MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

Artificial Intelligence
Memo No. 195 April 1970

INSIM1:

A Computer Model of Simple Forms of Learning

Thomas L. Jones

INSIM1 is a computer program, written in LISP, which models simple forms of learning analogous to the learning of a human infant during the first few weeks of his life, such as learning to suck the thumb and learning to perform elementary hand-eye coordination.

The program operates by discovering cause-effect relationships and arranging them in a goal tree. For example, if A causes B, and the program wants B, it will set up A as a subgoal, working backward along the chain of causation until it reaches a subgoal which can be reached directly; i.e. a muscle pull.

Various stages of the simulated infant's learning are described.

INSIM1: A Computer Model of Simple Forms of Learning

I. Introduction and Summary

INSIM1 is a computer program, written in LISP, which models simple forms of learning analogous to the learning of a human infant during the first few weeks of his life, such as learning to suck the thumb and learning to perform elementary hand-eye coordination.

The program operates by discovering cause-effect relationships and arranging them in a goal tree. For example, if A causes B, and the program wants B, it will set up A as a subgoal, working backward along the chain of causation until it reaches a subgoal which can be reached directly, i.e. a muscle pull.

A typical problem is the one-dimensional, three-point thumb-sucking problem, which can be described in logical notation as follows:

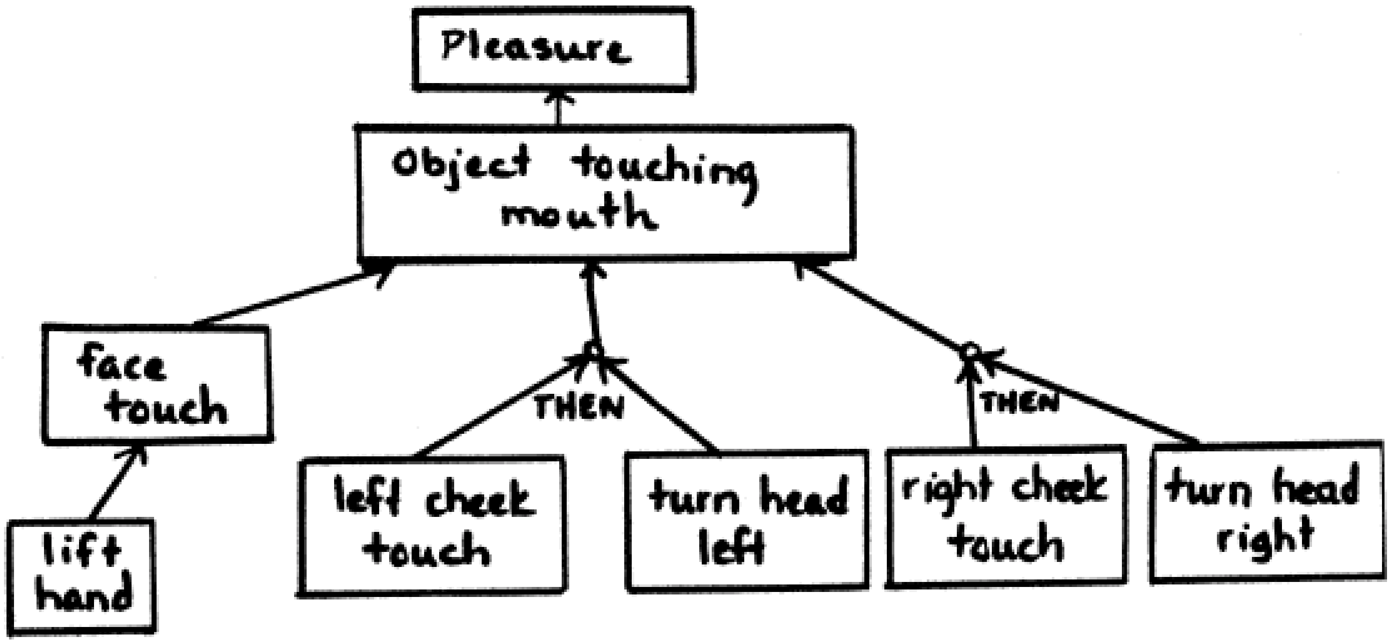
(1) object touching mouth $\rightarrow$ pleasure

