13

Experiments with a Pleasure-seeking Automaton

J. E. Doran
Department of Machine Intelligence and Perception
University of Edinburgh

INTRODUCTION

Attempts to write 'intelligent' computer programs have commonly involved the choice for attack of some particular aspect of intelligent behaviour, together with the choice of some relevant task, or range of tasks, which the program must perform. The emphasis is sometimes on the generality of the program's ability, sometimes on the importance of the particular task which it can perform. Well-known examples of such programs are Newell, Shaw, and Simon's General Problem Solver (1959; *see also* Ernst and Newell, 1967), which is applicable to a wide range of simple problems, Samuel's checker (draughts) playing program (1959, 1967), and the program written by Evans (1964), which solves geometric analogy problems.

However, there is another approach to the goal of machine intelligence which stresses the relationship of an organism to its environment and which sets out from the start to understand what is involved in this relationship.

Long ago Grey Walter (1953) experimented with mechanical 'tortoises' which could range over the floor in a lifelike manner. Toda (1962), in a whimsical and illuminating paper, has discussed the problems facing an automaton in a simple artificial environment. Friedman (1967), a psychologist, has described a computer simulation of instinctive behaviour involving an automaton equipped with sensory and motor systems. Andreae and Gaines (Andreae, 1964), in their STELLA project, have considered the design of an automaton faced with general control tasks. Sandewall (1967) has gone deeply into an automaton/environment relationship with a rather more formal approach. This list is far from complete. In particular, robots of various kinds are under construction at a number of research centres, notably at the Stanford Research Institute (Nilsson and Raphael, 1967).

The reader may find it helpful to meditate on the situation of, say, a rat in a cage, as seen by the rat. I hope the reader will agree that the animal perceives its rather simple environment in a variety of ways, that it remembers, learns, predicts, and acts towards goals which ultimately derive from basic necessities such as food, sleep, and the avoidance of pain. Can one write a computer program which, however primitively, simulates the rat and its surroundings and which demonstrates the simulated rat displaying such rudimentary intelligence as it is reasonable to require?

In attempting to write such programs I suggest that the ultimate objective is a classification of possible environments for the automaton (e.g. the simulated rat) *from the standpoint of the automaton*, together with an optimal automaton design in some non-trivial and useful sense for each class of environment. The approach described in this paper involves setting up a fairly 'natural', if simple, environment in the hope that the automaton design ultimately achieved by a combination of insight and trial and error experimentation will not only perform well in that environment, but will also display acceptable marks of intelligence.

THE AUTOMATON AND ITS ENVIRONMENT

I shall now try to specify a little more precisely the various concepts at issue, with the objective of constructing a helpful frame of reference. First of all I wish to distinguish three things:

(*a*) the running *computer program*, together with associated data, which mimics, however primitively, a portion of the real world. At this level the automaton is only arbitrarily distinguished from its environment;

(*b*) the 'objective' view of the automaton in its environment. This corresponds to looking at a rat in a cage from outside the cage, and implies a separate observer and viewpoint. I shall refer to the automaton's *objective environment*, that is, its surroundings as we see them;

(*c*) the environment as perceived by the automaton, corresponding to the rat's perception of its surroundings. This paper is primarily concerned with the automaton's efforts to understand and control this *subjective environment*. The reader must bear this last distinction in mind throughout.

The subjective environment involves a sequence of (subjective environment) *states*, each of which is the total possible perception by the automaton at any instant. There is a (subjective environment) *transition rule*, unknown to the automaton, which gives the next state in the sequence, given the history of the system including the automaton's *actions*. This rule will often be conveniently expressed in terms of the objective environment. Each of the actions available to the automaton may be applied at any time. They are distinguishable but otherwise unstructured. The following points are important.

1. 'time' means time as measured by a clock within the running program ((*a*) above);

2. the use of a sequence of states is a convenient approximation to one continuously varying state;
3. states will typically be very complex including (from the viewpoint (*b*) above) perceptions of the automaton's own internal functioning;
4. the actions will in general enable the automaton to achieve some measure of control over its future.

To motivate the automaton we associate with a state some idea of its *desirability*. Suppose that certain variables appearing in the state have bounds within which their values should lie. The desirability of a state then refers to the extent to which these constraints are met. The automaton must act so as to maximize the desirability of the states it encounters. If the desirability may be represented by an integer, then the automaton's motivation may be made more precise, if a little artificial, by assigning to it a fixed, known, lifetime and requiring it to maximize the mean desirability of its states over that lifetime. Notice that the desirability function cannot be varied by the automaton.

The automaton performs information processing operations needed to decide which actions to select and when. It is natural to suppose that processing proceeds at a *finite speed* (program time), and that information *storage capacity* is limited. These constraints form part of the design problem. The rate of processing achieved by the automaton relative to the rate of change of its subjective environment and to the lifetime available to it is very important.

The design of the automaton must take into account the anticipated properties of its subjective environment. The reader may find it illuminating to imagine himself (the automaton) before a screen on which is displayed a complex pattern which changes from time to time (sequence of states). He has access to a row of buttons (actions) any of which he can press at any time, and he has a given scale of preference for the patterns which occur. He has a limited supply of pencil and paper. His task is so to press buttons that over a given period of time the preference level is kept as high as possible. Naturally the reader's strategy will vary according to the information he is given relating button-pressing to the patterns displayed.

I shall now take as an example a particularly simple, if artificial, class of subjective environments which will serve to introduce some further concepts.

SUBJECTIVE ENVIRONMENT GRAPHS

Let the reader again imagine himself before the screen introduced in the last section, but now given the following additional information:

1. that no information is contained in any similarity between patterns shown. That is, patterns may only usefully be said to be identical or non-identical;
2. that the pattern displayed changes only when a button is pressed;
3. that the effect of pressing a particular button when a particular pattern is displayed is always the same, and that the effect is always immediate.

The reader may care to consider the strategy he would adopt.

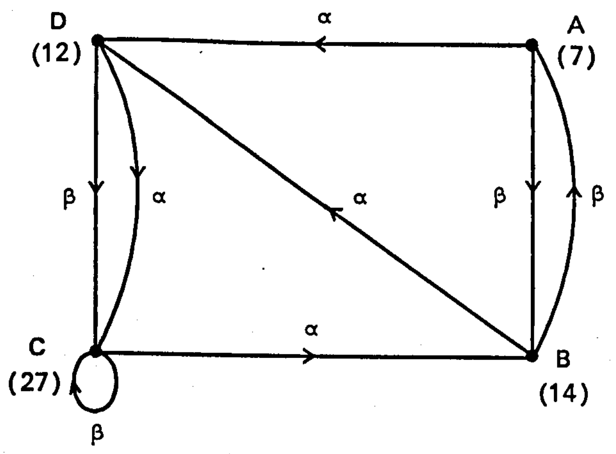


Figure 1. An environment graph–one way of specifying a simple subjective environment transition rule. The nodes correspond to states, the labelled arcs to action transitions, and the figures give the corresponding desirabilities

Figure 1 represents this situation in abstract form. The transitions of the subjective environment are described by a graph with 'labelled' arcs, the nodes of the graph representing states, and the arcs with a particular label representing the consequences of a particular action. Thus if the automaton is 'at' state D of figure 1, and applies the action α, then it immediately finds itself 'at' state C. The figures give the desirability of each state. How can the automaton be designed so that it performs well in any subjective environment of this type, without it ever knowing which particular graph it is faced with?

