the case. The simple law tree in question is shown in figure 1. It is easily seen that $P(a)$ is $F$ at $n$, since it certainly is not $T$. But, $\neg P(a)$ is $F$ also at $n$, since it is $F$ at $m$. Therefore $P(a) \vee \neg P(a)$ must be $F$ at $n$.

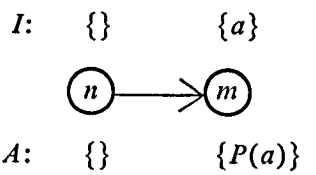



Figure 1

The reason for this behaviour is clearly the insistence upon two-valued truthfunctionality. It may be that one could avoid classifying the law of the excluded middle as a falsehood by introducing a third truthvalue; and indeed this would fall under the general plan outlined earlier. But the law could still not be classed as a truth without sacrificing truthfunctionality. Attempting to accommodate this within a modal logic leads to inconsistency, as follows from results of Montague (1963). I feel that this kind of behaviour should simply be accepted.

Of course, the robot will never find a *counterexample* to the excluded middle, or indeed to any classically valid statement: but that is another question (see Theorem 3 below).

One may quarrel with the condition R1 on law-trees, even in the context of law discovery, on the grounds that *individuals* must be allowed to decay, albeit that *facts* may not change. For where are eternal individuals found? They certainly cannot be physical objects in everyday experience.

There are two answers to this objection. One is that it is appropriate for the robot to regard as eternal any object of sufficient stability that it has a *long* life expectancy: houses, geological features of the landscape, etc. The resulting oversimplification will be true most of the time; and on the few occasions it is not, it is acceptable for the robot to spend time on a lengthy revision of its world-model. It may be indeed that this will work – people seem to have similar problems.

The second answer to the objection is more far-reaching. It is to regard

existence as including past existence. This goes part way towards the 'de-tenser' attitude (Massey 1969) that the only meaningful use of 'exists' is *timeless*: thus one has to speak of existence *at a certain time* to capture the meaning of 'exists' in tensed language. The suggested compromise amounts to accepting this dogma for the past, but retaining the tense-logical view for the future. Thus '$\exists x P(x)$' now reads 'there is, *or has been*, an individual $x$ such that $P(x)$'. To keep the expressive power of the old theory one must introduce a one-place predicate *dead*$(x)$: *dead*$(a)$ is to be true at $n$ when $a \in I_n$ but $a$ does not denote any individual which actually exists – in the tensed sense – at $n$. Then the assertion of *present* existence becomes $\exists x$ $(\neg$*dead*$(x) \wedge \ldots)$. In this way one satisfies condition R1, trivially.

In order to render all time-logic into the law-tree semantics one can go further and assume that the atomic propositions are *dated*: that they include a reference to the node $n$ at which they are asserted. This is even more of a step towards the 'de-tenser' view of time, but again only with reference to the past; the future is still to be handled using $G$. This makes condition R2 trivially satisfied also, but at considerable cost in brevity of the formalism and memory requirements.

This asymmetry between past and future, although probably repugnant to a philosopher, seems very appropriate to the robot. Its memory will be far more reliable than its predicting mechanism; moreover it can choose which facts to forget, but it cannot completely choose which to predict.

Theorems 1 and 2 are closely related. There is in fact a deduction-preserving mapping (*see* Appendix) from IPC into CS4 under which Theorem 2 becomes a corollary of Theorem 1.

Grzegorczyk has extended this result in another direction. Still considering law-trees, one can ask for the logic of hypotheses. Let us say that an assertion is *supposable* at a node $n$ if its negation does not follow from the set of ground facts $A_n$, together with a fixed consistent set of statements $L$ (a set of underlying laws which are taken as given). One can ask, what statements are always *supposable*?

Grzegorczyk (1968) proves

*Theorem* 3

*If L contains the theorems of IPC, then the set of statements supposable at all nodes of all law-trees contains all theorems of the classical predicate calculus.* And

*Theorem* 4

*If L is empty, then the set of statements supposable at all nodes of all law-trees contains all theorems of the system of strict entailment of Anderson and Belnap* (1962).

(Theorem 4 is weak, since the set certainly contains other assertions also.)

Theorem 3 makes precise the feeling that classical logic should somehow be 'correct': there will never be a counterexample to any thesis of it. Note

however that in general the set of statements supposable at a given node is *inconsistent*: at node $n$ in figure 1, for instance, it contains both $P(a)$ and $\neg P(a)$.

It seems to me that a clear case exists for regarding CS4, or its situation-calculus counterpart, as a good candidate for the basic logic for use by a robot. This is not a restriction upon the use of classical reasoning: CS4 (unlike IPC) contains classical predicate logic as a subsystem. But it also contains a mechanism for handling future contingency which, by the above results, seems exactly appropriate for the robot's situation.

## CODA

Throughout this paper I have tried to use results in, and arguments from, philosophical logic and analytical philosophy. It seems to me that these fields have more to offer artificial intelligence than is generally realized. True, the philosopher's motivation in constructing, for instance, a phenomenalistic theory within first-order logic, may be different from that of the AI worker: nevertheless he will have to consider similar problems (consider the quotation from Strawson in the introduction for instance), and to the extent that he does, so will his results be relevant.

Both the analytical philosopher and the designer of intelligent software are doing what might be called 'mental engineering': constructing precise, formal models of some aspects of intelligent thought or behaviour. There is a wide gap between them, but it is narrowing. On the philosopher's side, through the use of the analytical tools of formal logic; on the AI side, through the development and use of efficient theorem-proving programs, and the linguistic approach to problem-solving in general (Minsky 1968). Even methodologically there are similarities between the two fields: compare for instance the use of 'toy universes' by Strawson (1959) and Goodman (1966) to that by Toda (1965) and Doran (1969).

I hope that we will see more co-operation between AI and philosophy in the future. The traffic would not be one-way: for instance, consideration of the perceptual feedback mentioned in section 2.2 above may perhaps throw new light on the longstanding phenomenalist/physicalist controversy.

**Acknowledgements**

This work was supported by the Science Research Council.

I have already remarked on my indebtedness to John McCarthy and Cordell Green. Some of the other people who have helped me in various ways are Bruce Anderson, Rod Burstall, Jim Doran, Bob Kowalski, Bernard Meltzer, Donald Michie and Alan Robinson.

In particular, many of the ideas in section 1 arose from conversations with Bob Kowalski.

## REFERENCES

Allen, J. & Luckham, D. (1970) An interactive theorem-proving program. *Machine Intelligence 5* (eds Meltzer, B. & Michie, D.).

Anderson, A.R. & Belnap Jr., N.D. (1962) The pure calculus of entailment. *J. Symbolic Logic*, **27**, 19–52.

Averbach, E. & Coriell, A.S. (1961) Short term memory vision. *Bell System tech. J.*, **40**.

Bar-Hillel, Y. (1950) On syntactic Categories. *J. Symbolic Logic*, **15**, 1.

Braithwaite, R.B. (1955) *The Theory of Games as a Tool for the Moral Philosopher.* Cambridge University Press.

Davidson, D. (1967) Casual relations. *J. of Philosophy*, **64**, 21.

Davidson, D. (1967) The logical form of action sentences. *The Logic of Decision and Action* (ed. Rescher, N.). University of Pittsburgh Press.

Denes, P.B. (1967) On the motor theory of speech perception. *Models for the perception of speech and visual form* (ed. Wahten-Dunn, W.). Cambridge, Mass.: MIT Press.

Doran, J. (1969) Planning and generalization in an automaton/environment system. *Machine Intelligence 4*, pp. 433–54 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Feldman, J.A., *et al.* (1969) The Stanford Hand–Eye project. *Proc. First Int. Conf. on Artificial Intelligence.* Washington.

Goodman, N. (1966) *The Structure of Appearance.* New York: Bobs-Merrill Co. Inc.

Gould, W.E. (1966) A matching procedure for *w*-order logic. *Scientific Report No. 4.* Princeton: Applied Logic Corporation.

Green, C. (1969) Application of theorem-proving to problem-solving. *Proc. First Int. Conf. on Artificial Intelligence.* Washington.

Grzegorczyk, A. (1964) A philosophically plausible formal interpretation of intuitionistic logic. *Indagationes Math.*, **26**, 596–601.

Grzegorczyk, A. (1968) Assertions depending upon time and corresponding logical calculi. *Compositio Mathematica*, **20**, 83–7.

Guard, J.R. *et al.* (1969) Semiautomated Mathematics. *J. Ass. comput. Mach.*, **16**, 1.

Hayes, P. (1969) A machine-oriented formulation of the extended functional calculus. *A.I. Memo*, A.I. Project. Stanford University.

Kleene, S.C. (1952) *Introduction to Metamathematics.* Princeton, New Jersey: Van Nostrand.

Kripke, S. (1963) Semantical analysis of modal logic *I. Zeitshr. f. math. Logik und Grundlagen d. Math.*, **9**, 67–96.

Kripke, S. (1965) Semantical analysis of intuitionistic logic I. *Formal Systems and Recursive Functions* (eds Crossley, N. & Dummett, M.). Amsterdam: North Holland.

Lauchli, H. (1968) A decision procedure for the weak second-order theory of linear order. *Contributions to mathematical logic* (eds Scmidt, H.A., Schutte, K. & Thiele, H-J.). Amsterdam: North Holland.

Lewis, D.K. (1968) Counterpart theory and quantified modal logic. *J. of Philosophy*, **45**, 5.

Lukasiewicz, J. (1967) On determinism. *Polish Logic* 1920–1939 (ed. McCall, S.). Oxford: Clarendon Press.

Massey, G.J. (1969) Tense logic! Why bother? *Nous*, **3**, 17–32.

McCarthy, J. (1964) A tough nut for proof procedures. *A.I. Memo*, A.I. Project, Stanford University.

McCarthy, J. (1968) Programs with common sense. *Semantic Information Processing* (ed. Minsky, M.L.). Cambridge, Mass.: MIT Press.

McCarthy, J. & Hayes, P.J. (1969) Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, pp. 463–502 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Minsky, M.L. (1968) Descriptive languages and problem solving. *Semantic Information Processing* (ed. Minsky, M.L.). Cambridge, Mass: MIT Press.

Montague, R. (1963) Syntactical considerations on modal logic. *Acta Phil. Fennica*, **16**.

Montague, R. (1968) Pragmatics. *Contemporary Philosophy* (ed. Klibansky, R.). Firenze: La Nuova Italia Editrice.
Murphy, A.S. (1969) An application of heuristic search procedures to picture interpretation. *Memorandum MIP–R–61*, Dept. of Machine Intelligence, Edinburgh University.
Prawitz, D. (1960) An improved proof procedure. *Theoria*, **26**, 102–39.
Prawitz, D. & Malmnäs, P.-E. (1968) A survey of some connections between classical, intuitionistic and minimal logic. *Contributions to Mathematical Logic* (eds Schmidt, H.A., Schutte, K. & Thiele, H-J.). Amsterdam: North Holland.
Prior, A.N. (1968) *Past, Present and Future*. Oxford: Clarendon Press.
Quine, W.V.O. (1953) Mr. Strawson on logical theory. *Mind*, **42**, 435–51.
Quine, W.V.O. (1960) *Word and Object*. Cambridge, Mass.: MIT Press.
Quinlan, J.R. & Hunt, E.B. (1968) A formal deductive problem-solving system. *J. Assoc. comput. Mach.*, **15**, 4.
Rescher, N. & Garland, J. (1968) Topological Logic. *J. symbolic Logic*, **33**, 4.
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.
Siefkes, D. (1968) Recursion theory and the theorem of Ramsay in one-place second-order successor arithmetic. *Contributions to mathematical logic* (eds Schmidt, H.A., Schutte, K. & Thiele, H-J.). Amsterdam: North Holland.
Simon, H.A. (1967) The logic of heuristic decision making. *The Logic of Decision and Action* (ed. Rescher, N.). University of Pittsburgh Press.
Simon, H.A. (1969) *The Sciences of the Artificial*. Cambridge, Mass.: MIT Press.
Strawson, P.F. (1959) *Individuals*. London: Methuen & Co.
Toda, M. (1963) Utilities, induced utilities, and small worlds. *Behav. Science*, **10**, 238–54.
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.
Yarbus, A.L. (1967) *Eye movements and Vision*. New York: Plenum Press.