(2) (left check touch $\wedge$ turn head left) $\rightarrow$ mouth touch

(3) (right cheek touch $\wedge$ turn head right) $\rightarrow$ mouth touch

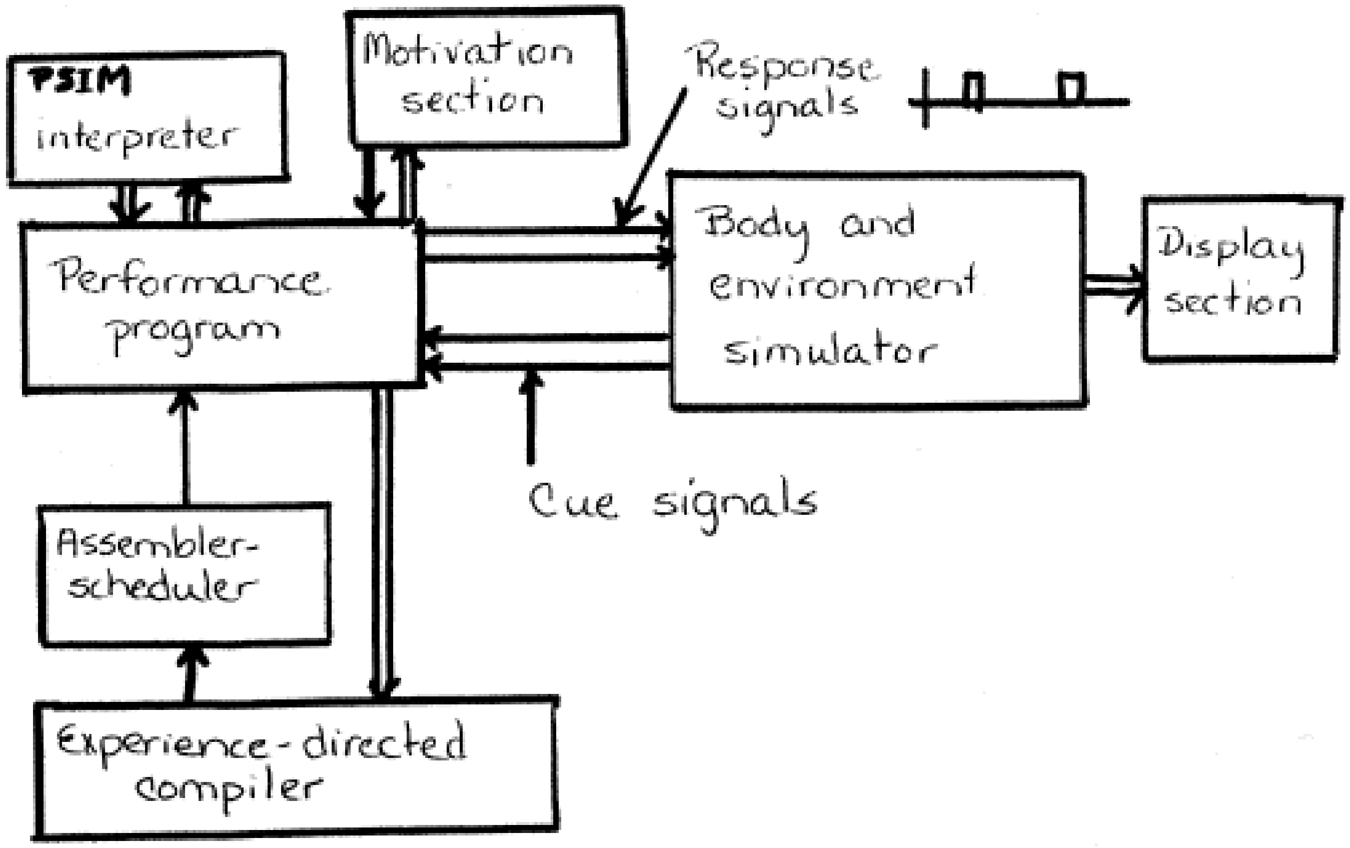
(4) (left cheek touch $\vee$ right cheek touch) $\rightarrow$ mouth touch

(5) lift hand $\rightarrow$ face touch



After the program has learned these connections, it will emit the behavior sequence "lift hand, turn head (left or right)," resulting in pleasure.

Below is a block diagram of INSIM1:



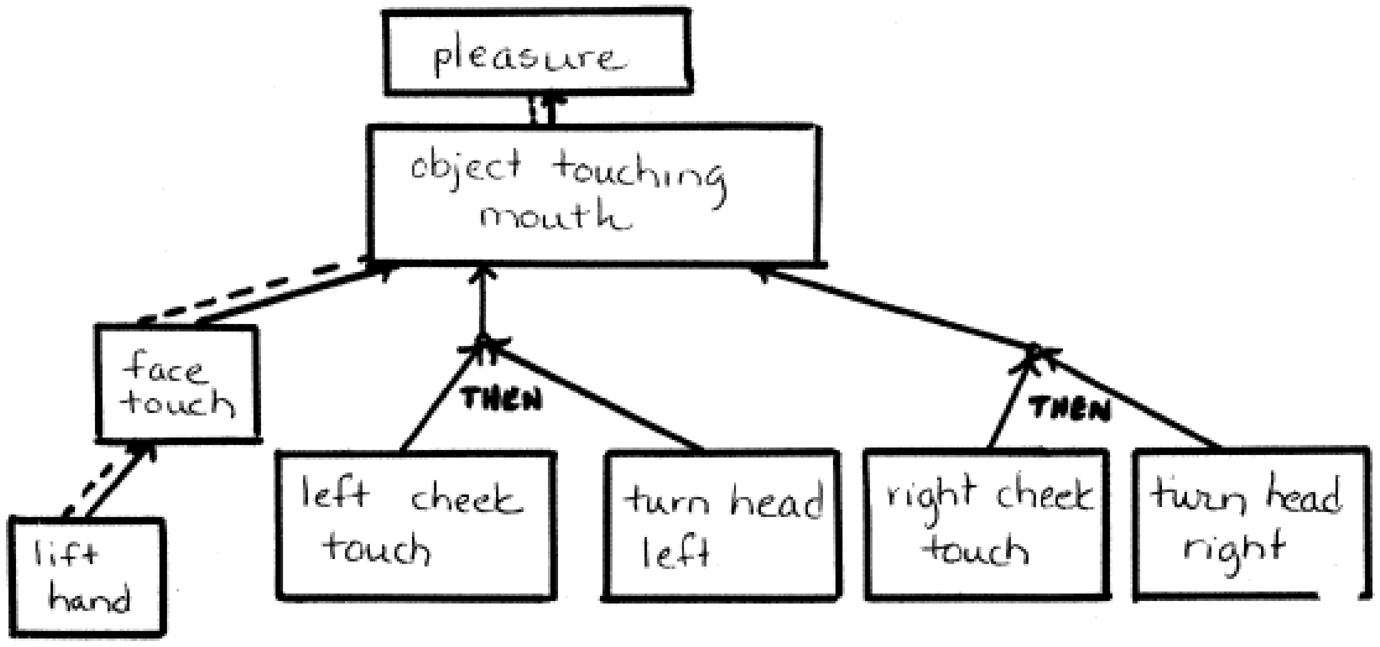
The <u>performance program</u> has the direct responsibility for synthesizing behavior. It is written in an interpretive language called PSIM (parallel simulator). The performance program receives stimuli from and sends responses to a <u>body and environment simulator</u>; the <u>display section</u> provides real-time monitoring on the cathode-ray tube. The <u>motivation section</u> activates the main goal (oral gratification or curiosity).

Relatively little of the performance program is innate. Most of it is generated by an experience-driven compiler, which is the core of the learning part of the program.

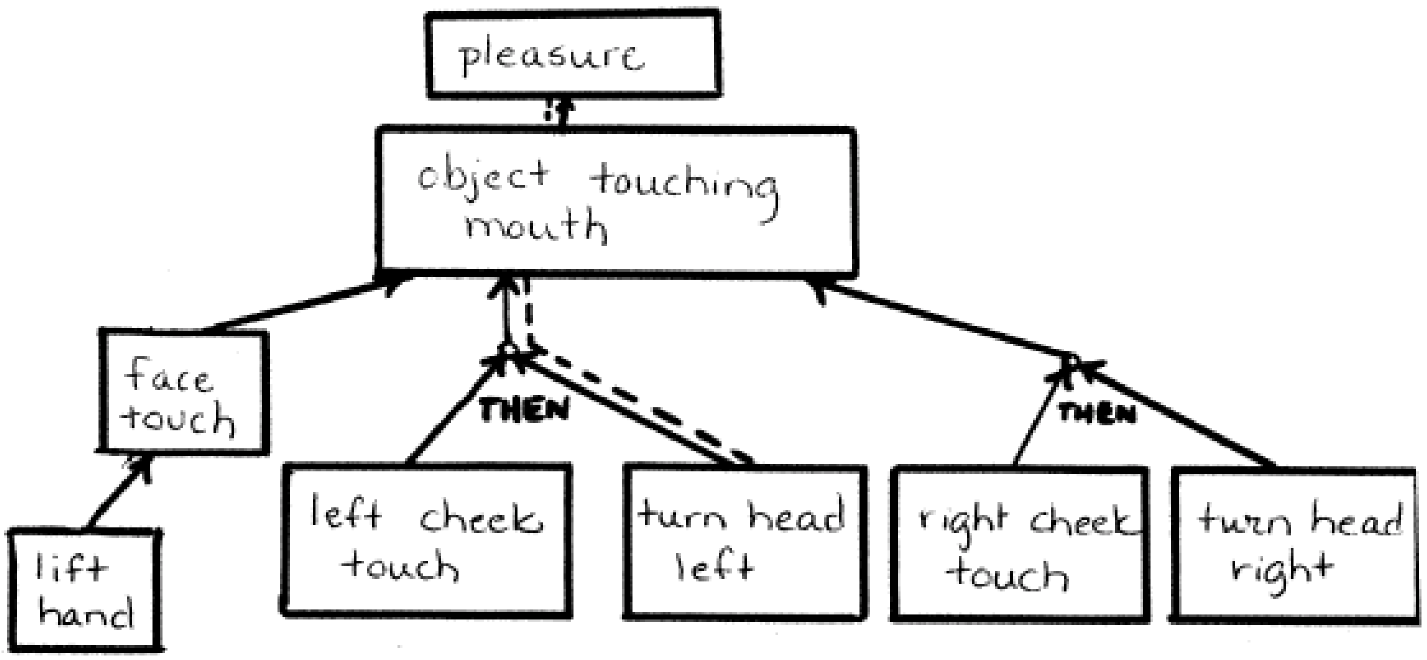
Causality is detected by statistical correlation; if a signal occurs on line A followed by one on line B, and if this sequence is repeated sufficiently many times, the program assumes that A causes B. The program is equipped for the simplest type of pattern recognition and concept formation: the formation of logical AND's and OR's of previously known variables. The program has an intellectual motivation system which causes it to exhibit simple forms of curiosity, play, and exploratory behavior.