At any instant the automaton will be in some state, and we may assume that its history is accessible. Thus it might have stored that its history is as shown in figure 2 (*a*). If so we can now distinguish *three* relevant graphs. These are:

1. the *subjective environment graph* (figure 1),
2. the *stored graph* which is that portion of the subjective environment graph which the automaton has stored in its memory as a result of its experience (figure 2 (*b*)), and
3. the *option graph* which is that fragment of the stored graph which the automaton 'knows' how to reach (figure 2(*c*)).

The automaton has, I suggest, a broad choice between *exploration* and *exploitation*. Thus it can either decide to 'move' to some state in its option graph, and once there try out a new action, *or* it can select some suitably desirable state in its option graph, move to it, and do no more. In the first case the *goal* might be B and the *plan* $\beta\ \alpha\ \beta$, and in the second the goal might be C and the plan β^*. Which of these two types of plan the automaton should adopt will depend upon (1) the expected mean desirability to be achieved by

$$\begin{tikzcd}
\underset{(7)}{A} \arrow[r, "\alpha"] & \underset{(12)}{D} \arrow[r, "\beta"] & \underset{(27)}{C} \arrow[r, "\beta"] & \underset{(27)}{C} \arrow[r, "\alpha"] & \underset{(14)}{B} \arrow[r, "\alpha"] & \underset{(12)}{D} \arrow[r, "?"] & {}
\end{tikzcd}$$

(*a*) history

$$\begin{tikzcd}
\underset{(7)}{A} \arrow[r, "\alpha"] & \underset{(12)}{D} \arrow[r, "\beta"] & \underset{(27)}{C} \arrow[r, "\alpha"] \arrow[loop above, "\beta"] & \underset{(14)}{B} \arrow[ll, bend right=60, "\alpha"']
\end{tikzcd}$$

stored graph

(*b*)

$$\begin{tikzcd}
\underset{(12)}{D} \arrow[r, "\beta"] & \underset{(27)}{C} \arrow[r, "\alpha"] \arrow[loop above, "\beta"] & \underset{(14)}{B} \arrow[ll, bend right=60, "\alpha"']
\end{tikzcd}$$

option graph.

(*c*)

Figure 2. A transition history based on the graph of figure 1, together with the corresponding stored and option graphs (see text)

exploration at the best exploration goal, and (2) the desirability of the best exploitation goal.

The reader should note that I am using the word 'plan' to refer to a situation where a course of action is determined upon in advance, and then carried through without further 'thought'. For this class of subjective environments at least, there is clearly no point in reconsidering a plan part way through its implementation.

An important complication is introduced by the time that the automaton must take to select a plan, which has the consequence that the selection process will involve a succession of steadily more promising plans (of either type) until the time cost of further improvement tilts the balance in favour of actual implementation of the current best plan. The time cost of decision making is more pressing the lower the desirability of the current state.

After an exploratory action the automaton may either be forced to explore again, since it has encountered a quite new state, or it may find itself at a state it recognizes when there will be a non-trivial option graph and therefore an opportunity for further plan formation.

In practice, heuristics will be needed at many points in the automaton design. The difficulty of finding optimal decision strategies taking into account the time and storage constraints is far too great for precise solution. Whenever there is a choice between a small benefit immediately and a possibly large benefit in the future, then a plausible heuristic will probably be needed.

The difficulties which arise as soon as more interesting subjective environments are considered are best introduced in a more concrete context. I shall therefore now describe the present computer program which involves a subjective environment with complex states, and with a more complicated transition rule. For a more extensive general discussion the reader is referred to Doran (1967, 1967a).

THE POP-2 PROGRAM: OBJECTIVE AND SUBJECTIVE ENVIRONMENTS

The program now to be described is written in the list processing language POP-2 (see the paper by R.J.Popplestone in this volume) which has been developed at the Department of Machine Intelligence and Perception of the University of Edinburgh. The language is implemented on an Elliott 4100 computing system and is oriented towards machine intelligence work.

The reader may find his understanding of this program and its behaviour aided if he keeps in mind the following analogy. A small boy lives at the busy centre of a large city. One day he is taken to a quiet suburb where he has never been before, and left to find his own way home. We suppose that he is too shy to ask someone the way and that he does not think of buying a map. He therefore starts walking, always preferring streets or districts where there is traffic and bustle, the more the better, since he remembers that he lives in a very busy place. Sometimes he realizes that he has walked in a circle, and then he sets off in some new direction from the busiest point he remembers how to reach. Ultimately, we suppose, he arrives home after a very long walk. Suppose that he is now taken back to the remote suburb and left there a second time. Now he should return home in less time and by a much more direct route, for he has the memory of his previous trek to guide him and therefore, for example, can plan ahead to some extent.

This small boy is very much in the situation of the simulated automaton now to be introduced. The automaton's surroundings, which I shall describe first, correspond to the city in the foregoing analogy.

The objective environment or 'enclosure' provided for the automaton consists of a square area (10 units × 10 units) with boundary and interior walls. Figure 3, which is an example of program output, shows this enclosure. The walls are not uniform but are 'formed' of letters of the alphabet. The automaton is represented by an asterisk and has location, and orientation to the top, left, right, or bottom of the picture. The automaton's co-ordinates are always integral. Note that 'objective' output is preceded by a double set of asterisks, 'subjective' by a single set.

The actions available to the automaton are the following:

(*a*) STEP–to move forwards into the next unit square.