## APPENDIX

The modal predicate calculus CS4 is obtained by adding to the axioms of the classical first-order predicate calculus the following:

S4 A1 $Gp \supset p$
S4 A2 $G(p \supset q) \supset . Gp \supset Gq$
S4 A3 $Gp \supset GGp$

and the additional rule of inference:

S4 R1 If $\vdash p$, then $\vdash Gp$,

where $p$, $q$ are arbitrary propositions.

The intuitionistic predicate calculus IPC uses the vocabulary of classical predicate calculus, but the quantifiers are not interdefinable and the implication connective is not definable in terms of disjunction and negation.

The axioms and rules of inference are listed in Kleene's monograph (1952) at the top of page 82. IPC is obtained from postulates 1a, 1b, 2, 3, 4a, 4b, 5a, 5b, 6, 7, 9, 10, 11 and 12, together with $8^{I}$: $\neg A \supset (A \supset B)$ which replaces $8°$. Kleene also gives a Gentzen-type formulation of IPC on page 481.

The mapping from IPC into CS4 is defined as follows (Prawitz and Malmnäs 1968). A formula $P$ maps into $P^\circ$ where:

$P^\circ = GP$ if $P$ is an atom
$(P \wedge Q)^\circ = P^\circ \wedge Q^\circ$
$(P \vee Q)^\circ = P^\circ \vee Q^\circ$
$(\neg P)^\circ = G\neg P^\circ$
$(P \supset Q)^\circ = G(P^\circ \supset Q^\circ)$
$(\exists x P)^\circ = \exists x P^\circ$
$(\forall x P)^\circ = G\forall x P^\circ$.

The reader will see that the introductions of $G$ for universal quantifiers, negation and implication correspond precisely to clauses L6, L1 and L4 in the law-graph semantics. The introduction of $G$ for atoms corresponds to the conditions R1 and R2 on law-graphs, which effectively eliminate the distinction between $A$ and $GA$ when $A$ is an atom.

# 29

# Design of Low-Cost Equipment for Cognitive Robot Research

H. G. Barrow and S. H. Salter
Department of Machine Intelligence and Perception
University of Edinburgh

## MARK I DEVICE

A minimal robot, known as Freddy, has been constructed with the aim of connecting a usable device on-line to the Department's ICL 4130, under the Multi-POP time-sharing system, and discovering the snags. (See figure 1).

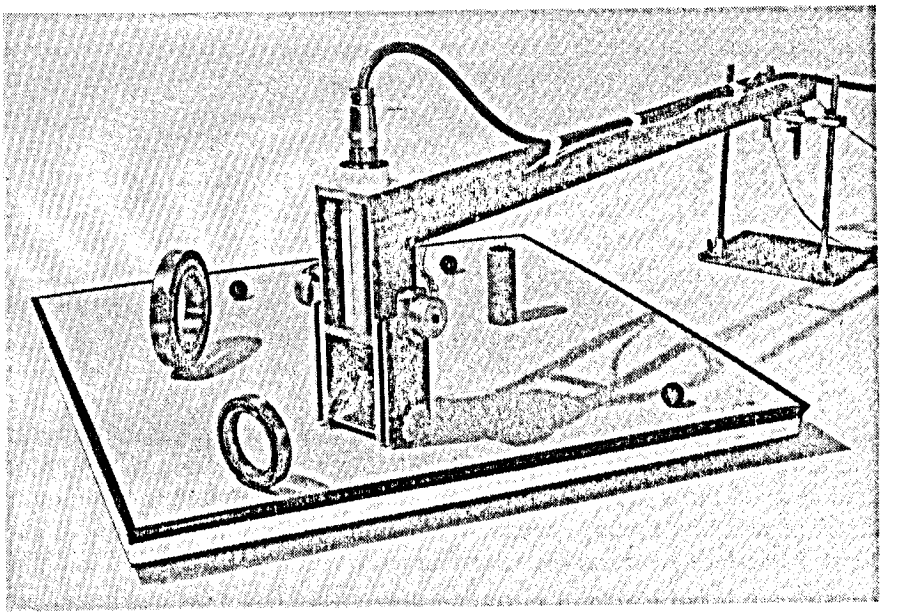


Figure 1. Freddy Mark 1 and his world

Various technical problems arise when such a device runs free. It is much easier to anchor it and allow it to push its world about. Our present world is a three-foot diameter sandwich of hardboard and polystyrene which is light and rigid. It rests on three steel balls and is moved by wheels driven by small

stepping motors, mounted on the robot. Provided that the weights of robot and slab are chosen correctly, a wide range of movements is possible. When stimulated, the motors drive for about half a second, causing the device to move about 5 mm or turn through about 4°.

A pair of bumpers, one in front, one behind, operate two micro-switches to signal contact with obstacles. A television camera mounted vertically sees the world through a 45 degree mirror. A wide angle lens provides extremely large depth of field – about 1½ in. to 3 ft at reasonably high light levels.

Television techniques are not particularly suitable for application to computer eyes, but as we already owned a closed circuit system, the cost of the project so far has been very low.

We have built the circuitry to sample and hold the instantaneous amplitude of the video signal at any point in the picture. The acquisition time is about 50 nanoseconds, which gives a horizontal resolution rather better than the vertical line resolution. Voltage levels from simple D–A converters set the *X* and *Y* coordinates and a simple A–D converter codes the sampled level for transmission to the computer. Bright-up pulses are added to the video signal to produce a cross superimposed on the monitor picture, to show an observer the position in the picture under study.

From the nature of television it is impossible to take fresh measurements 'upstream' without waiting for a new frame. If only a single measurement is made on each frame the data rate is prohibitively slow. It may be possible to achieve random scanning by using the RCA Alphechon tube, in which one gun writes a picture on a charge storage screen, which may be read non-destructively by a second gun.

The camera system which we have now is past its prime, and has a particularly poor signal-to-noise ratio, which can be seen as 'snow' on the monitor. A good system should be able to make measurements to about 1 per cent, whereas in this one the noise is about 5 per cent of maximum brightness.

Control is effected through an 8-bit, loadable command register, and an 8-bit readable conditions register.

#### General-purpose interface

Communication between Freddy and the ICL4130 is via a general-purpose interface. The aim of the design of the interface was to provide a high speed data channel through a 'bomb-proof' socket so that a number of different devices (including Freddy) could be connected or disconnected at will. The Mark I interface possesses the following features:

*High speed.* It is capable of operation at over 500,000 8-bit characters per second, but is limited by the ICL4130 transfer rate to about 300,000 characters per second.

*Robustness.* It will withstand even the application of 240 V mains to its input without permitting the signals to the ICL4130 to exceed their specified limits.

*Long-range operation.* Devices have operated reliably at over 300 ft from the computer. Maximum length of the connecting cable has not yet been determined.

*Compatibility.* The Mark I interface uses the same logical signals as a standard ICL4130 interface channel. It could, in principle, be used to operate a standard ICL4100 peripheral remotely.

*Expandability.* The Mark I interface is logically 'transparent' and of modular construction. Provision has thus been made for the incorporation of a logic module in the Mark II version to detect error states and take appropriate action, and also to perform certain simple operations.

The interface and its protection system have been approved by ICL; devices connected via the interface do not now need approval.

The interface and Freddy control logic have now correctly performed over 65 million operations, during normal public Multi-POP sessions.

### Computer communications

Data transferred between Freddy, the device, and the 4130 is in the form of single word transfers of 8 bits (ODUM or IDUM).

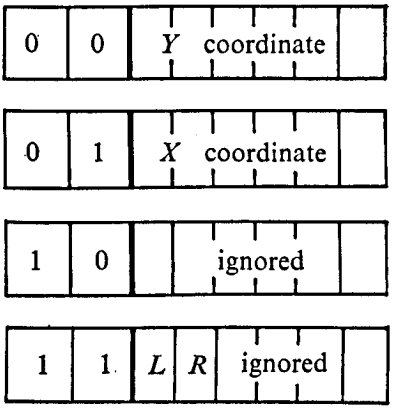
The POPMESS facility in Multi-POP gives the user a function doublet which outputs the least significant 8 bits of an item, or inputs a word from the robot.

```
e.g.   POPMESS([ROBOT])→RFN;
       X→RFN();                    for output,
or     RFN()→X;                    for input.
```

We now come to the decoding of commands and the encoding of sensory input.

### Output words

The most significant two bits of the 8 bit word specify the command type:



The commands 00 and 01 cause the least significant 6 bits of the word to be loaded into the *Y* or *X* coordinate register. The 00 command initiates the sampling of the picture.

The third and fourth significant bits of the 11 command specify the directions of drive of the two motors. When the 11 command is given, the motors are started. No further commands may be given until motion has ceased, when the 10 reset command must be given before another move can be made.

#### Input word

The input word is structured as follows:

| Motor flag | Left bump | Right bump | Picture flag | Comparator outputs |
|---|---|---|---|---|

The motor flag is cleared to 0 when the drive command is given, and is reset to 1 when motion ceases.

Bumper flags are 1 if in contact with an object.

The picture flag is cleared by the set *Y* command, and is reset to 1 when the sample has been taken.

The four least significant bits give the states of the threshold circuits. Five brightness levels are discriminable:

0000 = 0 decimal = level 0
0001 = 1 „ = „ 1
0011 = 3 „ = „ 2
0111 = 7 „ = „ 3
1111 = 15 „ = „ 4

Level 0 is black, level 4 is white.

#### Programming practicalities

*Motors.* The sequence of operations for taking a step or making a turn are as follows:

1. Give the appropriate 11 drive command.
2. Wait several milliseconds for the motor flag to fall.
3. Monitor the flag and wait for it to rise again.
4. Give the 10 reset command.
5. Wait for about ½ second for the drive circuitry to recover.

The waits (steps 2 and 5) can be implemented by the simple function:

```
FUNCTION WAIT N;
L: N-1→N; IF N>0 THEN GOTO L CLOSE;
END;
```

WAIT(100) is sufficient for operation 2. WAIT(700) for 5. To save central processor time during the flag monitoring, the SWAPOFF system function (which swaps the user off the time-sharing system) should be used thus:

```
L: .SWAPOFF;
IF NOT(LOGAND(.RFN, 8:200)) THEN GOTO L CLOSE;
```

*Picture sampling.* The sequence of operations is as follows:

1. Set $X$ coordinate with 01 command.
2. Set $Y$ and initiate with 00 command.
3. Monitor picture flag until it returns TRUE.
4. Read and decode picture intensity.

In this case it is not necessary to use SWAPOFF in the monitoring loop in 3 above.

$Y$ runs from 0 to 63 from the bottom of the picture upwards.

$X$ runs from 0 to 63 from left to right in the real world (from right to left on the TV monitor picture).

If the picture is sampled randomly, on the average it will be necessary to wait 10 msec for the picture scan to reach the chosen point (since the frame time is 20 msec). The time taken to sample all the available 4096 points is thus 40·96secs.

If the picture is scanned horizontally, row by row, this is almost the worst possible case. The wait for each point is now nearly 20 msec, and the overall time to read all the points is 81·92 sec.

A suitable scanning scheme can markedly reduce these times. By setting $X$ and sampling at several values of $Y$ from top to bottom it is possible to race the TV scan down the picture and take at least 10 samples per frame (20 msec). (The command structure has been chosen for maximum speed; as the most significant two bits of the set $Y$ command are zero, simply outputting the $Y$ coordinate will initiate the sample.) In this way, the time taken to read the whole picture can be less than 8 seconds, the actual time depending upon the amount of computing between samples.