II. The Performance Program

As described above, the performance program has the direct responsibility for receiving cues from the environment and emitting properly timed and sequenced behavior. It is coded in PSIM, a language which will be described in detail below. The performance program operates by activating various branches of the goal tree at the appropriate times. In the thumb-sucking problem, assume that the motivation section has activated the main goal "oral gratification." The first step is to activate the extreme left branch of the tree (the dotted line indicates activation):



The "lifthand" response at the bottom of this branch is emitted to the body and environment simulator. After a delay of roughly two simulated-time seconds a cue, e.g. "left cheek touch," comes back, indicating that the simulated hand has been lifted to touch the (simulated) left cheek. Next, the branch ending in "turn head left" is activated:



A "mouth touch" signal comes back from the body and environment simulator, indicating that this goal has been reached; the motivation section activates the oral gratification flag, "rewarding" the program for its successful effort.

The basic problem is to decide which branch of the goal tree to activate. (INSIM1 performance programs allow only one branch to be active at a time; hence there is no way to work on two goals simultaneously.) In a given situation, the decision is made in two phases, a feasibility study phase and a choice phase.

In the feasibility study phase, each path to the main goal is assessed, and an estimate is made of which path is the quickest and surest way to the main goal. Two numerical quantities are computed for each subgoal, GPR (global success probability) and a GC (global cost). The GPR of a subgoal is an estimate of the conditional probability that, if the program attempts to achieve the subgoal, it will succeed in reaching it.

A. Computation of GPR and GC

This section is devoted to a detailed discussion of how GPR and GC are computed. On a first reading, readers may skip to Section C on the choice phase.

GPR is defined recursively as follows:

(1) for a "response" (directly controllable) variable, such as "lift hand" or "turn head left", GPR=1.

(2) Suppose that A is one of several OR'ed subgoals of B; further, suppose that A is the "best" subgoal of B in the sense that it maximizes the Slagle coefficient ( ):

$$\frac{GPR(B)}{GC(B)}$$

Then GPR(B) = GPR(A) Pr (B/A), where Pr (B/A) is the conditional probability of B given A (i.e. the probability of getting from A to B) as estimated by the coefficient learning program PRDLRN, discussed below.

(3) Suppose that A1 and A2 are components of the ordered-AND goal A1TH2 (A1 then A2). Then GPR (A1TH2) = GPR (A1) * GPR (A2).

(4) Notwithstanding any of the above, if a goal has already been achieved, its GPR=1. A goal is defined as "already achieved" if the corresponding signal has occurred within the last five seconds.

Similarly, the GC (time delay) of a subgoal is defined recursively as follows:

(1) For a response variable, GC = 0.

(2) If A is the best of several OR'ed subgoals of B, then GC(B) = GC(A) + GPR(A) Delay $(A \rightarrow B)$.

(3) The GC of an ordered-AND goal A1THA2 (A1 then A2) is GC(A1THA2) = GC(A1) + GPR(A1)*GC(A2).

(4) Notwithstanding any of the above, the GC of a goal is 0 if the goal is already achieved (in the past five seconds).

To summarize, in the feasibility study phase, estimates are made of the success probability and time delay of each path to the main goal.