(*b*) LEFT–to turn through a right angle to its left.

```
*** SAMPLE IS [ C 1 STAND 4Ø 1138 ]
*** I KNOW THIS
*** I HAVE A PLAN
*** STEP
*** *** TIME 1378 X 4 Y 2 FACING TOP

CCCCCCCCCC
E  *     B
E    DDD B
E    HHD B
E IJ HHHAG
E IJ HHHHG
E IJ     G
E IJ     G
E        G
FFFFFFFFFF


*** SAMPLE IS [ C Ø STEP 41 1379 ]
*** FORSEEN
*** RIGHT
*** *** TIME 1398 X 4 Y 2 FACING RIGHT

CCCCCCCCCC
E  *     B
E    DDD B
E    HHD B
E IJ HHHAG
E IJ HHHHG
E IJ     G
E IJ     G
E        G
FFFFFFFFFF


*** SAMPLE IS [ B 5 RIGHT 39 1399 ]
*** FORSEEN
*** EXPLORE
*** STEP
*** *** TIME 1425 X 5 Y 2 FACING RIGHT

CCCCCCCCCC
E  *     B
E    DDD B
E    HHD B
E IJ HHHAG
E IJ HHHHG
E IJ     G
E IJ     G
E        G
FFFFFFFFFF
```

Figure 3. Sample output from the automaton/environment program showing the objective environment, typical subjective state vectors, and the automaton's 'chatty' remarks

(*c*) RIGHT–to turn through a right angle to its right.

(*d*) STAND–to remain still.

If STEP is impossible since the automaton is against a wall, then STEP has the effect of STAND. Selecting the action STAND must not be confused with selecting no action at all.

Note that the automaton knows nothing of the effect of these actions except what it learns through experience.

The automaton's subjective environment state is a 5-vector of the following form:

[⟨wall⟩, ⟨distance⟩, ⟨last action⟩, ⟨desirability⟩, ⟨time⟩]

Examples of state vectors appear in figure 3 after the words 'SAMPLE IS'. The first shown is

[C 1 STAND 40 1138]

and this implies that the automaton is facing the C wall, that there is one empty square between it and the wall, that its last action was STAND, that the desirability of the current state is 40, and that the system time is 1138.

Note that the automaton's view of its surroundings is very restricted. It can see only which 'letter' is in front of it, and how far away it is. Thus the subjective environment is very different from the objective environment. Let us call the first two elements of the state vector the *reduced state* (C1 in the above example). This reduced state is the unit used by the automaton in its processing. Now a given reduced state does *not* uniquely specify the automaton's true location, and this turns out to be both a source of confusion and of assistance to the automaton. Confusion occurs because the subjective environment may not react in the future as it has in the past, assistance because the automaton may react correctly in quite new situations because from its point of view they are *not* new but are identical to previously experienced situations.

Two further points need comment. First, the appearance of the automaton's last action in the state vector means that the automaton remembers past actions in much the same way as it remembers other information. Second, the desirability is included as a direct perception in this program for simplicity. It is calculated as a simple numerical function of the reduced state, namely:

50–(DISTANCE+3*WALL)

where WALL is 1 in the case of A, 2 in the case of B, and so on. The desirability is maximized when the automaton is facing and against the letter A, and we may then say that the automaton is in its 'nest'. There are local maxima elsewhere.

The automaton is typically placed at some point in its enclosure and expected to find its way to the nest and remain there. If moved out of the nest it should quickly return. More generally, it should always behave 'sensibly'.

Simple program facilities are provided which permit the automaton to be moved about, guided, and reset to some point in its history by an 'on-line' experimenter.

THE POP-2 PROGRAM: ORGANISATION OF THE AUTOMATON

Figure 4 shows a skeleton flow chart for the automaton. The cycle is as follows. The automaton first reads a state vector (SAMPLE), and stores in its memory

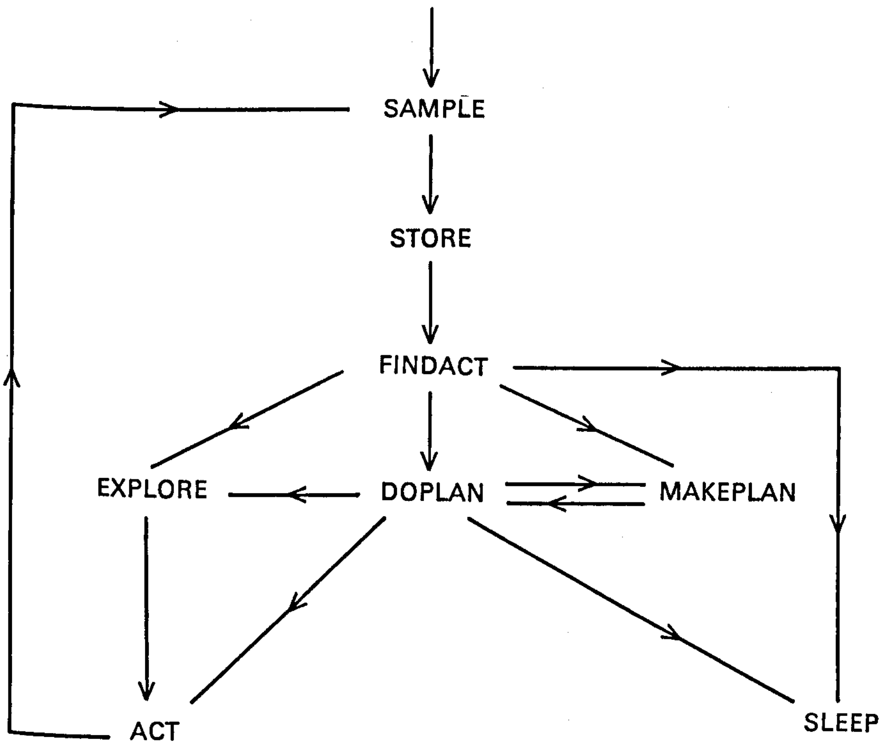


Figure 4. Outline flow-chart of the automaton

the last state vector transition (STORE). It then enters FINDACT and decides which of the following alternatives is appropriate:

EXPLORE–select randomly an action never before tried at this reduced state.

MAKEPLAN–carry out a lookahead and form a plan.

DOPLAN–select the next action of the current plan or go to EXPLORE MAKEPLAN or SLEEP if this is indicated by the current plan.

SLEEP–stop.

In ACT the automaton actually implements an action, and the state vector and objective environment change appropriately before the automaton again goes to SAMPLE.

Sufficient has been said about the action of SAMPLE in the previous section. STORE will now be explained by reference to figure 5, which shows the arrangement of the automaton's 'memory' as a sort tree. The sequence of transitions shown at the top of the figure is represented as the tree structure shown. In general suppose that the automaton has observed a transition from state vector X to state vector Y. It stores this as follows:

(*a*) the tree node corresponding to the reduced state of X is located by starting at the top and branching appropriately;

(*b*) relevant information kept at this node, for example, the desirability of the reduced state and the time of the last encounter is updated (not shown in figure);

(*c*) the appropriate terminal node is located (by reference to item 3 of Y) and a pair containing the time of X and the reduced state of Y (a *consequence*) is added to the list of observed consequences already there.

Branches are created if they do not already exist.