## PROPOSALS FOR MARK II DEVICE

Our present Mark I is about the simplest possible machine and, as expected, is becoming obsolete. Its mechanical drawbacks are as follows:

(1) There are several awkward restrictions on possible movements. It cannot, for example, move to the perimeter of the world and then do an about turn.
(2) Rotating movements are slow because of the high inertia of the world.
(3) It is not possible to move directly sideways. It has to: left turn, forward, right turn. This makes building up pictures of an object from all sides very slow.

We have considered many schemes for overcoming these drawbacks. The overall scheme is now complete and many of the details are decided.

Mark II will retain the anchored robot principle for translations. The 'world' will consist of a five foot square of aluminium honeycomb or, perhaps, the present hardboard/polystyrene sandwich mounted on a compound $XY$ slide. The slides will be rails of $2\frac{1}{2} \times \frac{1}{4}$ light alloy with ball bearing constraints. The drive will be through stainless steel multi-strand wires from two static servomotors. The swept area will be 10 foot square. Maximum acceleration will be 0·1 g up to a speed of 10 inches/sec.

Above the world will be a bridge from which may be slung various eyes, or bodies. If rotations are required they will have to be done on this bridge. Several designs are being considered: In the simplest a single small camera mounted on a rotating vertical mast will be limited to the number of rotations allowed by its cable. A second possibility is the use of periscope optics in which the entrance element is rotated to cover a panorama and the resultant rotation of the picture removed with a half speed dove prism. This allows complete freedom of movement and uses little space in the world at the cost of optical complexity. It seems rather difficult but not impossible to extend the principle to binocular vision.

The third possibility is an integrated hand/eye/body system. It seems clear that fairly good range finding is wanted, and that there is a need for higher acuity at points of interest. These needs would be satisfied with some form of triclops arrangement as shown in figure 2.

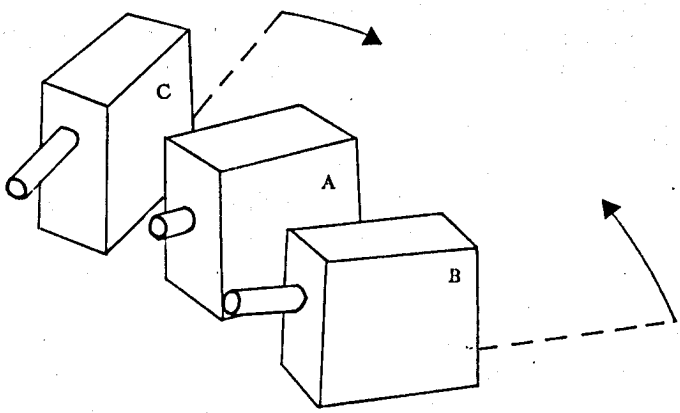



Figure 2. Three cameras for foveae and convergence ranging.

A is a television camera fitted with a wide angle lens, say 10 mm, which provides low acuity information of intensities over a 256 × 256 matrix on a 6 bit grey scale. This camera might well be a Pye Lynx, which could use a slightly improved version of our present Mark I sampling and conversion circuitry. Its function is to provide modest acuity wide angle coverage, similar to peripheral vision in humans.

On either side of this central camera are two others B and C, which are fitted with narrow angle lenses, say 100 mm, to provide the high acuity vision of the foveae. These cameras should really have image dissection tubes, but perhaps we may have to manage with vidicons. An actuator coupled to both outer cameras will cause them to rotate about vertical axes and so converge on objects at various distances. The angle of convergence necessary to fuse the two images will be a measure of the distance of the object. The moment of fusion is detected by a high positive correlation between the two video signals. It seems to be much more efficient to correlate outside the computer, using stochastic multipliers. In order to maintain focus over the required range of distance with such narrow angle lenses, it will be necessary to accommodate the outer pair of eyes. The cams and linkages required will be quite simple. The foveae cover small areas in the centre of the field. In order to examine and range any part, it is necessary to perform some sort of head movement, but before discussing this one should consider hands.

Figure 3 shows two pantograph linkages. The motion of H is a linear combination of the motions of $P_1$ and $P_2$, provided that $P_1$ and $P_2$, and H are co-linear. If $P_1$ and $P_1'$ are moved away from each other, then H and H′ move towards each other. If $P_2$ and $P_2'$ move together in the directions shown, then H and H′ are extended. If the palms of the hands are to be kept parallel then a secondary linkage will be necessary. The advantage of the pantograph linkage is that both actuators are static and well away from the palm position. Now imagine that the eye and hand assemblies are built separately on chassis a little larger than office filing boxes. In figure 4 the two chassis are joined by links at each side. Each chassis can be rotated relative to the link, which will allow the centre of the hand area to be moved vertically as seen by the eyes.

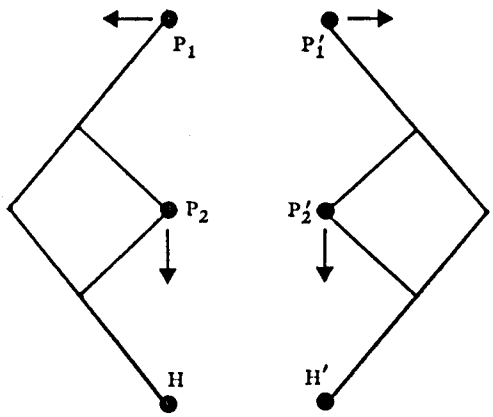



Figure 3. Pantograph arms

The link itself is able to rotate about its mid point on a bearing supported by a vertical fork. The fork hangs from a large horizontal beam and can rotate about a vertical axis. These last two motions are equivalent to body movement with eye and hand relatively locked. If this equipment is slung over the $XY$ table, we will have the minimum number of degrees of movement for free activity.

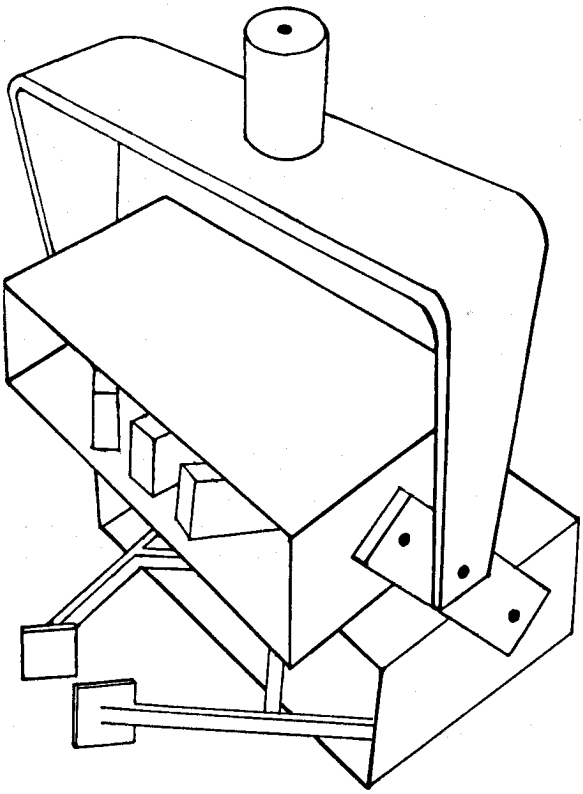


Figure 4. A possible hand/eye arrangement

## PRIMITIVE COMMANDS

The following is an excerpt from the documentation provided with our Departmental Multi-POP consoles.

POP-2 PROGRAM LIBRARY
PROGRAM SPECIFICATION

| | |
|---|---|
| PROGRAM NAME | LIB ROBOT PACK |
| SOURCE | H.G. BARROW, DMIP. |
| DATE OF ISSUE | September 1969 |

### Description

This package contains a set of basic functions for on-line operation of the receptors and effectors of the Mark I version of the robot Freddy.

### How to use the program

The package assumes that the function variable ROBOTDATA has been assigned the repeater function for the robot, so that this must be done before the package is compiled, namely:

```
VARS FUNCTION ROBOTDATA;
POPMESS([[ROBOT ⟨n⟩]])→ROBOTDATA;
```

where ⟨*n*⟩ is the time in minutes for which the robot is required.

Note that the result of this POPMESS is a doublet for both inputting from, and outputting to the robot.

The package is then compiled by typing:

```
COMPILE(LIBRARY([LIB ROBOT PACK]));
```

The following facilities will then be available:

*Constants.* A number of useful constants are defined in the package. They are listed later.

*Touch.* There are three Boolean functions which sample the two bump-detecting switches:

| | |
|---|---|
| BUMPLEFT() | TRUE if left bumper is operated, else FALSE |
| BUMPRIGHT() | „ „ right „ „ „ „ „ |
| ABUMP() | „ „ either „ „ „ „ „ |

Note that each bump detector responds to contact in front or behind. Where the object actually is can be determined by remembering which way Freddy was moving when the bump occurred.

*Movement.* There are two movement commands, each giving a Boolean result:

```
WALK(⟨distance in millimetres⟩);
TURN(⟨angle in radians⟩);
```

As moves are in fact quantized, the distance Freddy actually walks may differ from the specified distance by ±2·5 millimetres. If the WALK argument is negative, Freddy moves the specified distance in reverse.

Turns are similarly quantized, and actual angle turned may differ from that specified by ±0·035 radians. If the argument is positive, Freddy turns left, if negative he turns right.

The Boolean variable BUMPON determines whether or not Freddy stops on contact with an object.

If BUMPON is FALSE, he ignores bumps, and the results of WALK and TURN are always TRUE. (It is thus possible to push objects.)

If BUMPON is TRUE, then if no obstacles are encountered, he steps or turns the full distance and returns the result TRUE. If an obstacle is touched, he stops immediately and exits from the move function with the result FALSE.

The variable LASTWALK contains the distance actually moved during the last call of the WALK function. LASTTURN contains the angle turned during the last call of TURN. If no obstacles are encountered, the value will be approximately equal to the argument of the function. If a bump causes exit from the move function, the value will be the distance actually travelled (or angle turned) up to the moment of impact. (i.e., WALK(-LASTWALK); will return Freddy to his starting place before the last move).

*Dead reckoning.* The move routines endeavour to maintain a running estimate of Freddy's position and orientation. Position is measured in Cartesian coordinates, in millimetres. Orientation is measured in radians and is the angle between the $x$-axis and the direction in which Freddy is pointing. The angle is restricted to the range 0 to $2\pi$.

The estimates are held in the variables:

XNOW, YNOW, ANGNOW

(Typing CTRL and G during motion will only introduce an error of one quantum of angle or distance.)

The function SETPOSITION(X, Y, THETA); sets the estimates to the specified values.

The function PRPOSITION(); prints the current values of the estimates.

*Vision.* Note first that the TV camera looks into a mirror. What is seen on the TV screen is therefore a laterally inverted version of the real world. The following refers to the *real world* and not the TV monitor.

The picture is sampled at one of 4096 points in a $64 \times 64$ array. Bottom left is (0, 0), top left is (0, 63), top right is (63, 63) and bottom right is (63, 0).

The function PICINT(X, Y); samples the picture at point (X, Y) and returns an integer value for light intensity from 0 to NBRIGHTLEVEL. In the Mark I robot, there are five possible values, 0 to 4. Black is signified by 0, white by 4.

The function AVINT(X, Y, N) returns the mean over N samples of light intensity at (X, Y) to reduce effects of noise on the TV signal.

The function DISPLAY(FN, INC, X0, X1, Y0, Y1) prints a picture (via CUCHAROUT) using the characters:

⟨space⟩ , ↑ + * N M to simulate a grey scale.

FN is a function of two arguments, which yields a result (integer or real) in the range 0 to NBRIGHTLEVEL, and INC specifies the increment, and X0, X1, Y0, Y1 specify bounds for X and Y.