B. The Choice Phase

The next step is to activate the goal tree branch which is estimated, according to simple heuristics, to be the quickest and surest path to the main goal. A goal is active if, and only if, its WANT variable has the value TRUE.

C. Computation of the WANT Variables

(On a first reading, readers may skip to section E on the inner loop.) The WANT variable of a goal G is defined recursively as follows:

(1) If G is a main goal, WANT(G) = TRUE or FALSE as set by the motivation system.

(2) If A is one of several OR'ed subgoals of B,

WANT(A) = (WANT B) $\wedge$ (A is not already achieved) $\wedge$ (A is the best subgoal of B)) $\vee$ (A is a curiosity goal (see below)),

where "already achieved" means that the signal A has occurred in the last five seconds, and the "best" subgoal is that which maximizes

$$\frac{GPR(B)}{GC(B)}$$

(3) If A1THA2 is the ordered-AND subgoal "A1 then A2",

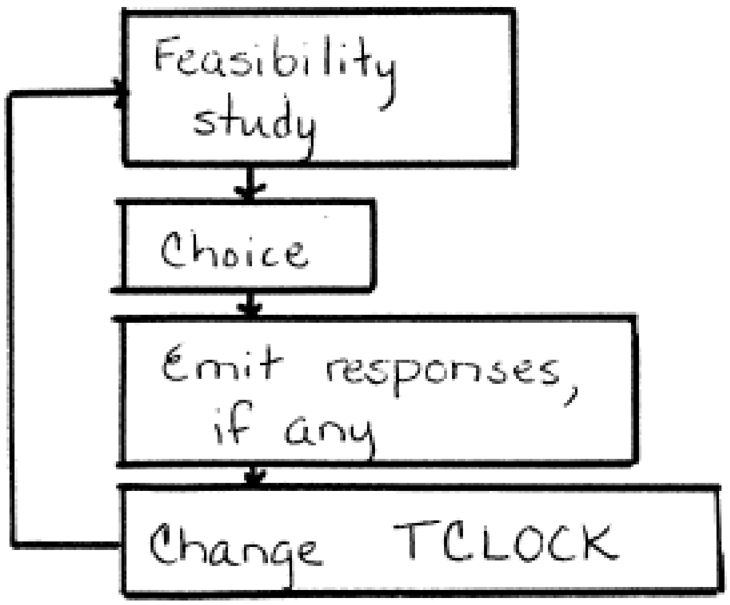
WANT (A1) = WANT (A1THA2) $\wedge$ (A1 is not already achieved)

WANT (A2) = WANT (A1THA2) $\wedge$ (A1 is achieved) $\wedge$ (A2 is not achieved).

(4) If G is a response (directly controllable) variable, WANT(G) causes the response to be emitted.

D. The Inner Loop

The feasibility study and choice phases are performed every time the simulated-time clock, TCLOCK, changes.



Thus the GPR, GC, and the program's decisions are constantly being updated on the basis of changing conditions. The PSIM interpreter ensures reasonable efficiency by recomputing only the variables which depend on some condition which has changed since the last TCLOCK time.

E. Discussion

INSIM1 performance programs incorporate simple heuristics which work well in cases where the assumptions on which they are based hold true.

Among the assumptions are:

(1) Success probabilities and time delays are assumed to be statistically independent. If this is not true, the chaining formulas used in computing success probabilities and time

delays will not be accurate.

(2) Predictions of success probability and time delay are based on the value of variables which may change with time. Suppose the program says that Pr(B/A) is high, and the $A \rightarrow B$ branch of the tree is selected. Then suppose that one of the variables used in computing Pr(B/A) changes before A is reached. If the new value of Pr(B/A) is low, this may indicate that B cannot be reached.