The length of the list of observed consequences kept at each terminal node is limited by the program parameter FORGETP. If a new addition would cause the length of the list to exceed the limit set, then the oldest consequence is deleted to make room.

This storage system has the following properties:

1. the detailed history of the automaton can be recovered (with a little difficulty) provided that no information has been erased in the manner just indicated, and
2. the consequences in the past of applying a particular action to a particular reduced state are very easily retrieved.

The function of the *plan vectors* shown in the figure will be explained in the next section.

EXPLORATION AND PLAN FORMATION

In FINDACT the automaton must choose between the options EXPLORE, DOPLAN, MAKEPLAN, and SLEEP. To do this the automaton first refers to its memory to establish if it has ever before encountered the current reduced state. If not, and if the desirability of the reduced state is at the *target value*, then SLEEP is chosen, otherwise EXPLORE. The general significance of this target value will be explained below.

If the reduced state is recognized then the automaton inspects the corresponding plan vector (see figure 5) to establish whether it has already decided what to do in this situation. If so then it selects DOPLAN. Note that the stored plan instruction implemented in DOPLAN may cause entry to EXPLORE, MAKEPLAN, or SLEEP, rather than merely indicating a basic action.

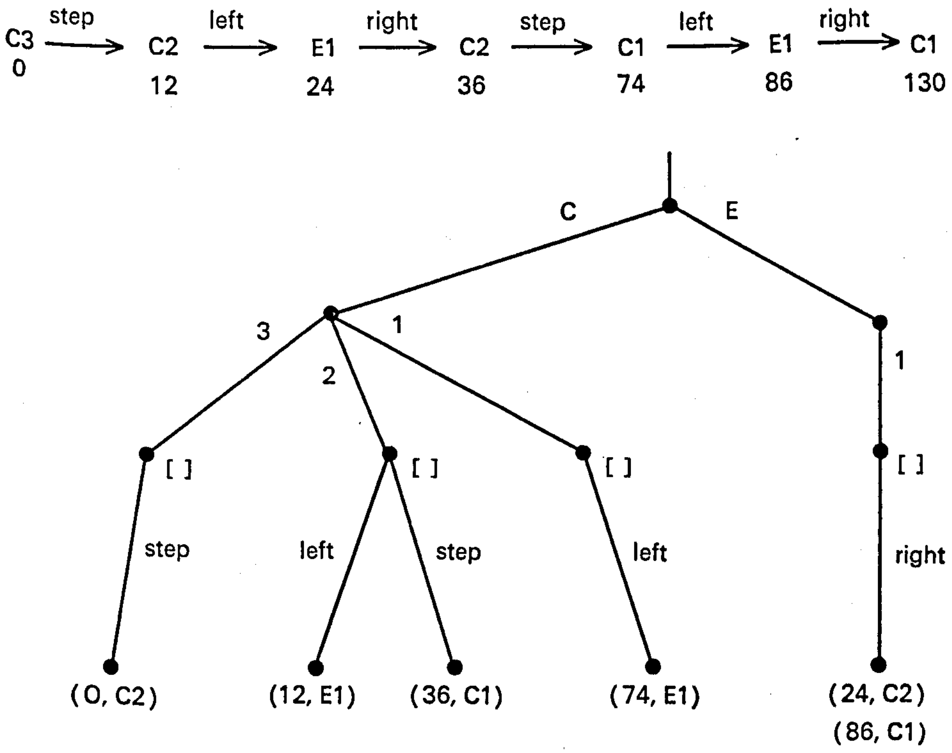


Figure 5. The memory tree. The transition history at the top of the figure is stored as the tree shown below. [ ] indicates location of plan vector. For details see text

MAKEPLAN is entered if the current reduced state is recognized, but not anticipated as part of a plan. Its main function is to grow a 'lookahead tree' of a type analogous to that used in many game-playing programs, and to select a plan. Figure 6(a) shows this tree with unimportant details omitted.

The tree is grown (downwards) by reference to the automaton's memory, and corresponds to the option graph earlier defined. Each node in the figure represents a reduced state. Each branch labelled with a Greek letter corresponds to the selection of a particular action. The unlabelled branches represent the observed consequences of applying given actions to given reduced states. Thus in the figure the root node corresponds to the automaton's current reduced state. The actions α and β have been tried previously in this state. α has been applied twice with differing consequences. β has been applied

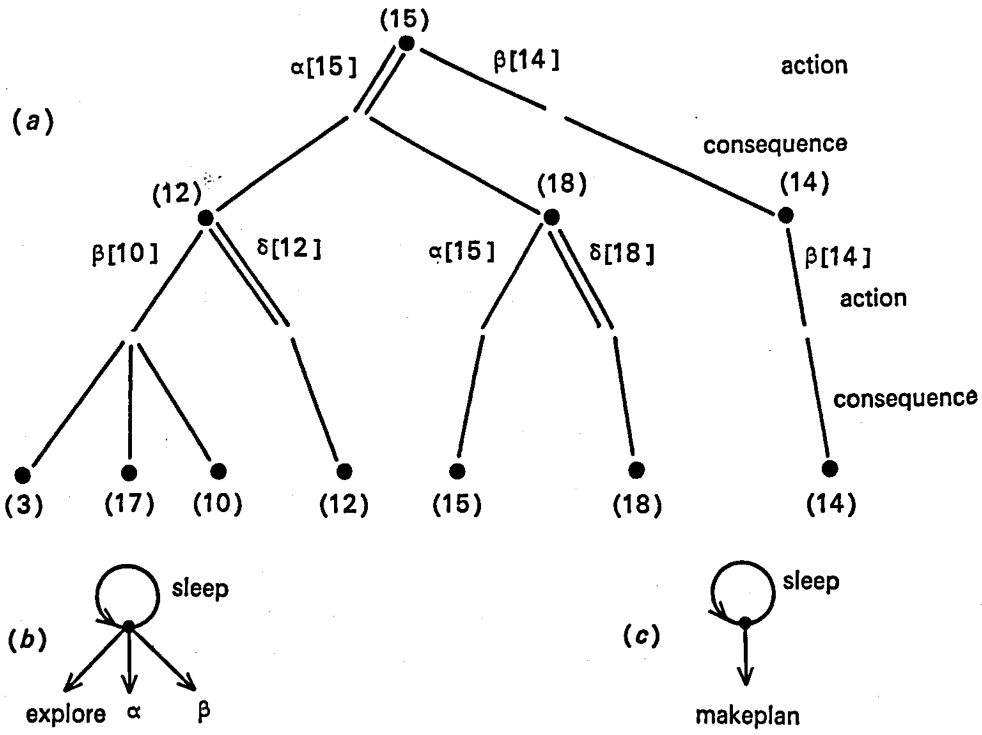


Figure 6. The lookahead mechanism. The tree (*a*) is grown by reference to the memory tree of figure 5, and values are assigned to the terminal states and 'backed-up' as indicated. Diagrams (*b*) and (*c*) show in greater detail non-terminal and terminal nodes respectively. For full details see text

once. The first consequent reduced state of the application of α has itself had β and δ applied to it, and so on. This lookahead tree is taken out to a fixed 'depth' determined by the program parameter DEPTHP. Having 'called to mind' the relevant information, the automaton now selects an action for its current reduced state, and for all of the consequences that that action may have, and so on out to the depth of the tree. That is, the automaton selects a plan. In more detail, it uses the following process which is a crude form of the 'expectimaxing' process described by Michie and Chambers (1968a).