The function DISPIC is DISPLAY(%0, 63, 0, 63%) and thus will display the values of FN over the entire field.

e.g., DISPIC(PICINT, 8);

```
%%%
```

*Useful Constants (contd.)*

| | |
|---|---|
| MMPERWALK | quantum of forward motion, mm. |
| MMPERBWALK | ,, ,, backward ,, ,, (negative) |
| RADPERLTURN | ,, ,, left turn, radians. |
| RADPERRTURN | ,, ,, right ,, ,, (negative) |
| PICHAR (0 to 7) | array, holding characters printed for grey scale. |

*Useful Variables*

XNOW, YNOW, ANGNOW current estimates of position and orientation.

| | |
|---|---|
| LASTWALK | distance travelled during last call of WALK, mm. |
| LASTTURN | angle turned during last call of TURN, radians. |
| BUMPON | if TRUE, contact causes exit from move functions. |

*Useful Functions*

ABS result is absolute value of argument.

ABUMP, BUMPLEFT, BUMPRIGHT,
SETPOSITION, PRPOSITION,
WALK, TURN,
PICINT, AVINT,
DISPLAY, DISPIC,
STATS, PRSTATS.

*Other Globals*

Functions WAIT, FLAG, DRIVE, MOVEPAUSE, MOVEDONE, ANGUPDATE, WALKUPDATE, MOVEIT, CHARLOTS, ROBOTDATA.

**Store used**

The compiled program occupied about 4 blocks of core.
The uncompiled program occupies 17 sectors of disc.

**Acknowledgements**

The greater part of the hardware work was done in the Department's Bionics Research Laboratory directed by Professor R.L.Gregory and the programming and interfacing work in the Experimental Programming Unit directed by Professor D.Michie. The authors wish to acknowledge financial support from the Nuffield Foundation (S.H.S.) and from the Science Research Council (H.G.B.).

# APPENDIX

# Bibliography on Proving the Correctness of Computer Programs

Ralph L. London
Computer Sciences Department
University of Wisconsin

**Preface and acknowledgements**

As this bibliography shows, there is considerable research interest in the topic of proving the correctness of computer programs. Included in the bibliography are all published or unpublished papers, books, reports, theses, etc., known to me that are at all related to the topic. Material cited encompasses very abstract papers dealing with the transformation and equivalence of program schemata; proposals and methods for proving individual programs correct; papers proving the correctness of specific algorithms and programs; and work that has proofs as an incidental part of the paper. In other words, the range is from the very abstract to the very practical. An item is included unless it appeared obvious to me that it did not belong. Thus material of marginal relevance is included.

A complete citation was impossible to obtain for a few items. The best available information is given. An item is in the same language as that of the title, unless otherwise indicated at the end of a citation. Note that the contents of the Russian publication *Problemy Kibernetiki* appears in English translation as *Problems of Cybernetics;* similarly *Kibernetika* appears as *Cybernetics.* I would be most interested in learning of corrections and additional citations to update the bibliography.

This work was supported by the National Science Foundation under Grant GP-7069 and the Mathematics Research Center, United States Army, under Contract Number DA-31-124-ARO-D-462. The contributions of D. C. Cooper, J. W. de Bakker, D. I. Good, L. Green, Z. Manna, R. Milner, C. M. I. Rattray, C. S. Wells and D. Wood to compiling the bibliography are acknowledged without in any way holding them responsible for what has been included, excluded or omitted.

Abrahams, P. W., Machine verification of mathematical proof. SC. D. Thesis, Massachusetts Institute of Technology 1963; also *Mathematical Algorithms*, **1**(2)(1966)11–32; **1**(3)(1966)19–38; **2**(1967)28–79; and **3**(1968) 28–155.

Allen, F. E., Program optimization. *IBM research paper RC–1599; see also Annual review of automatic programming*, **5** (eds Halpern, M. I. & Shaw, C. J.). Oxford: Pergamon 1969.

Ashcroft, E. A., Functional programs as axiomatic theories. *Centre for computing and automation report no. 9*. London: Imperial College. Undated.

Baer, J. L., Graph models of computations in computer systems. *Department of engineering report no. 68–46*. University of California at Los Angeles 1968.

Balzer, R. M., Studies concerning minimal time solutions to the firing squad synchronization problem. PH.D. Thesis. Carnegie-Mellon University 1966.

Balzer, R. M., An 8-state minimal time solution to the firing squad synchronization problem. *Information & Control*, **10** (1967) 22–42.

Barron, D. W., Recursive techniques in programming. New York: American Elsevier 1968.

Basu, S. K., Transformation of program schemes to standard forms. *IEEE Conference record of the ninth annual symposium on switching and automata theory*, pp. 99–105. New York: IEEE 1968.

Basu, S. K., Transformation of program schemes to standard forms. *Technical report*. Carnegie-Mellon University 1968.

Basu, S. K., On reduction of program schemes. *SIAM Journal on Applied Mathematics*, **16**, (1968) 328–39.

Best, S. F., A proof that a demonstration program works. *Decision Systems Inc. internal document PR–0516*, pp. 37–61. 1967.

Block, H. D., The perceptron: a model for brain functioning, 1. *Reviews of Modern Physics*, **34** (1962) 123–35.

Blum, E. K., Towards a theory of semantics and compilers for programming languages. *Ass. comput. Mach. Symposium on Theory of Computing, 1969*, pp. 217–27. Updated version in *J. comput. & system Sci.*, **3** (1969) 248–75.

Bohm, C. & Jacopini, G., Flow diagrams, Turing machines and languages with only two formation rules. *Comm. Ass. comput. Mach.*, **9** (1966) 366–71; *see also* Cooper, D. C.

Burstall, R. M., Semantics of assignment. *Machine Intelligence 2*, pp. 3–20 (eds Dale, E. & Michie, D.). Edinburgh: Oliver & Boyd 1968.

Burstall, R. M., Proving properties of programs by structural induction. *Comput. J.*, **12** (1969) 41–8.

Burstall, R. M. & Landin, P. J., Programs and their proofs: An algebraic approach. *Machine Intelligence 4*, pp. 17–43 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press 1969.

Caviness, B. F., On canonical forms and simplification. PH. D. Thesis. Carnegie-Mellon University 1968.

Chartres, B. A. & Florentin, J. J., A universal syntax-directed top-down analyzer. *J. Ass. comput. Mach.*, **15** (1968) 447–64.

Cooper, D. C., The equivalence of certain computations. *Comput. J.*, **9** (1966) 45–52.

Cooper, D.C., Mathematical proofs about computer programs. *Machine Intelligence 1*, pp. 17–28 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver & Boyd 1967.

Cooper, D.C., Reduction of programs to a standard form by graph transformation. *Theory of graphs, international symposium, Rome, 1966*, pp. 57–68 (ed. Rosenstiehl, P.). New York: Gordon and Breach 1967.

Cooper, D.C., Bohm and Jacopini's reduction of flow charts. Letter to editor, *Comm. Ass. comput. Mach.*, **10** (1967) 463, 473; *see also* Bohm, C. & Jacopini, G.

Cooper, D.C., Some transformations and standard forms of graphs with applications to computer programs. *Machine Intelligence 2*, pp. 21–32 (eds Dale, E. & Michie, D.). Edinburgh: Oliver & Boyd 1968.

Cooper, D.C., Program scheme equivalences and second order logic. *Machine Intelligence 4*, pp. 3–15 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press 1969.

Cooper, D.C., Program schemes, programs and logic. *Computation Services Department Memorandum No. 6*. University College of Swansea 1969. (Presented at Journées d'études sur l'analyse syntaxique, Fontainebleau, France, March 1969.)

de Bakker, J.W., Axiomatics of simple assignment statements. *Mathematical Centre Tracts 94*. Amsterdam: Mathematisch Centrum 1968.

de Bakker, J.W., Semantics of programming languages. *Advances in Information Systems Science, 2*, pp. 173–227 (ed. Tou, J.T.). New York & London: Plenum Press 1969.

Dijkstra, E.W., Solution of a problem in concurrent programming control. *Comm. Ass. comput. Mach.*, **8** (1965) 569; *see also* Knuth, D.E.

Dijkstra, E.W., Co-operating sequential processes. *Programming Languages*, pp. 43–112 (ed. Genuys, F.). London and New York: Academic Press 1968.

Dijkstra, E.W., A constructive approach to the problem of program correctness, *B.I.T.*, **8** (1968) 174–86.

Dijkstra, E.W., Towards correct programming, EWD 241. Eindhoven: Mathematics Department, Technological University 1968.

Dijkstra, E.W., Goto statement considered harmful. Letter to Editor, *Comm. Ass. comput. Mach.*, **11** (1968) 147–8. Reply, *Comm. Ass. comput. Mach.*, **11** (1968) 538, 541; *see also* Rice, J.R.

Dijkstra, E.W., The structure of the 'THE' multiprogramming system. *Comm. Ass. comput. Mach.*, **11** (1968) 341–6.

Di Paola, R.A., A survey of Soviet work in the theory of computer programming. *Rand Corporation Memorandum RM–5424–PR* 1967.

Earley, J.C., Generating a recognizer for a BNF grammar. *Technical Report*. Carnegie-Mellon University 1965.

Elgot, C.C., Abstract algorithms and diagram closure. *Programming Languages*, pp. 1–42 (ed. Genuys, F.). London and New York: Academic Press 1968.

Engeler, E., Algorithmic properties of structures. *Mathematical Systems Theory*, **1** (1967) 183–95.

Engeler, E., *Formal Languages, Automata and Structures*. Chicago: Markham Publishing 1968.

Engeler, E., *Approximation by machines*. Zürich: Forschungsinstitut für Mathematik, ETH 1969.

Engeler, E., *Algorithmic approximations*. Zürich: Forschungsinstitut für Mathematik, ETH, undated.

Engeler, E., Proof theory and the accuracy of computations (to appear).

Ershov, A.P., Operator algorithms, I. *Problems of Cybernetics 3*, pp. 697–763 (trans. Kiss, G.R.). New York: Pergamon Press 1962.

Ershov, A.P., Operator algorithms, II (Basic programming constructions). *Problems of Cybernetics 8*, pp. 377–407. Washington D.C.: Joint publications research service, Department of Commerce 1964.

Ershov, A.P., Operator algorithms, III (On the operator schemes of Yanov). *Problemy Kibernetiki 20*, pp. 181–200. Moscow: Nauka 1967 (Russian). Also *Problems of Cybernetics* (in press).

Ershov, A.P. & Lyapunov, A.A., On the formalization of the notion of program. *Kibernetika (Kiev)*, **5** (1967) 40–57 (Russian).

Evans, A.Jr., Syntax analysis by a production language. Ph.D. Thesis. Carnegie-Mellon University 1965.

Fels, E.M., Kaluzhnin graphs and Yanov writs. *Logik und Logikkalkul*, pp. 159–78 (eds Kasbauer, M. & von Kutschera, F.). Freiburg/Munchen: Verlag Karl Alber 1962.

Florentin, J.J., Language definition and computer validation. *Machine Intelligence 3*, pp. 33–41 (ed. Michie, D.). Edinburgh: Edinburgh University Press 1968.

Floyd, R.W., Assigning meanings to programs. *Proceedings of a Symposium in Applied Mathematics, Vol. 19 – Mathematical Aspects of Computer Science*, pp. 19–32 (ed. Schwartz, J.T.). Providence, Rhode Island: American Mathematical Society 1967.

Floyd, R.W., The verifying compiler. *Computer Science Research Review*, pp. 18–19. Carnegie–Mellon University 1967.

Glebov, N.I., O stroenii klassa r-kriteriev zkvivalentnosti. *Problemy Kibernetiki, 9*. Moskva: Fizmatgiz 1962.

Glebov, N.I., On algebraic equivalence of subsets of categories. *Problems of Cybernetics, 8*, pp. 358–76. Washington, D.C.: Joint Publications Research Service, Department of Commerce 1964.

Glushkov, V.M., Automata theory and formal microprogram transformations. *Cybernetics*, **1** (1965) 1–8.

Goldstine, H.H. & von Neumann, J., Planning and coding problems for an electronic computer instrument, Part 2, Vols. 1–3. *John von Neumann, Collected Works Vol. 5*, pp. 80–235 (ed. Taub, A.H.). New York: Pergamon Press 1963. (*See* especially p. 92 for the use of an assertion box.)