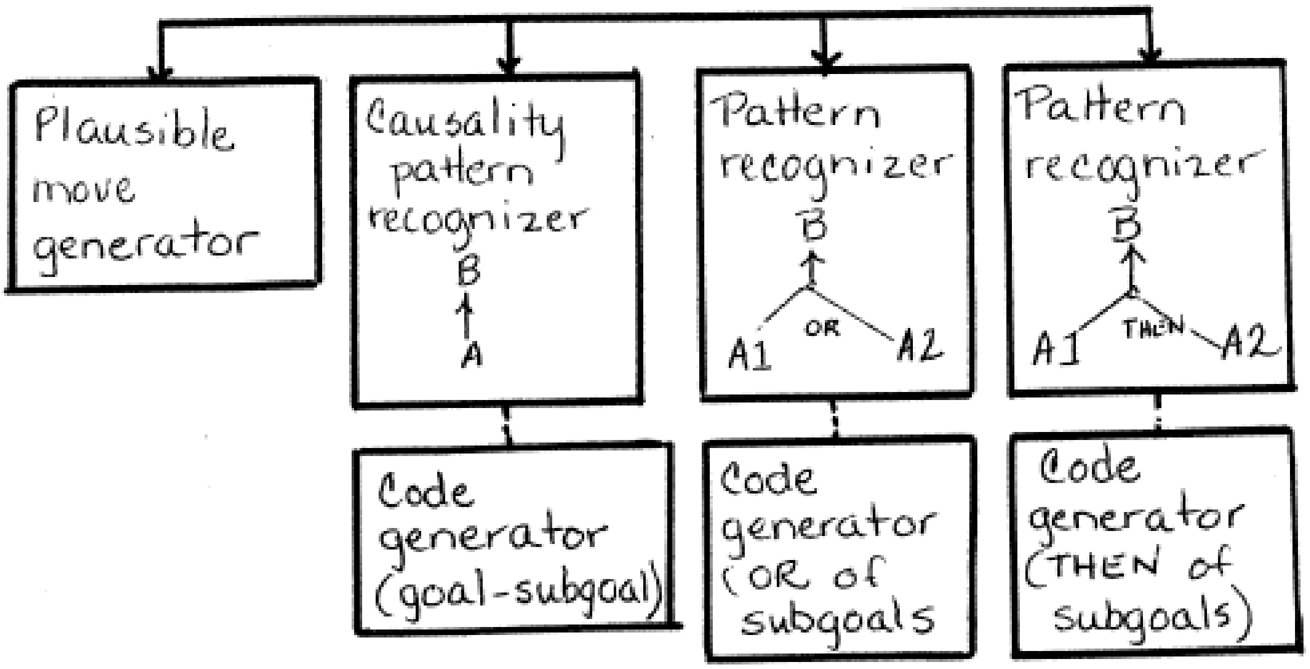
(3) It is assumed that goals do not conflict; i.e. that the achievement of one goal does not decrease the probability of achieving another goal.

Removing these performance limitations would require additional machinery beyond the scope of the INSIM1 project, such as a look-ahead method of the type used in chess programs.

F. The Experience-Directed Compiler

As mentioned previously, most of the performance program is coded by an internal compiler which, instead of using as its input a source code prepared by a human, is controlled by the experience acquired by the program as it interacts with its (simulated) environment. In keeping with the dictum that in order to learn something, one must know something already, the compiler incorporates the probability formulas described above, plus knowledge of basic aspects of the physical world, including time and causality.

The compiler consists of pattern recognizers, code generators, and a plausible move generator (not implemented at this writing).

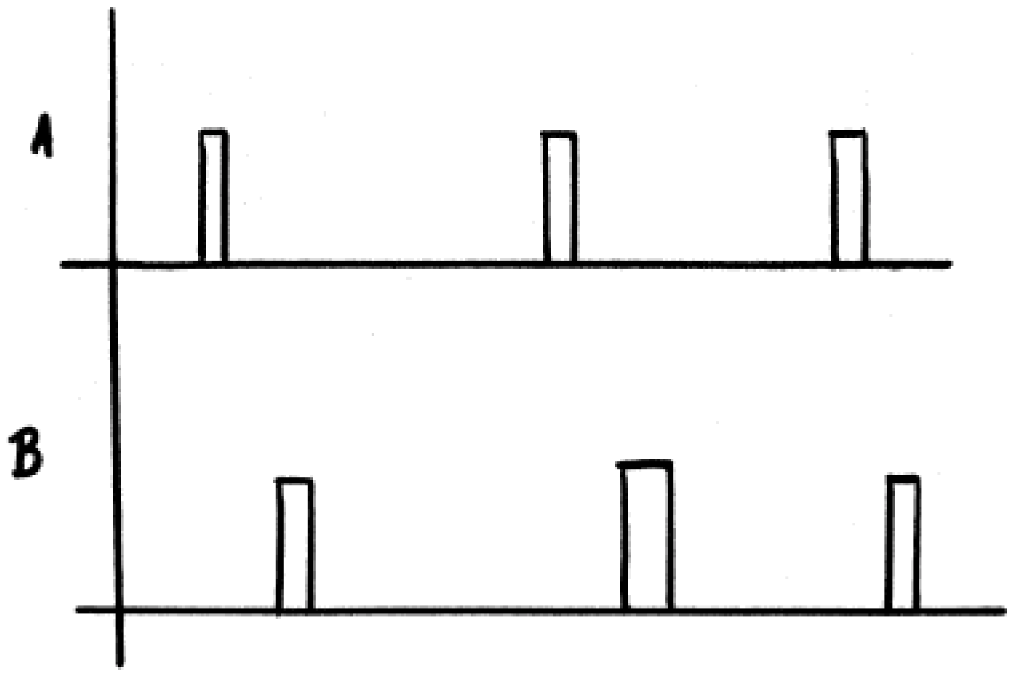

Plausible move generator
Causality pattern recognizer
B
A
Pattern recognizer
B
A1
OR
A2
Pattern recognizer
B
A1
THEN
A2
Code generator (goal-subgoal)
Code generator (OR of subgoals
Code generator (THEN of subgoals)

The plausible move generator is used instead of testing for causality between all possible variables A, B. The latter approach would involve on the order of n^2 tests, where n is the number of variables.