(*a*) Values are assigned to the reduced states forming the branch tips. These values are loosely to be interpreted as the future mean desirability level to be expected if the automaton actually reaches the corresponding reduced state, and continues planning from there.

(*b*) The reduced states one step back from the terminal states now have values assigned to them. Each action tried is considered and a value associated with it which is the mean of the values of its observed consequences. The greatest such action value is then assigned to the reduced state, and the corresponding action is adopted as the plan action for that reduced state.

(*c*) This process is carried up the tree until the root state is given a value.

In figure 6(*a*) a consistent set of values has been indicated (square brackets for action values and round brackets for state values) and the plan actions are shown by double lines. The most important information kept in the plan vector of a reduced state is the plan value together with the corresponding plan action.

The process is rather more complicated than has been described, for at each non-terminal state of the lookahead tree the automaton has additional SLEEP and EXPLORE options. The true situation is indicated in figure 6 (*b*). At terminal states the automaton has a choice between the MAKEPLAN option–corresponding to the assigned value already mentioned–and SLEEP (figure 6 (*c*)). Each option is assigned a value, the greatest value is selected and passed up the tree, and the corresponding choice (say from α, β, EXPLORE, SLEEP) is stored away in the plan vector for use if the corresponding reduced state should actually be encountered.

How are these various values estimated? This involves a target value which is a desirability level the automaton is 'told' it can achieve but not exceed (compare the 'level of aspiration' used by the BOXES program (Michie and Chambers, 1968a)).

SLEEP–the corresponding value is simply the desirability of the reduced state at which sleep is proposed.

EXPLORE–the value assigned lies between the corresponding SLEEP value and the target value, according to the expression

$$S+(TV-S)*\mathrm{EXPVALP}$$

where S is the SLEEP value, TV the target value and EXPVALP is a program parameter lying between 0 and 1.

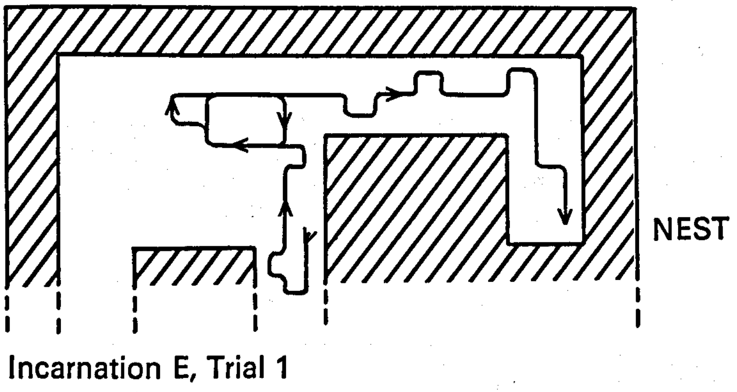
MAKEPLAN (i.e. tip value)–this is calculated in a manner akin to that used for EXPLORE, but is made rather larger following the argument that planning can be expected to give better performance than random exploration. This increment is made a decreasing function of the number of times the corresponding reduced state has been encountered using the parameter TRANSITP.

This outline description of the functioning of the lookahead and planning mechanism has avoided a number of complexities which, although trying in practice, have little general significance. For example, branches of the lookahead tree often coalesce or loop with consequent difficulties.

An important question concerns the situation once a plan has been used or has gone awry (the latter is quite possible–an action may have a consequence never before observed). Should any part of the information gathered during the plan's formation be used again? More concretely, should the plan vectors be erased or kept for future use? Several possible answers to this question will be considered in the next section. It will suffice here to state that the basic version of the program effectively deletes the plan vectors once the corresponding plan has been used.

EXPERIMENTAL FINDINGS

As already stated, figure 3 shows a fragment of output from the program. The 'chatty' remarks made by the automaton serve to indicate the type of processing it is currently engaged upon. They are not part of the automaton



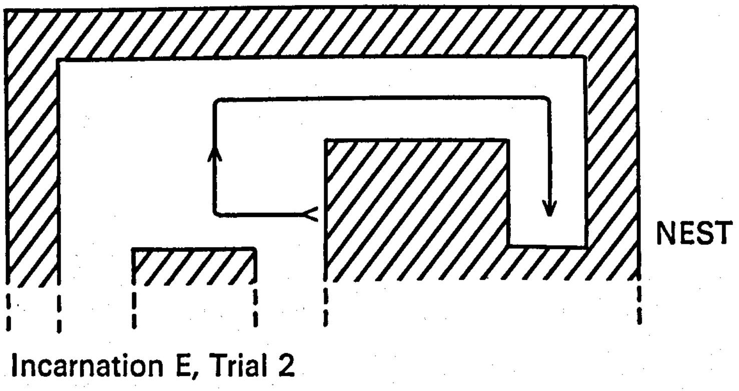


Figure 7. The first route the automaton finds to its nest will typically be very circuitous. Starting from the same point the second route will be far better but not necessarily optimal

		TRIALS										
		1	2	3	4	5	6	7	8	9	10	*totals*
INCARNATIONS	A	449	179	322	344	265	203	211	1625	288	427	4313
		43	*14*	*22*	*18*	*19*	*18*	*17*	*50*	*16*	*20*	*237*
	B	528	129	122	111	143	143	143	143	143	144	1749
		41	*12*	*12*	*12*	*12*	*12*	*12*	*12*	*12*	*12*	*149*
	C	722	194	295	257	451	245	297	394	227	269	3351
		66	*18*	*21*	*22*	*28*	*16*	*17*	*21*	*14*	*16*	*239*
	D	3436	525	157	486	265	257	378	259	321	452	6536
		191	*18*	*12*	*16*	*13*	*14*	*14*	*13*	*14*	*14*	*319*
	E	858	193	374	417	426	303*	623	535*	688*	2275*	6692
		58	*14*	*19*	*18*	*19*	*13*	*23*	*18*	*20*	*52*	*254*
	F	589	183	270	215	249	179	200	200	222	189	2496
		46	*14*	*16*	*14*	*14*	*12*	*12*	*12*	*13*	*12*	*165*
means		1097	234	257	305	300	222	309	526	315	626	
		74	*15*	*17*	*17*	*18*	*14*	*16*	*21*	*15*	*21*	

Table 1. Sample results showing the automaton's performance in the environment of figure 3. Each incarnation involved a blank initial memory and ten successive trials from the starting point of figure 7. The upper figure of each entry is time taken to reach the nest and the lower figure is the number of actions used. Asterisks indicate unsuccessful trials. The minimum possible number of actions required was 12.

design. As a result of experimentation with the program the following remarks may be made.