Good, D.I., Toward the realization of a program proving system. PH.D. Thesis, to be submitted to the University of Wisconsin 1970.

Good, D.I. & London, R.L., Interval arithmetic for the Burroughs B5500: Four ALGOL procedures and proofs of their correctness. *Computer Sciences Technical Report No. 26.* University of Wisconsin 1968.

Good, D.I. & London, R.L., Computer interval arithmetic: Definition and proof of correct implementation. *J. Ass. comput. Mach.* (to appear).

Green, C., Application of theorem proving to problem solving. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 219–39 (eds Walker, D.E. & Norton, L.M.). New York: ACM 1969.

Habermann, A.N., On the harmonious co-operation of abstract machines. Doctoral Thesis. Technological University, Eindhoven, The Netherlands, 1967.

Habermann, A.N., Prevention of system deadlocks. *Comm. Ass. comput. Mach.*, **12** (1969) 373–7, 385.

Hoare, C.A.R., Set manipulation. *ALGOL Bulletin 27* (1968) 29–37.

Hoare, C.A.R., An axiomatic basis of computer programming. *Comm. Ass. comput. Mach.*, **12** (1969) 576–80, 583.

Hoare, C.A.R., Proof of programs: Partition and find. Department of Computer Science, The Queens University of Belfast 1969.

Hoare, C.A.R., Notes on a theory of types and date structuring, *WG.2.2 Paper* 1969.

Ianov, Y.I., On the equivalence and transformation of program schemes. *Comm. Ass. comput. Mach.*, **1** (1958) 8–12.

Ianov, Y.I., On matrix program schemes. *Comm. Ass. comput. Mach.*, **1** (1958) 3–6.

Ianov, Y.I., The logical schemes of algorithms. *Problems of Cybernetics, 1*, pp. 82–140 (trans. Nadler, M. *et al.*). New York: Pergamon Press 1960.

Ianov, Y. I., On transformations of program flow-diagrams. Joint Publications Research Service, 6868. Office of Technical Services 1962.

Ianov, Y. I., On local transformation of algorithm schemes. *Problemy Kibernetiki, 20.* Moscow: Nauka 1967 (Russian).

Igarashi, S., On the logical schemes of algorithms. *Information Processing in Japan*, **3** (1963) 12–18.

Igarashi, S., A formalization of the descriptions of languages and the related problems in a Gentzen-type formal system. *Research Notes*, Series 3, No. 80. Research Association of Applied Geometry 1964. (*See* especially pp. 28–9.)

Igarashi, S., An axiomatic approach to the equivalence problems of algorithms with applications. PH.D. Thesis. University of Tokyo 1964; *also Report of the Computer Centre, University of Tokyo*, **1** (1968) 1–101.

Igarashi, S., On the equivalence of programs represented by ALGOL-like statements. *Report of the Computer Centre, University of Tokyo*, **1** (1968), 103–18.

Igarashi, S., Comments on the formal treatment of types including sets. *ALGOL Bulletin, 29* (1968) 12–15.

Igarashi, S., Equivalence-theoretical treatment of verification conditions, *WG.2.2 (under TC2 of IFIP) Meeting*. Vienna, April 1969.

Ito, T., Some formal properties of a class of non-deterministic program schemata. *IEEE Conference Record of the Ninth Annual Symposium on Switching and Automata Theory*, pp. 85–98. New York: IEEE 1968.

Ito, T., Notes on theory of computation and pattern recognition. *Artificial Intelligence Memo No. 61*. Stanford University 1968.

Iturriaga, R., Contributions to mechanical mathematics. PH.D. Thesis, Carnegie-Mellon University 1967.

Iverson, K.E., *A Programming Language*. New York: John Wiley and Sons 1962.

Kaluzhnin, L.A., Algorithmization of mathematical problems. *Problems of Cybernetics*, **2**, pp. 371–91 (trans. Kiss, G.R.). New York: Pergamon Press 1961.

Kaplan, D.M., Correctness of a compiler for ALGOL-like programs. *Artificial Intelligence Memo No. 48*. Stanford University 1967.

Kaplan, D.M., Some completeness results in the mathematical theory of computation. *J. Ass. comput. Mach.*, **15** (1968) 124–34.

Kaplan, D.M., A formal theory concerning the equivalence of algorithms. *Artificial Intelligence Memo No. 59*. Stanford University 1968.

Kaplan, D.M., The formal theoretic analysis of strong equivalence for elemental programs. PH.D. Thesis. Stanford University 1968; *also Artificial Intelligence Memo No. 60*. Stanford University 1968.

Kaplan, D.M., Regular expressions and the equivalence of programs. *J. comput. & sys. Sci.*, **3** (1969) 361–86.

Kaplan, D.M., Recursion induction applied to generalized flowcharts. *Proc. 24th Nat. Ass. comput. Mach. Conf.*, pp. 491–504. New York: ACM 1969.

Karp, R.M., Some applications of logical syntax to digital computer programming. PH.D. Thesis. Harvard University 1959.

Karp, R.M., A note on the application of graph theory to digital computer programming. *Information and Control*, **3** (1960) 179–90.

Karp, R.M. & Miller, R.E., Properties of a model for parallel computations: Determinancy, termination, queuing. *SIAM Journal on Applied Mathematics*, **14** (1966) 1390–1411.

Karp, R.M. & Miller, R.E., Parallel program schemata: A mathematical model for parallel computation. *IEEE Conference Record of the Eighth Annual Symposium on Switching Theory and Automata Theory*, pp. 55–61. New York: IEEE 1967

Karp, R.M. & Miller, R.E., Parallel program schemata. *J. comput. & sys. Sci.*, **3** (1969) 147–95.

King, J.C., A program verifier. PH.D. Thesis. Carnegie-Mellon University 1969.

Knuth, D.E., Additional comments on a problem in concurrent programming control. Letter to Editor, *Comm. Ass. comput. Mach.*, **9** (1966) 321–2; *see also* Dijkstra, E.W.

Knuth, D.E., *The art of computer programming, Vol. 1 – Fundamental Algorithms*, pp. 14–20, 318–19. Reading, Mass.: Addison-Wesley 1968; *Vol. 2 – Seminumerical Algorithms*, pp. 245, 246, 508, 510. Addison-Wesley 1969.

Knuth, D.E., Semantics of context-free languages. *Mathematical Systems Theory*, **2** (1968) 127–45.

Krinitskiy, N.A., Ravnosilnye preobrazovaniya logicheskikh skhem. Avtoreferat Dissertatsii. MGU, Moskva 1959.

Kunze, J., Selektive Graphschemata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, **13** (1967) 101–22.

Laski, J.G., Sets and other types. *ALGOL Bulletin 27* (1968) 41–8.

Levy, S. & Paull, M.C., An algebra with applications to sorting algorithms. *Proceedings of the Third Annual Princeton Conference on Information Sciences and Systems* (1969) 286–91. Also *RCA Technical Report PTR–2176*. 1969.

London, R.L., A computer program for discovering and proving sequential recognition rules for well-formed formulas defined by a Backus normal form grammar. Ph.D. Thesis. Carnegie-Mellon University 1964.

London, R.L., A computer program for discovering and proving recognition rules for Backus normal form grammars. *Proc. 19th Nat. Ass. comput. Mach. Conf.* pp. A1.3-1 to A1. 3–7. New York: ACM 1964.

London, R.L., Correctness of the ALGOL procedure Askforhand. *Computer Sciences Technical Report No. 50*. University of Wisconsin 1968.

London, R.L., Computer programs can be proved correct. *Proceedings of Fourth Systems Symposium – Formal Systems and Non-numerical Problem Solving by Computers* (eds. Banerji, R.B. & Mesarovic, M.D.). Springer-Verlag (in press).

London, R.L., Proof of algorithms: A new kind of certification (Certification of Algorithm 245 Treesort 3). *Comm. Ass. comput. Mach.* (to appear).

London, R.L., Proving programs correct: Some techniques and examples. In press.

London, R.L. & Halton, J.H., Proofs of algorithms for asymptotic series. *Computer Sciences Technical Report No. 54A*. University of Wisconsin 1969.

London, R.L. & Wasserman, A.I., The anatomy of an ALGOL procedure. *Computer Sciences Technical Report No. 5*. University of Wisconsin 1967.

Lucas, P., Two constructive realizations of the block concept and their equivalence. *Technical Report 25.085*. Vienna: IBM Laboratory 1968.

Luckham, D.C., Investigation of the theory of algorithms with emphasis on program simplification. *Bolt, Beranek and Newman Inc., Report No. 1225*. 1965.

Luckham, D.C. & Park, D.M.R., The undecidability of the equivalence problem for program schemata. *Bolt, Beranek and Newman Inc., Report No. 1141.* 1964.

Luckham, D.C., Park, D.M.R. & Paterson, M.S., *On formalized computer programs.* Programming Research Group, Oxford University 1967.

Lyapunov, A.A., Logical schemes of programs. *Problems of Cybernetics Vol. 1,* pp. 48–81. (trans. Nadler, M. *et al.*). New York: Pergamon 1960.

McCarthy, J., Recursive functions of symbolic expressions and their computations by machine, Part 1. *Comm. Ass. comput. Mach.,* 3 (1960) 184–95.

McCarthy, J., Computer programs for checking mathematical proofs. *Proceedings of Symposia in Pure Mathematics, Vol. 5 – Recursive Function Theory,* pp. 219–27 (ed. Dekker, J.C.E.). Providence, Rhode Island: American Mathematical Society 1962.

McCarthy, J., Towards a mathematical science of computation. *Information processing; Proceedings of IFIP Congress 62,* pp. 21–8 (ed. Popplewell, C.M.). Amsterdam: North Holland 1963.

McCarthy, J., A basis for a mathematical theory of computation. *Computer Programming and Formal Systems,* pp. 33–70 (eds Braffort, P. & Hirschberg, D.). Amsterdam: North Holland 1963. Also *Proceedings of the Western Joint Computer Conference,* pp. 225–38. New York: Spartan Books 1961.

McCarthy, J., Problems in the theory of computation. *Proceedings of IFIP Congress 65,* pp. 219–22 (ed. Kalenich, W.). Washington, D.C.: Spartan Books Inc. 1965.

McCarthy, J., A formal description of a subset of ALGOL. *Formal Language Description Languages for Computer Programming,* pp. 1–12 (ed. Steel, T.B., Jr.). Amsterdam: North Holland 1966.

McCarthy, J. (ed.), *Papers in Theory of Computation* (Reprinted papers). Stanford University 1966.

McCarthy, J. & Hayes, P.J., Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4,* pp. 463–502 (*see* especially pp. 481–3) (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press 1969.

McCarthy, J. & Painter J.A., Correctness of a compiler for arithmetic expressions. *Proceedings of a Symposium in Applied Mathematics, Vol. 19 – Mathematical Aspects of Computer Science,* pp. 33–41 (ed. Schwartz, J.T.). Providence, Rhode Island: American Mathematical Society 1967.

Manna, Z., Termination of algorithms. PH.D. Thesis. Carnegie-Mellon University 1968.

Manna, Z., The correctness problem of computer programs. *Computer Science Research Review,* pp. 34–6. Carnegie-Mellon University 1968.

Manna, Z., Formalization of properties of program. *Artificial Intelligence Memo No. 64.* Stanford University 1968.

Manna, Z., Properties of programs and the first-order predicate calculus. *J. Ass. comput. Mach.*, **16** (1969) 244–55.

Manna, Z., The correctness of programs. *J. comput. & sys. Sci.*, **3** (1969) 119–27.

Manna, Z. & Pnueli, A., The validity problem of the 91-function. *Artificial Intelligence Memo No. 68.* Stanford University 1968.

Manna, Z. & Pnueli, A., Formalization of properties of recursively defined functions. *Ass. comput. Mach. Symposium on Theory of Computing*, pp. 201–10. New York: ACM 1969.