It is thecompiler which sets the upper limit on the program's ability to learn. For example, INSIM1 could never learn to play chess even with very long training, because the necessary pattern recognizers and code generators are not present.

The experience-driven compiler operates as follows: The program starts out with an innate main goal which is "oral gratification" in the thumb-sucking problem. First, the plausible move generator is called to generate a list of variables which are likely to be "relevant" to the oral gratification goals, and causality test links (indicated by dotted connections) are formed.

Next, the causality pattern recognizer learns which test links represent actual causal relationships. The pattern it is looking for is:



If a pulse on variable A is followed by a pulse on variable B sufficiently often, A is assumed to cause B. More precisely, if $Pr(B/A) - Pr(B|A \text{ or } \neg A) > 0.25$ after at least 15 pulses on A and 15 on B have occurred, A is assumed to cause B. The pulses on A and B must be less than five simulated-time seconds apart. (If there are any pulses at all on B, then a pulse on A will always be "followed" by a pulse on B if we wait sufficiently long.) $Pr(B|A)$ is estimated by the coefficient learning program PRDLRN, discussed below.

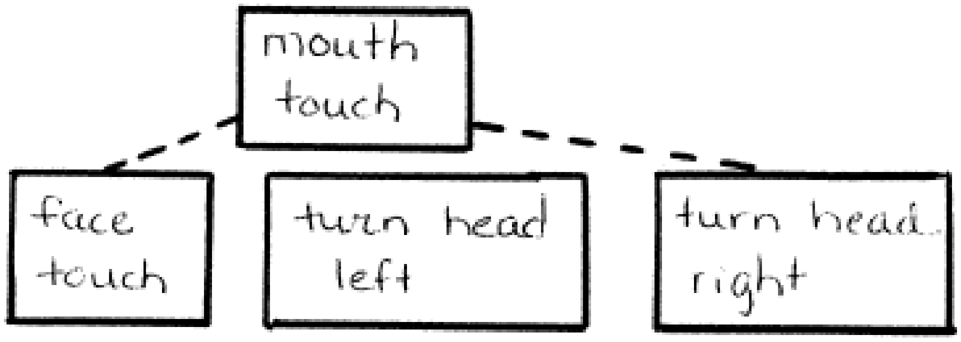
In some cases, it is sufficient to wait passively for a pulse on A. In other cases, the <u>curiosity section</u> of the performance program activates A as a goal in order to see if B follows (e.g. it activates "turn head left" to see if "mouth touch" follows); this is the "play" or "exploratory behavior" mentioned above. The curiosity section attempts to test links which are new and have not been tested many times; links where the initial variable, A, is reasonably easy to obtain; and where the final variable, B, is "biologically useful " (if one may use the term to describe a computer program) in that ability to obtain B would contribute to the program's ability to obtain primary reward. Specifically, the curiosity section tests the link $A \rightarrow B$ which maximizes

$$\frac{GPR(A)}{GC(A)} \text{ Satfunc}(A,B) \text{ Need}(B)$$

where Satfunc(A,B) (saturation function) decreases linearly from 1 to 0 as the number of times when A,B has been tested increases from 0 to 15. Need(B) is an index of how much the ability of the program to obtain primary reward would be improved by improvements in its ability to obtain B.