The automaton successfully uses its record of its past explorations to form and implement plans. These plans enable it to find its way to its nest by a much more direct route on the second or subsequent trials than that followed on the first trial. The planned route need not be optimal, however. Figure 7 shows a typical example of this improvement. Note that the automaton does much more than merely retrace its steps, and that the improved route involves an objective location not previously visited by the automaton. Prior to trial 1, the automaton's memory was quite blank. Table 1 presents some sample detailed results. In each of six 'incarnations' the automaton was placed at the starting point of figure 7 with a blank memory, and left to find its way to the nest. It was then replaced nine times, but allowed to cumulate its memory. The upper figure of each entry in the table is the time taken to reach the nest in tens of basic units, and the lower figure is the number of actions used. The following points are worthy of note:

(*a*) the incarnations are not all identical since, particularly in the initial stages, the automaton sometimes engages in random exploration;
(*b*) performance on trial 2 is uniformly much better than on trial 1;
(*c*) the automaton's behaviour does not necessarily become stereotyped (see TRANSITP below). During trial A8, for example, the automaton tries an exploration action and gets thoroughly 'lost' in consequence;
(*d*) the asterisks indicate trials when the automaton 'gave up' just before reaching the nest (for fairly sensible reasons).

Figure 8 shows trials, A, B, C of Table 1 plotted as graphs.

The behaviour of the automaton depends heavily upon the values chosen for the parameters mentioned previously.

FORGETP–this fixes the maximum number of consequences held at each branch tip of the memory tree. A value of 6 rather than 3 for this parameter increased the automaton's thinking time significantly. The automaton was also slower to escape from the fairly common type of situation in which it is misled by events recalled from the past which are not in fact relevant, and in consequence loops repeatedly (*see* below).

EXPVALP–this is involved in value setting as indicated above. If it is set too low (0·2 rather than 0·6) then, in the type of environment described, the automaton will tend to 'sleep' at points of locally high desirability.

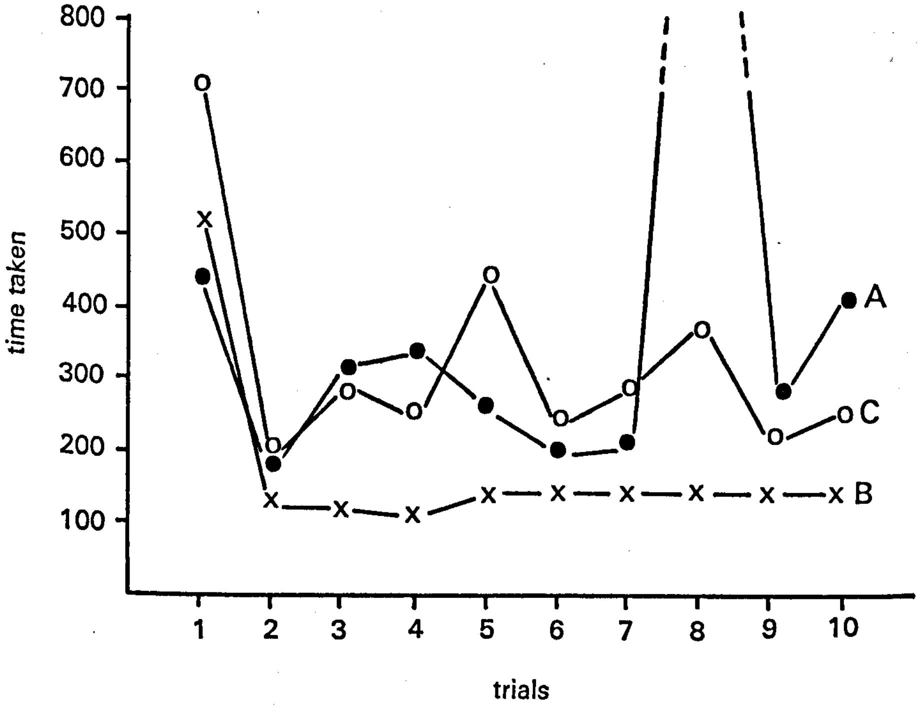
DEPTHP–taking the lookahead to a fixed depth of 7 rather than 3 made the automaton less likely to be trapped in locally desirable areas, but markedly increased the thinking time.

TRANSITP–this helps determine the relative value which the automaton assigns to planning rather than exploration. Suitably set it causes the automaton to vary from time to time an apparently fixed route.

The point has already been made that the automaton has a deliberately limited view of its surroundings. This causes it to generalize a response judged best in one situation to all situations which have the same reduced state. In the type of objective environment used here, the automaton will sometimes benefit from this built-in tendency to generalization, and will sometimes repeatedly go astray. In passing we may note that the automaton is assuming a truly stochastic transition rule, when this is not the case in reality.

Finally, consider again the question raised at the end of the last section. The basic version of the program makes no further use of plans once they have been fully or partially implemented. However, two variants of this program have been tested, the first of which (Variant A) uses an old plan value attached to a reduced state as a source of information when calculating a makeplan value, and the second of which (Variant B) uses an old plan vector as if it had been calculated as part of the current lookahead. Both variants are generalizations of the basic program in that they still reject plan information of over a fixed age.