Markov, A. A., *Theory of Algorithms* (trans. Schorr-Kon, J.J.). No. OTS 60–51085. U.S. Department of Commerce, Office of Technical Services 1961.

Martynyuk, V.V., Ob analize grafa perekhodov dlya operatornoy skhemy. *Zhurnal Vychislitelnoy Matematiki I Matematicheskoy Fiziki*, **5** (1965) Moskva.

Martynyuk, V.V., O proverke zhvivalentnosti operatornykh skhem. *Tezisy Kratkikh Nauchykh Soobshcheniy Mezhdunarodnomu Kongressu Matematikov 1966 G.* (Sektsiya 14). Moskva 1966.

Maurer, W.D., A theory of computer instructions. *J. Ass. comput. Mach.*, **13** (1966) 226–35.

Milner, R., A partial decision procedure for $P \neq Q$, where P, Q are program schemes with non-intersecting loops. *Computation Services Department Memorandum No. 1.* University College of Swansea, Wales 1968.

Milner, R., Some equivalence relations on program schemes. *Computation Services Department Memorandum No. 5.* University College of Swansea, Wales 1969.

Munteanu, E., Metod analiva graf-skhemnykh algorifmov. *Mathematica*, **5** (28) (1965) 2.

Munteanu, E., Analyse logique des algorithms (1). *Mathematica*, **9** (32) (1967) 111–28.

Naur, P., Proof of algorithms by general snapshots. *B.I.T.*, **6** (1966) 310–16.

Nievergelt, J. & Irland, M.I., NUCLEOL as a formal system. Presented at *Sigplan Symposium on programming language definition.* 1969.

Nikitin, A.S., A class of equivalent transformations of operator schemes, 1. *Cybernetics*, **2** (1966) 51–7.

Oettinger, A.G., Automatic syntax analysis and the pushdown store. *Proceedings of a Symposium in Applied Mathematics, Vol. 12 – Structure of Language and its Mathematical Aspects*, pp. 104–29 (ed. Jakobson, R.). Providence, Rhode Island: American Mathematical Society 1961.

Orgass, R.J., A mathematical theory of computing machine structure and programming. PH.D. Thesis. Yale University 1967. Also *IBM Research Paper RC–1863.* 1967.

Orgass, R.J., Problems in the theory of programming. *IBM Research Paper RC–2236/8* 1967.

Orgass, R.J., Some results concerning proofs of statements about programs. *IBM Research Paper RC-2324*. 1969.

Orgass, R.J. & Fitch, F.B., A theory of programming languages. *IBM Research Paper RC-1979*. 1967.

Orgass, R.J. & Fitch, F.B., A theory of computing machines. *Studium Generale*, **22** (1969) 83–104; 113–36.

Painter, J.A., Semantic correctness of a compiler for an ALGOL-like language. *Artificial Intelligence Memo No. 44*. Stanford University 1967; *also* Ph.D. Thesis. Stanford University 1967.

Paterson, M S., Equivalence problems in a model of computation. Ph.D. Thesis. University of Cambridge 1967.

Paterson, M.S., Program schemata. *Machine Intelligence 3*, pp. 18–31 (ed. Michie, D.). Edinburgh: Edinburgh University Press 1968.

Perlis, A.J., Unpublished notes on proofs of three numeric algorithms. University of Michigan Summer Session 1963.

Podlovchenko, R.I., Primer ispolzovaniya preobrazovaniy logicheskikh skhem programm. *Problemy Kibernetiki*, **11**. Moskva: Fizmatgiz 1963.

Prosser, R.T., Applications of Boolean matrices to the analysis of flow diagrams. *Proceedings of the Eastern Joint Computer Conference*, pp. 133–8. New York: Spartan Books 1959.

Rechard, O.W. & Stark, R.H., Equivalence of two algorithms for Cooper's generalized factorial function. *Comput. J.*, **12** (1969) 33–7.

Rennie, M.K., Theory of procedures: 1, simple conditionals. *Notre Dame Journal of Formal Logic*, **10** (1969) 97–112. Abstract only in *Journal of Symbolic Logic*, **32** (1967) 577.

Rice, J.R., The **goto** statement reconsidered. Letter to Editor, *Comm. Ass. comput. Mach.*, **11** (1968) 538. *See also* Dijkstra, E.W.

Rosser, J.B., Schoenfeld, L. & Yohe, J.M., Rigorous computation and the zeros of the Riemann zeta-function. Invited paper, *Proceedings IFIP Congress 68*, pp. 13–18. Amsterdam: North Holland 1968.

Rutledge, J.D., On Ianov's program schemata. *J. Ass. comput. Mach.*, **11** (1964) 1–9.

Rutledge, J.D., The problem of correct programming. Undated.

Sanderson, J.G., A contribution to the theory of programming languages. Ph.D. Thesis. University of Adelaide 1966.

Sanderson, J.G., The theory of programming languages – a survey. *Proceedings of the Third Australian Computer Conference 1966*, pp. 321–4. Chippendale, New South Wales: Australian Trade Publications Pty Ltd 1967.

Sanderson, J.G., Mathematical models of programming languages. *J. Ass. comput. Mach.* (in press).

Sauer, G., Remark on algorithm 268. *Comm. Ass. comput. Mach.*, **12** (1969) 407.

Shirey, R. W., Implementation and analysis of efficient graph planarity testing algorithms. PH.D. Thesis. University of Wisconsin 1969. Also *Computer Sciences Corporation Report* 1969.

Slutz, D. R., The flow graph schemata model of parallel computation. PH.D. Thesis. Massachusetts Institute of Technology 1968.

Smirnov, Yu. I., O preobrazovanii operatornoy skhemy. *Zhurnal Vychislitelnoy Matematiki I Matematicheskoy Fiziki*, **3** (1963) No. 3. Moskva.

Stark, R. H., On means to record algorithms to facilitate generation of error-free programs. *Computing Center Report* 68–1, Washington State University 1968.

Thiele, H., Wissenschaftstheoretische Untersuchunger in Algorithmischen Sprachen, 1. Berlin: VEB Deutscher Verlag der Wissenschaften 1966.

Thorelli, L. E., An algorithm for computing all paths in a graph, *B.I.T.*, **6** (1966) 347–9.

Tobey, R. G., Rational function integration. PH.D. Thesis. Harvard University 1967.

Tonoyan, R. N., Logicheskie skhemy algoritmov i ikh zkvivalentnye preobrazovaniya. *Problemy Kibernetiki, Vol. 14.* Moskva: Izd–Vo Nauka 1964.

Van Wijngaarden, A., Numerical analysis as an independent science, *B.I.T.*, **6** (1966) 66–81.

Van Wijngaarden, A., Mailloux, B. J., Peck, J. E. L. & Koster, C. H. A. (eds), Final draft report on the algorithmic language ALGOL 68. *Mathematisch Centrum Report M R 100.* Amsterdam 1968.

Wagner, R. A., Some techniques for algorithm optimization with application to matrix arithmetic expressions. PH.D. Thesis. Carnegie-Mellon University 1968.

Waldinger, R. J. & Lee, R. C. T., PROW: A step toward automatic program writing. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 241–52 (eds Walker, D. E. & Norton, L. M.). New York: ACM 1969.

Walters, D. A., A representation system for parsing procedures. *RCA Laboratories Scientific Report No. 1.* 1968. *Also* PH.D. Thesis. University of Pennsylvania 1968.

Wirth, N., On certain basic concepts of programming languages. *Computer Science Technical Report No. CS65.* Stanford University 1967.

Wood, D., A proof of Hamblin's algorithm for translation of arithmetic expressions from infix to postfix form. *B.I.T.*, **9** (1969) 59–68.

Wood, D., The theory of left factored languages: Part 1. *Comput. J.*, **12** (1969) 349–56. Part 2 (in press).

Yanov, Yu. I. *See* Ianov, Y. I.

Yohe, J. M., Computer programming for accuracy. *Mathematics Research Center Technical Summary Report No. 866.* University of Wisconsin 1968.

Zama, N., On models of algorithms and flowcharts. *Commentarii Mathematici Universitatis Sancti Pauli*, **14** (1966) 123–34.

Zama, N., On a generalization of algorithms and one of its applications (1). *Commentarii Mathematici Universitatis Sancti Pauli*, **15** (1967) 109–16.

Zama, N., On some algebraic formulation of algorithms. *Commentarii Mathematici Universitatis Sancti Pauli*, **17** (1968) 53–62.

Zaslavskii, I. D., Graph-schemes with memory. *Matematycheskogo Instuta Imeni V. A. Steklova*, LXXII, pp. 99–192 (Russian).

**Added in proof**

Burstall, R. M., Formal description of program structure and semantics in first order logic. *Machine Intelligence 5*, pp. 79–98 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press 1970.

Engeler, E., Provability for programs. Unpublished 1969.

Manna, Z., The correctness of non-deterministic programs. *Artificial Intelligence Memo No. 95*. Stanford University 1969.

Rattray, C. M. I., Aspectos teoricos de la comprobacion de programas de ordenador. *Estudios Empresariales*, **13** (1969) 83–93.

Waldinger, R. J., Constructing programs automatically using theorem proving. Ph.D. Thesis. Carnegie-Mellon University 1969.

# INDEX

# INDEX


ACE 3, 6, 8, 12
Achieser 501, 515
Ackermann 166, 177
adaline 204, 205
Aizerman 503, 512, 515
Alexander 363, 375, 380
algebra 99, 105, 106, 108-13, 116-20, 321, 535
ALGOL-60 27, 28, 80, 90, 93, 95, 100, 101, 105, 116, 126, 304, 424, 471
Allen, C. D. 79, 96
Allen, J. 321, 536, 551
Alt 416, 431
Amarel 219, 220, 235, 520, 525, 527, 529
Anderson, A. R. 550, 552
Anderson, D. B. 96, 551
Andreae 463, 477, 492, 523, 528, 529
Andrews 165, 176, 325, 326, 328, 336
Antony 464, 492
Arbib 473, 492
Arnoult 462
association analysis 361, 368, 370
associative net 351, 355-8
Ashcroft 59, 77, 79, 96
Atlas 470
Attneave 451, 462
Austin 203, 208
automata theory 99, 105, 106
  generalized 106
Averbach 543, 552

B5000 118
Balashek 492
Banerji 520, 529
Bar-Hillel 219, 236, 540, 552
Barrow 555, 562
Baumert 228, 236
Becker 528, 529
Bekić 64, 120
Belar 466, 493
Bellman 372, 374, 380
Belnap Jr, N. D. 550, 552
Bennett 336
Bird 120
Birkhoff 49, 52, 136, 154, 163
Blaydon 502, 503, 509, 512, 515
Bloom 476, 493
Blum, H. 427, 431, 440
Blum, J. 524, 530
boundary analysis 438, 447, 453, 454
Braithwaite 540, 552
Braverman 503, 512, 515
Bruner 203, 208
Buchanan 176, 253, 280
Buneman 351, 359, 383, 406
Bursky 220, 236
Burstall 35, 37, 70, 77, 79, 80, 96, 105, 120, 154, 163, 182, 200, 219, 236, 281, 298, 520, 529, 551
Butler 413, 431

Cahn 432
Cameron 462
Carnap 176, 177
Carson 151, 177, 181, 182, 200, 321, 325, 327, 328, 336, 515, 553
Cashin 492, 523, 528, 529
Chambers 317, 318, 522, 531
chess 5, 13, 23, 219, 256, 257, 289, 303
Chien 499, 503, 515
Chinthayamma 322, 331, 336
Chomsky 175, 177, 383, 390
chromosome 411-34, 435-61
Church 4, 23, 63, 77
Church's theorem 136
Churchman 280
Clegg 438, 462
Clowes 476, 492, 529
Cohn 105, 120
Colby 528, 530
combinatory logic 125
computation
  rule 27
  sequence 28-30, 34
conic
  section 411ff.
  skeleton 419, 421, 427, 428
Conway 462

Cooper, D. C. 59, 60, 77
Cooper, F. S. 493
Coriell 543, 552
correlograph 351-5, 357, 358
Court Brown 412, 431
Cox 493
Crocker 219, 236
cross-resolution 147, 148, 151
Cuenod 496, 515
Curry 125, 132

Darlington 165, 166, 177
Davidson 540, 552
de Bakker 64, 70
Dekker 47, 52
Delattre 493
Delfino 280
DENDRAL 176, 253ff.
Denes 470, 492, 543, 552
Dersch 471, 492
De Soto 529, 530
Deutsch 523, 528, 530
Devyaterikov 503, 515
diagonal search 173, 181, 189, 190, 192, 193, 197, 200
directed graph *see* graph
Djerassi 255, 263, 280
Doran 183, 200, 207, 208, 219-21, 236, 282, 290, 298, 301, 304, 307, 309, 311, 312, 318, 519, 520, 524, 528, 530, 551, 552
Doshita 493
Dudley 471, 492
Duffield 255, 280

Eastlake 219, 236
Elcock 203, 208
ENIAC 6
entropy 361, 363-79
epsilon operator 126
equation schema 60-3, 66, 71, 73, 74
E-resolution 127
Ernst 302, 318, 519, 520, 530
error 219, 225, 228, 230-2, 437, 441, 498-510, 512, 514
ESOTERIC 463, 468, 472, 474, 477-9, 485-90
evaluation function 221, 233, 291, 302-5, 307, 309, 311, *see also* heuristic
Evans, E. F. 466, 476, 493
Evans, H. J. 412, 431, 451, 462
Evans, T. G. 520, 530

Faddeeva 507, 515
Fant 466, 492
Farrow 462
fast Fourier transform 403, 458
Feigenbaum 176, 219, 236, 253, 280, 530
Feldman 203, 204, 208, 530, 543, 552
Feller 228, 236
Fenichel 337-9, 345
Feys 125, 132
FIDAC 416
first-order logic 27, 59-63, 72, 73, 76, 79, 80, 95, 125, 132, 153, 165, 170, 175, 205, 237, 250, 322, 323, 525, 529, 535, 541-3, 545, 547, 549-51, 553
fixpoint induction 59, 60, 66, 67, 70, 73, 74, 77
Fleming 220, 236, 520, 531
Florentin 79, 96
Floyd 59, 60, 65, 71, 73, 77, 79, 96, 149, 151, 228, 236
FORTRAN IV 507
Fogel 203, 208
Freddy 555-7, 563-5
Freeman 451, 462
Friedman 524, 530
Fu 499, 503, 515

Galanter 527, 531
Gallus 413, 431, 437, 438, 443, 448, 455, 456, 462
Gandy 1
Garland 540, 553
Gelernter 525, 530
Gerstman 493
Gips 208
Gödel 4, 23, 166
Gödel's theorem 4
Gold 466, 492
Goldberg 462
Golomb 228, 236
Good 375, 380
Goodman 537, 538, 551, 552
Goodnow 203, 208
Gould 535, 552
GPS 219, 267, 269, 270, 520, 521, 524-6
grammar 204, 383 ff, 474, 478
graph 254, 260, 298, 413, 447
  bar 253, 272
  directed 99-104, 108, 148, 219, 224, 547
  matching 276, 277
  of function 74
  problem 303, 304, 307, 312, 520-3
  resolution 406, 407
  search 181
  theorem-proving 181-5
  traverser 207, 211, 213, 220, 222, 223, 233, 281, 282, 290-4, 297, 300, 301 ff, 498, 520-2, 524, 527
Grau 322, 331, 332, 334, 336
Green, C. C. 79, 80, 96, 165, 166, 177, 182, 187, 200, 237, 252, 323, 336, 520, 525-7, 530, 533, 534, 540, 541, 543, 551, 552
Green, D. K. 462
Green, H. 493
Greenblatt 219, 236, 256

Gregory 566
Grzegorczyk 546, 549, 550, 552
GT 4 301, 305, 307, 311
Guard 321, 334, 336, 536, 539, 540, 552
Guiliano 416, 431
Guzmán 476, 492

Hall 495, 496, 501, 502, 512, 514, 515
Halle 466, 492
Halmos 418, 432
Hammersley 500, 515
Handel 529, 530
Handscomb 499, 500, 515
Hart 181, 187-9, 195, 196, 198, 200, 220, 221, 233, 236
Hayes 27, 32, 165, 173, 176, 177, 182, 200, 203, 208, 526, 528-31, 533, 540, 543, 546, 552
Heicke 493
Henkin 63, 64
heuristic 77, 181, 188, 189, 191, 192, 194, 195, 197-99, 220-7, 229, 233-5, 256, 265, 274, 278, 291, 316, 497, 519-22, 524, 525, 528, 536, 539, *see also* evaluation function
  search 181, 219, 222, 224, 282, 302, 303
HEURISTIC DENDRAL, *see* DENDRAL
Hext 301, 318
Heydorn 374
higher order logic 95, 123, 125, 132, 166, 181, 243, 535, 537, 538
Hilbert 126, 166, 177
Hilditch 384, 390, 413, 416, 427, 432, 446, 462
Hill 463, 466, 470, 472, 475, 477, 479, 480, 490-2
Himsworth 301, 318
Hoare 79, 96
Holmes 465, 492
Hooke 301, 302, 308, 318
Hormann 519, 524, 530
Horning 208
HPA 221-34
Hu 416, 432
Hubel 466, 492
Hubermann 526, 530
Hungerford 462
Hunt 519, 532, 536, 553
Huttenlocher 529, 530
hyper-resolution 168, 535
hysteresis 469, 472, 479-83, 485, 486, 491

IBM 360 327
IBM 704 118
IBM 7094 431
ICL 4130 173, 307, 555-7
induction 153, 162, 165, 170, 175, 176, 179, 203-6, 497, *see also* fixpoint, mathematical, recursion, transfinite
Inoue 493
intuitionistic logic 549, 550, 553, 554

Jacobs 416, 432
Jakobson 466, 492
Jeeves 301, 302, 308, 318

Karp 104, 106, 118, 120
Kashyap 502, 503, 509, 512, 515
Kendall 502, 505, 515
Kerse 96
Khinchin 364, 377, 380
Kieburtz 326, 327, 336
Kilmer 524, 530
Kirsch 427, 432
Kleene 27, 32, 118, 552, 553
Kline 496, 515
Knaster-Tarski theorem 60, 64, 65
König's lemma 46, 188
Kopf 493
Kowalski 165, 173, 176, 177, 181, 182, 200, 551
Kozdrowicki 220, 236
Kripke 546, 547, 549, 552
Kuehner, 176, 192, 200
Kullback 375, 380

Lambert 368, 369, 380
lamda calculus 123-5, 128, 129, 131, 132, 535
Lanczos 502, 515
Landin 80, 96, 99, 105, 116, 120
lattice 40, 48-51, 56, 60, 64, 65, 68, 70, 76, 135, 136, 140-6, 163, 172, 173, 175, 179, 321
Lauchli 538, 552
Lauer 80, 96
Lawrence 464, 492
Lechner 398, 408
Lederberg 254, 280
Ledley 404, 408, 413, 416, 432, 437, 462
Lee 165, 177
Levan 411, 432
Levien 238, 252
Lewis 540, 552
Liberman 466, 492, 493
Lin 310, 318
Lindenbaum algebra 154
Lindsay 238, 252
LISP 69, 70, 93, 126, 132, 204, 205, 226, 254, 257-9, 261, 262, 264, 269, 275, 276, 278, 279, 323, 325, 327, 337-40, 344
list
  processing 80, 93, 453
  structure 69, 277
Littlepage 438, 462

logic, *see* combinatory, first-order, higher-order, intuitionistic, modal, partial function, second-order
London 529, 530
Longuet-Higgins 313, 318, 351, 359
Loveland 165, 177, 325, 326, 336
Łukasiewicz 32, 37, 542, 552
Luckham 39, 41, 42, 44, 45, 51, 52, 59, 77, 321, 323-8, 336, 536, 551

McBride 337
Maccacaro 366, 380
McCarthy 27, 28, 32, 40, 45, 52, 60, 61, 69, 71, 74, 77, 79, 80, 97, 203, 208, 226, 236, 282, 526, 528, 530, 531, 533, 534, 538, 540, 543, 546, 551, 552
McCormick 427, 432
McCulloch 524, 530
Mackay 473, 492
MacLane 49, 52, 154, 163
Malmnäs 553, 554
Manna 27, 28, 59-61, 71, 73-5, 77, 79, 80, 97
Maron 238, 252
Marril 476, 493
Marsh 281, 291, 298, 305, 318, 520, 522, 530
Martin 477, 493
mass spectrometry 253 ff
Massey 541, 550, 552
mathematical induction 60, 65, 70, 84
Mattingly 492
Mayall 462
Mead 301, 318
Meltzer 96, 163, 165, 177, 200, 408, 497, 551
memo function 281-8, 290, 291, 293, 295, 297, 298, 455, 522
Mendelsohn 413, 432, 436, 439, 462
Michie 183, 200, 207, 208, 219-21, 235, 236, 281, 282, 290, 291, 298, 301, 302, 308, 311-13, 317, 318, 498, 520, 522, 524, 527, 529, 530, 531, 551, 566
Miller, G. A. 465, 527, 531
Miller, W. F. 235
Milner 39, 42, 52, 77
Minski 104, 120, 219, 236, 383, 390, 408, 524, 525, 531, 538, 543, 551, 552
modal logic 540-2, 544-7, 549, 553, 554
Montague 546, 549, 552, 553
Montanaro 443, 462
Moore 222, 236
Moorhead 416, 432
Moray 476, 493
Morris 127, 132
Morrison 337
Moses 520, 531
Mowshowitz 370, 380
Multi-POP 555-7, 562
Murphy 543, 553
Murray 203, 208

Nagy 510, 512, 514, 515
Narasimhan 427, 432
Neisser 477, 493
Nelder 301, 318
Nelson 493
Neurath 413, 432, 437, 455, 456, 462
Newell 219, 236, 302, 318, 519, 524, 530-1
Nilsson 181, 187-9, 195, 196, 198, 200, 220, 221, 233-7, 252, 323, 336, 510, 512, 514, 515, 520, 524, 525, 528, 531

Occam's razor 206, 479
O'Connor 466, 493
Oglesby 336
Oldfield 220, 236, 520, 531
Olson 466, 493
organic chemistry 253 ff
Ortony 313, 318
Owens 203, 208

P1-deduction 173, 325
Painter 70, 77, 79, 97
Paley 405, 408
Papert 383, 390, 408, 524, 531
paramodulation 125, 127, 165, 171, 328, 329, 334
Park 39, 41, 42, 44, 45, 51, 52, 59
partial function 27-9, 32, 33-6, 39, 59, 65, 71, 74, 116, 304
  logic 27, 30
PAT 464
Patau 427, 432
Paterson 39-42, 44, 45, 48, 51, 52, 59, 60, 78
path search 136, 148
Paton 411, 428, 432, 512
pattern search 301, 302, 308, 310
PDP 8 470
PDP 10 257, 327
Pease 403, 408
Pengelly 337
perceptron 204, 384, 464
Perry 462
Pfaltz 427, 428, 432
Philbrick 427, 432
Pierce 175
Pitrat 168, 177
Plotkin 138, 153, 176, 177
Pneuli 28, 59-61, 71, 73-5, 77
Pohl 181, 188, 200, 219, 220, 235, 236, 298, 520, 531
Polyak 408
polygraph 105, 108, 109, 113, 114, 116, 119
POP-2 126, 173, 192, 281, 282, 283, 285, 291, 298, 299, 305-7, 562