Simple experiments with these programs suggest that Variant A is superior to the basic program in situations where a particularly deep lookahead is



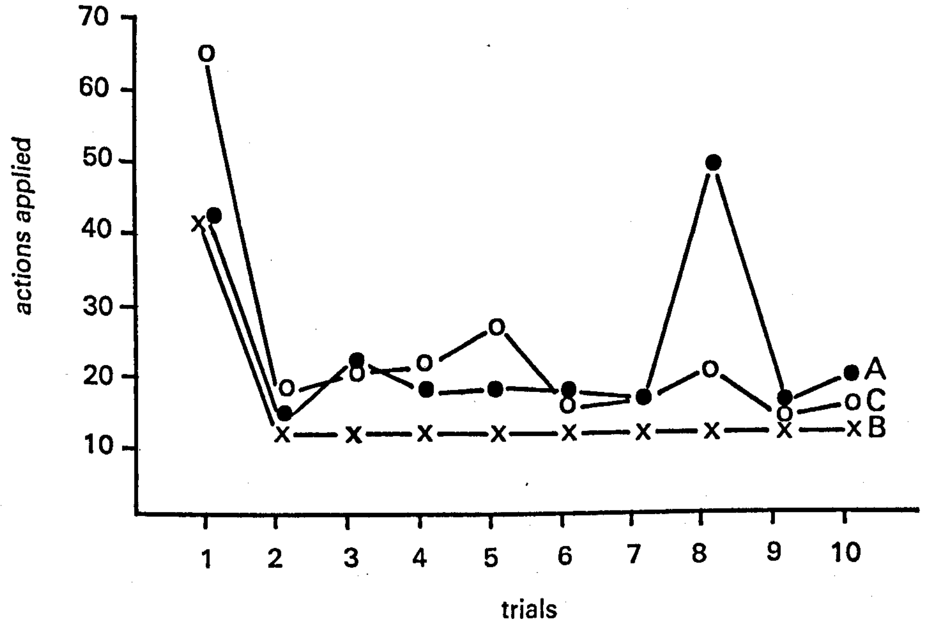


Figure 8. Incarnations A, B, and C of Table 1 plotted as graphs

essential. It is using a rote learning technique analogous to that of Samuel (1959) to increase effective lookahead depth. Variant B has the effect that when a planned route has once been followed, it is thereafter always followed 'without further thought' until the plans defining the route become too old. The automaton then forms fresh plans. Following a route 'without further thought' implies much more rapid movement, but no chance at all of possibly beneficial variation.

MINOR LINES OF DEVELOPMENT

It will be clear to the reader upon reflection, if not before, that the program that I have described is no more than a first attempt at a very large and complex objective. Even without considering any major modification to the automation design, there are the following directions in which improvements to it could certainly be made.

(*a*) The *lookahead tree* used in forming plans is organized in a very arbitrary way. There are surely better ways for the automaton to decide whether or not to continue consideration of some part of the option graph than for it to use a fixed depth rule. For example, the work on the Graph Traverser program (Doran and Michie, 1966) immediately indicates the use of a state evaluation function of some kind to direct the growth of the tree. This could merely use the desirability, but more complex mechanisms can easily be imagined.

A complementary approach is to consider the probability that a given branch will ever be reached. In the extreme case when this probability is zero, further growth of the lookahead tree from that branch is entirely wasteful. Some probabilistic generalization of the '$\alpha-\beta$ heuristic' is perhaps required here (Edwards and Hart, 1963; *see also* Samuel, 1967).

(*b*) Once the lookahead tree has been grown there comes the problem of selecting a plan. The current simple method of selecting actions is open to the criticism that one 'unlucky' consequence of an action in a given situation can damn it for ever. This is essentially the 'Two-Armed Bandit' problem, and the reader is referred elsewhere for a discussion of its significance (e.g. Jacobs, 1967; Michie, 1966).

(*c*) The repeated use of plan information has already been discussed above. There are certainly more possibilities than those which have been actually tried.

(*d*) The present method of deleting information from the memory is very simple. More complex methods could be tried and could be chosen to use the expected properties of the subjective environment.

These various improvements to the design of the automaton, though of considerable interest, would not in themselves add up to any major step forward. Nor would this be achieved by a study of the automaton's performance in a variety of environments. I propose now to introduce some ideas which perhaps dig a little deeper.

MAJOR LINES OF DEVELOPMENT

The present automaton can ultimately learn the truth of such statements as the following (figure 3):

> If I am facing the G wall and two steps from it, and if I turn right, then I shall be facing the F wall and against it, or facing the F wall and one step from it, or facing the F wall and two steps from it, with (pseudo-) probabilities P_1, P_2, P_3, respectively.

It cannot, however, learn the truth of either of the two following statements:

> (A) If I am facing the G wall and I turn right, then I shall be facing the F wall.
>
> (B) The action STAND never has any effect.

If the automaton is to learn such facts as this it must be able to generate them in some representation. (A) might be handled as follows. Consider again the sort tree shown in figure 5, and find the node corresponding to the reduced state C3. Both the plan information at that node, and the recorded consequence of the action STEP, can be regarded as statements about C3. We would like the automaton to record and use broadly similar statements about C, more generally, about all nodes higher in the tree. A method of achieving this is to enable the automaton to seek statements which are true for all nodes on branches out of some node and to associate them with that higher-level node, (compare the proposed 'lumping' of boxes by the BOXES program (Michie and Chambers, 1968)). Plan formation would then involve a more complex 'lookahead' tree which would be grown using the *simplest* statements which would give satisfactory prediction.

This latter step leads us to the realization that the automaton need not always sample the full state vector–it need only note that part of the vector needed for its current planning.

To cope with statement (B)

> The action STAND never has any effect.

a further mechanism seems necessary. Each node of the sort tree can be regarded as a test. Suppose that we allow these tests to be not merely a matter of looking at the next element in the current state vector, but general tests on both the current state vector and on immediately preceding ones (compare the EPAM program of Feigenbaum, 1961,). Statement (B) can now perhaps also be handled for it is saying that the action STAND always leads to a situation which is (effectively) identical with that preceding it, and we have now allowed the automaton to test for this. Of course there is now the problem of how the automaton should decide *which* tests to apply and in which order to apply them; but at least the door seems open.

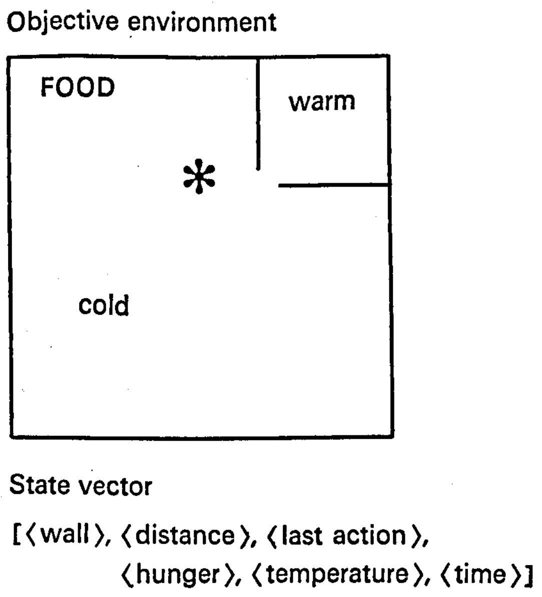

Objective environment
FOOD
warm
*
cold
State vector
[⟨wall⟩, ⟨distance⟩, ⟨last action⟩,
⟨hunger⟩, ⟨temperature⟩, ⟨time⟩]

Figure 9. An environmental situation requiring major improvements in the design of the automaton

Consider finally figure 9, which shows a more complex environment and state vector. The automaton is provided with a 'nest' which is 'warm' but which it must leave to obtain 'food' which is kept outside in the 'cold'. The state vector is augmented by elements indicating the degree of hunger and the temperature, and the explicit perception of desirability is omitted. A new action, EAT, is provided.