Popplestone 97, 98, 153, 163, 203, 207, 208, 281, 290, 298, 497, 520, 522, 527, 531
Post 47, 149, 151
Post's correspondence problem 136, 149, 150
Potter 468, 493
Powell 308, 318
Prawitz 535, 553, 554
predicate calculus, *see* logic
Prewilt 462
Pribram 527, 531
Prior 542, 553
program scheme 39-42, 44-6, 48, 49, 51, 52, 59, 60, 71, 77, 106
property structure 237-40, 242, 250
Propoi 503, 515
PROSE 176

Quillian 528, 532
Quine 536, 537, 539, 541, 544, 553
Quinlan 302, 318, 519, 522, 532, 536, 553

Raphael 181, 187-9, 195, 196, 198, 200, 220, 221, 233, 236-8, 252, 323, 336, 524, 528, 531
Ray 432
recursion
  equation scheme 40, 45, 48, 51
  induction 34, 60, 65, 71, 74, 75
recursive function theory 39, 40, 44, 46-8, 65
Reddy 466, 469, 470, 471, 493
Reder 208
Regliosi 455, 462
representation problem 211, 512, 520
Rescher 540, 553
Rescigno 366, 380
resolution 406, 407, 436
  principle 96, 125, 127-9, 135, 136, 146, 147, 165-73, 181, 182, 185-7, 189, 200, 237, 240, 322, 324, 325, 525, 528, 535, 541
retina 383-5, 387, 391, 395, 404-6, 408, 409, 511, 512, 565
Reynolds 135, 151, 153, 154, 163, 328
Robertson 280
Robinson, G. 127, 132, 151, 165, 177, 181, 182, 201, 321, 322, 324, 325, 327, 328, 334, 336, 535, 553
Robinson, J. A. 95, 97, 123, 125, 131-3, 135-8, 147, 151, 154, 155, 159, 163, 165, 166, 168, 171, 173, 177, 181, 182, 200, 201, 237, 252, 322-5, 336, 535, 551, 553
robot 1, 12, 187, 519-25, 527, 528, 534, 536-9, 541-8, 550, 551, 555-7, 560, 562-5
Rogers, C. A. 515
Rogers Jr, H. 43, 46, 48, 52, 65, 78
Rosenbloom 125, 133
Rosenfeld 427, 428, 432
Ross 301, 308, 318, 498, 520, 522, 531
Rozonoer 503, 512, 515
Ruddle 432
Russel 22
Rutovitz 413, 417, 431, 432, 435, 436, 447, 454, 462, 512

Sage 496, 515
Sakai 464, 493
Salmon 428, 432
Salter 555
Samuel 203, 208, 219, 236, 281, 293, 297, 298, 301, 307, 317, 318, 477, 493, 522, 532
San Diego problem 526
Sandewall 182, 183, 200, 220, 236, 303, 312, 318, 519, 520, 525, 532
Sayers 3
Schnelle 493
Schofield 311, 318
Schönfinkel 123-6, 131-3
Schreider 408
Schroll 255, 280
Scott 64, 70
search strategy 181-3, 186-8, 190-2, 198, 199, 266-8, 270, 290, 322
Searle 395, 512
Sebestyen 510, 512, 514, 515
second-order logic 60-3, 68, 70, 166, 537, 538
Selfridge 477, 493
sequential logic 477
set of support 181, 325-7, 535
Settle 336
Shalla 336
Shannon 367
Shaw 219, 236, 302, 318, 519, 524, 531
Shearme 492
Sibert 127, 133, 182, 200
Siefkes 538, 553
signature table 317, 477
Simon 219, 236, 302, 318, 519, 524, 531, 538, 548, 553
simplex 301
SIR 237
skeletonizing 435, 446, 462
Slagle 165, 177, 182, 220, 236, 237, 252
Smith, D. C. 528, 530
Smith, I. 95, 173, 176, 192, 200
Smith, P. 462
speech 463-6, 468, 472, 477-9, 485, 489
Spendley 301, 318
Spicer 431
Spinelli 478, 493
Steedman 390
STELLA 463-5, 523

Stone 438, 440, 462
Strachey 80, 88, 96, 97
Strawson 533, 551, 553
structural induction 70
Stuart 502, 505, 515
subsumption 148, 153, 154, 159, 162, 182, 327-9
Sutherland 176, 253, 280
Sweeney 451, 462
systems theory 495, 497 ff.
Szewczyk 431
Sz-Nagy 500, 501, 515

Tarski 78
Taylor 515
television 12, 13, 396, 528, 556, 559, 560, 564
Thatcher 106, 120
theorem proving 1, 59, 84, 95, 96, 98, 125, 127, 129, 132, 135, 165, 166, 168-71, 173, 174, 176, 181-6, 195, 207, 237, 276, 321, 323, 520, 525, 528, 534-6, 539, 540, 544, 551
Tillman 471, 493
Tjio 411, 432
Toda 523, 532, 551, 553
topology 383-5, 390, 406
Tou 368, 374, 380
transfinite induction 70
transformational system 135, 136, 147-50
Travis 524, 532
tree 182-6, 221, 233, 277, 298, 305, 383, 384, 390-3, 406, 408, 447, 453
  binary 227-32
  generation 268, 274
  labelled 105, 106, 113, 118
  lookahead 290, 293, 295-7, 306, 317, 548
  planning 294, 521, 522
  proof 326-33
  regular 225, 226
  search 299, 300, 308, 311, 521
tree search 136, 148, 181, 182, 183, 186, 189
Turing 1, 3, 4, 7, 23
Turing machine 6-9, 12, 18, 40, 42, 44, 48, 51, 52, 55
Tsypkin 503, 515

Uhr 464, 478, 493
Ungeheuer 493
Urban 432

Van Emden 288, 361, 512
Vicens 471, 493
Vigor 176, 177
Vossler 464, 478, 493

Wacker 463
Wagner 503, 515
Waldinger 165, 177
Walsh, J. L. 396, 408
Walsh, M. J. 203, 208
Walsh function 395-406, 514
Wang 27, 32
Watanabe 366-8, 374, 380
Waterman 274, 280
Weiner 48, 51, 52
Whitfield 466, 476, 493
Widrow 204, 208
Wiener 514
Wiesel 466, 492
Williams 368, 369, 380
Willshaw 351, 359
Wright 106, 120
Wos 127, 132, 151, 177, 181, 182, 200, 321, 322, 324, 325, 327, 328, 336, 535, 553

Yarbus 543, 553

Zadell 493

# MACHINE INTELLIGENCE, VOLS 1, 2, 3, 4

The contents of Machine Intelligence, Volumes 1 to 4, are detailed in the following pages.

# MACHINE INTELLIGENCE 1 (1967) CONTENTS


PREFACE v

INTRODUCTION ix

ABSTRACT FOUNDATIONS

1 Linear graphs and trees. H.I.SCOINS 3
2 Mathematical proofs about computer programs. D.C.COOPER 17

THEOREM PROVING

3 Beth-tree methods in automatic theorem-proving. R.J.POPPLESTONE 31
4 The resolution principle in theorem-proving. D.LUCKHAM 47

MACHINE LEARNING AND HEURISTIC PROGRAMMING

5 Tree-searching methods with an application to a network design problem. R.M.BURSTALL 65
6 Experiments with a learning component in a Go-Moku playing program. E.W.ELCOCK and A.M.MURRAY 87
7 An approach to automatic problem-solving. J.DORAN 105
8 Complete solution of the 'Eight-Puzzle'. P.D.A.SCHOFIELD 125
9 Strategy-building with the Graph Traverser. D.MICHIE 135

COGNITIVE PROCESSES: METHODS AND MODELS

10 Networks as models of word storage. G.R.KISS 155
11 Will seeing machines have illusions? R.L.GREGORY 169

PATTERN RECOGNITION

12 Perception, picture processing and computers. Dr M. B. CLOWES 181
13 Automatic speech recognition: a problem for machine intelligence. D.R.HILL 199

PROBLEM-ORIENTED LANGUAGES

14 Simply partitioned data structures: the compiler-compiler re-examined. R.A.BROOKER and J.S.ROHL 229
15 The third-order compiler: a context for free man-machine communication. R.B.E.NAPPER 241
16 Principles for implementing useful subsets of advanced programming languages. G.F.COULOURIS 257
17 Interrogation languages. J.M.FOSTER 267

SUBJECT INDEX 277

AUTHOR INDEX 278

# MACHINE INTELLIGENCE 2 (1968) CONTENTS


PREFACE v

INTRODUCTION ix

ABSTRACT FOUNDATIONS

1 Semantics of assignment. R. M. BURSTALL 3
2 Some transformations and standard forms of graphs, with applications to computer programs. D. C. COOPER 21
3 Data representation–the key to conceptualisation. D. B. VIGOR 33

MECHANISED MATHEMATICS

4 An approach to analytic integration using ordered algebraic expressions. L. I. HODGSON 47
5 Some theorem-proving strategies based on the resolution principle. J. L. DARLINGTON 57

MACHINE LEARNING AND HEURISTIC PROGRAMMING

6 Automatic description and recognition of board patterns in Go-Moku. A. M. MURRAY and E. W. ELCOCK 75
7 A five-year plan for automatic chess. I. J. GOOD 89
8 New developments of the Graph Traverser. J. DORAN 119
9 BOXES: an experiment in adaptive control. D. MICHIE and R. A. CHAMBERS 137
10 A regression analysis program incorporating heuristic term selection. J. S. COLLINS 153

COGNITIVE PROCESSES: METHODS AND MODELS

11 A limited dictionary for syntactic analysis. P. BRATLEY and D. J. DAKIN 173

PROBLEM-ORIENTED LANGUAGES

12 POP-1: an on-line language. R. J. POPPLESTONE 185
13 Self-improvement in query languages. J. M. FOSTER 195
14 POP-2 reference manual. R. M. BURSTALL and R. J. POPPLESTONE 205

INDEX 250

# MACHINE INTELLIGENCE 3 (1968) CONTENTS


INTRODUCTION ix

MATHEMATICAL FOUNDATIONS

1 The morphology of prex – an essay in meta-algorithmics. J. LASKI 3
2 Program schemata. M. S. PATERSON 19
3 Language definition and compiler validation. J. J. FLORENTIN 33
4 Placing trees in lexicographic order. H. I. SCOINS 43

THEOREM PROVING

5 A new look at mathematics and its mechanization. B. MELTZER 63
6 Some notes on resolution strategies. B. MELTZER 71
7 The generalized resolution principle. J. A. ROBINSON 77
8 Some tree-paring strategies for theorem proving. D. LUCKHAM 95
9 Automatic theorem proving with equality substitutions and mathematical induction. J. L. DARLINGTON 113

MACHINE LEARNING AND HEURISTIC PROGRAMMING

10 On representations of problems of reasoning about actions. S. AMAREL 131
11 Descriptions. E. W. ELCOCK 173
12 Kalah on Atlas. A. G. BELL 181
13 Experiments with a pleasure-seeking automaton. J. E. DORAN 195
14 Collective behaviour and control problems. V. I. VARSHAVSKY 217

MAN–MACHINE INTERACTION

15 A comparison of heuristic, interactive, and unaided methods of solving a shortest-route problem. D. MICHIE, J. G. FLEMING and J. V. OLDFIELD 245
16 Interactive programming at Carnegie Tech. A. H. BOND 257
17 Maintenance of large computer systems – the engineer's assistant. M. H. J. BAYLIS 269

COGNITIVE PROCESSES: METHODS AND MODELS

18 The syntactic analysis of English by machine. J. P. THORNE, P. BRATLEY and H. DEWAR 281
19 The adaptive memorization of sequences. H. C. LONGUET-HIGGINS and A. ORTONY 311

PATTERN RECOGNITION

20 An application of Graph Theory in pattern recognition. C. J. HILDITCH 325

## PROBLEM-ORIENTED LANGUAGES

21 Some semantics for data structures. D. PARK 351
22 Writing search algorithms in functional form. R. M. BURSTALL 373
23 Assertions: programs written without specifying unnecessary order. J. M. FOSTER 387
24 The design philosophy of POP-2. R. J. POPPLESTONE 393

INDEX 403

# MACHINE INTELLIGENCE 4 (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

## 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