The kind of behaviour we require of the automaton is that it should be able to find its way to the nest, and that it should only leave the nest when it is sufficiently hungry to brave the cold, returning to the nest immediately it has eaten. This 'sensible' behaviour is to be learned from an initial naïve state, perhaps with some simple 'tuition'.

The automaton must take into account two desirability factors, namely the hunger and the temperature, and must use the remainder of the state vector to help it control these factors. There is a conflict between these two factors, and more important hunger is essentially discontinuous even for optimal automaton behaviour, for it changes sharply with the single action EAT.

Consider the following sequence of events which might overcome these difficulties:

(*a*) Initially the automaton is in the nest, is warm, and is not hungry. It therefore sleeps.

(*b*) The automaton wakes because it becomes hungry. This implies a basic scan of the state vector while asleep.

(*c*) The automaton notices that it is hungry (rather than cold) and recalls its subjective state when it last ate. This state it adopts as a *goal.* This is a new *ad hoc* mechanism which can be regarded as a first step towards 'means end analysis' in the General Problem Solver program sense, and ultimately towards the complex type of planning discussed in Miller, Galanter, and Pribam (1960).

(*d*) The automaton then seeks a plan directed towards achieving its goal. Both the growth of the lookahead tree and the choice of plan can be guided by an evaluation function using the features of the automaton's goal. Note that the feature of the goal state that it is cold can be used to help guide the search.

(*e*) Having formed and implemented one or more plans the automaton will ultimately reach its goal. At this point the 'food' goal is cancelled, to be replaced at once by a directly analogous 'warmth' goal.

(*f*) Plans are made and implemented for the new goal. It is ultimately achieved and the automaton is returned to (*a*) above.

The automaton would *learn* to carry out the above operations in the sense that it would gather for itself the experience needed to form the plans involved. The goal setting and planning capacity would be inbuilt. 'Tuition' might prove necessary in the initial stages and could take the form of forcing the automaton to choose certain actions and thus perceive the consequences of them.

CONCLUDING REMARKS

In this paper I have described a program which simulates a heuristic automaton in a very primitive but natural environment. The environment is natural in the sense that its properties are analogous to those of the world around us, but primitive in the sense that those properties are vastly simplified.

The reasoning behind this program argues that intelligence is very much a response to our own everyday environment and that to understand its nature it is desirable, perhaps essential, to study the properties of that environment. By environment I must emphasize that I mean *subjective* environment. Thus we have been primarily concerned with the automaton's world as the automaton sees it, not as we the outsiders see it. Both this stress on the subjective nature of the automaton's problems, and the fact that I have proposed no formal representation of the information gathered and processed by the automaton are open to dispute. What does seem clear, however, is that the only way to make concrete progress is to write computer programs and to see how they perform.

Acknowledgments

The research described in this paper is sponsored by the Science Research Council under the general supervision of Professor Donald Michie. Many people have made helpful suggestions and criticisms concerning this work, but I am particularly in debt to my colleague Dr R. M. Burstall.

REFERENCES

Andreae, J.H. (1964), STELLA: a scheme for a learning machine. *Proceedings of second IFAC congress*, pp. 497–502. London: Butterworth.

Doran, J.E. and Michie, D. (1966), Experiments with the Graph Traverser program. *Proc. R. Soc.* A, **294**, 235–59.

Doran, J.E. (1967), Criteria for intelligence in a programmed automaton. *Memorandum MIP-R-26*. University of Edinburgh: Department of Machine Intelligence and Perception.

Doran, J.E. (1967a), Designing a pleasure-seeking automaton. *Memorandum MIP-R-28*. University of Edinburgh: Department of Machine Intelligence and Perception

Edwards, D.J. & Hart, T.P. (1963), The α-β heuristic. *Artificial intelligence project memorandum No. 30* (revised). RLE and MIT computation centre.

Ernst, G. & Newell, A. (1967), Some issues of representation in a general problem solver. *AFIPS*, **30**, 583–600. Spring J.C.C. Washington: Spartan Books.

Evans, T.G. (1964), A heuristic program to solve geometric-analogy problems. *AFIPS*, **25**, 327–38. Spring J.C.C. Baltimore: Spartan Books.

Feigenbaum, E.A. (1961), The simulation of verbal learning behaviour. *Proceedings of the Western J.C.C.*, **19**, 121–32. Baltimore: Spartan Books.

Friedman, L. (1967), Instinctive behaviour and its computer synthesis. *Behavl Sci.*, **12**, 85–108.

Jacobs, O.L.R. (1967), *Introduction to Dynamic Programming*, pp. 114–19. London: Chapman and Hall.

Michie, D. (1966), Game-playing and game-learning automata. *Advances in programming and non-numerical computation*, pp. 183–200 (ed. Fox, L.). Oxford: Pergamon Press.

Michie, D. & Chambers, R. A. (1968), 'Boxes' as a model of pattern-formation. *Towards a Theoretical Biology 1*, pp. 206–15 (ed. Waddington, C. H.). Edinburgh: Edinburgh University Press.

Michie, D. & Chambers, R.A. (1968a), BOXES: an experiment in adaptive control. *Machine Intelligence, 2* (eds Dale, E. and Michie, D.). Edinburgh: Oliver and Boyd.

Miller, G.A., Galanter, E. & Pribam, K.H. (1960), *Plans and the structure of behaviour*. New York: Holt.

Newell, A., Shaw, J.C. & Simon, H.A. (1959), Report on a general problem-solving program. *Proceedings of the international conference on information processing*, pp. 256–64. Paris: UNESCO.

Nilsson, N. J. & Raphael, B. (1967), Preliminary design of an intelligent robot. *Computer and information sciences 2*, pp. 235–59 (ed. Tou, J. T.). New York: Academic Press.

Samuel, A.L. (1959), Some studies in machine learning using the game of checkers. *IBM Jl Res. Dev.*, **3**, 211–29.

Samuel, A.L. (1967), Some studies in machine learning using the game of checkers II–recent progress. *Stanford artificial intelligence project memorandum*, No. 52.

Sandewall, E. (1967), An environment for an artificial intelligence. *Report No. 10*. University of Uppsala: Department of Computer Sciences.

Toda, M. (1962), The design of fungus-eater: a model of human behaviour in an unsophisticated environment. *Behavl Sci.*, **7**, 164–83.

Walter, W.G. (1953), *The living brain*. New York: Norton.