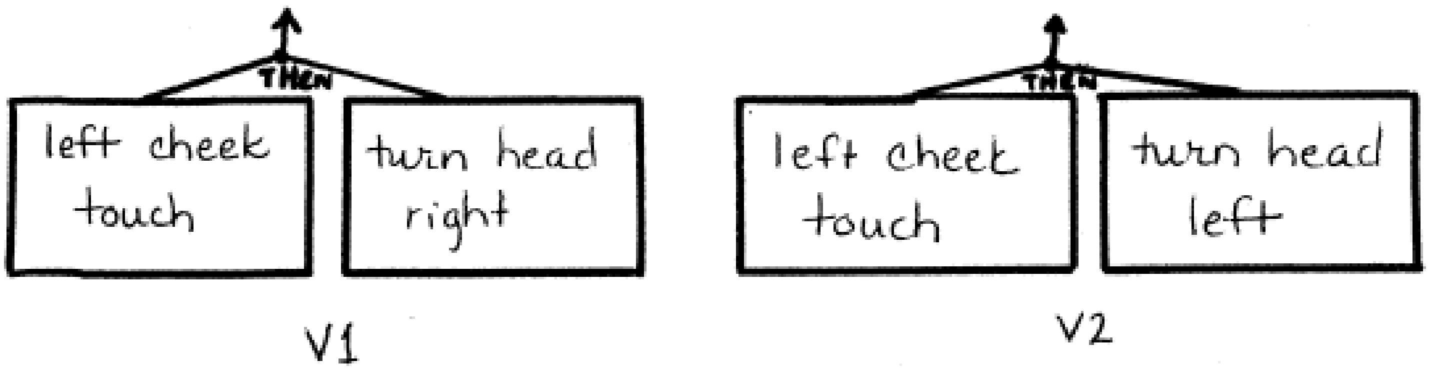
When the causality pattern recognizer detects that two variables, A and B, are causally related, the corresponding code generator is called to compile the link $A \rightarrow B$ in the goal tree. This code generator is a LISP function called MAKEORGOAL (A,B), so named because it also handles the case where A is one of several logically OR'ed goals. In LISP, the code generator turns out to be a straightforward and rather prosaic, if slightly long, program. Separate sections are provided for compiling the entries for WANT, GPR, GC, and each variable associated with the curiosity system. Each section looks up the names of the variables involved in the formula in question and substitutes them into the formula, using LISP's symbol-substituting capability.

In the thumb-sucking problem, the program first learns the links:



Although this version of the performance program will sometimes succeed in obtaining "mouth touch," it does not yet know which way to turn the (simulated) head.

Next, the plausible move generator is called to provide a list of variables to be THEN'ed with the partially successful subgoals. Causality test links are compiled for the ordered-AND variables. Among them are:



V1 correlates very poorly with mouth touch; V2 correlates very well. Since Pr (mouth touch | V2) is very high, the performance program will activate this branch, rather than the others, and the simulated infant will emit "turn head left" in response to "left cheek touch." Similarly, it learns to emit "turn head right" in response to "right cheek touch."

What is happening here is that the conditional probability figures, such as Pr (mouth touch | turn head left) are being used as a hill-climbing criterion in program space (Minsky,   ). (turn head left $\rightarrow$ mouth touch) works some of the time; INSIM1 forms new properties of the problem by combining properties which have proved useful in the past (Minsky,   ).

Finally, "face touch" is identified as a "biologically useful" variable, and the program learns to activate "lift hand"; when the (simulated) hand touches the face, the previously learned program takes over and completes the thumb-sucking operation.

It is interesting to note the similarity between this learning sequence and Piaget's observations on the learning of human infants. Although the real infant's learning is much more complicated, it follows the same gross sequence of stages; the real infant first learns to search from left to right with its head; then it learns which way to turn; then it learns to lift its hand and suck its thumb.

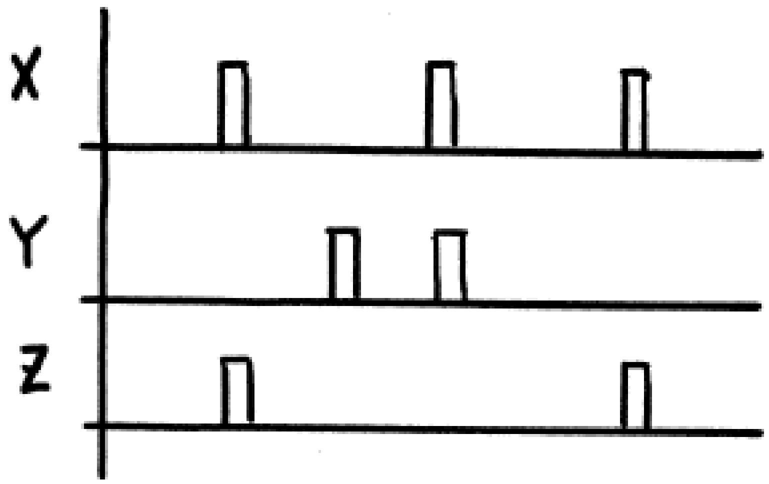
The remainder of this chapter is devoted to a discussion of the PSIM interpreter and the PRDLRN coefficient-learning program; it may be skipped on a first reading.

G. PSIM (Parallel Simulator)

The PSIM interpreter, embedded within LISP, handles the details of arranging the second-by-second occurrence of simulated events and relieves the compiler of the need to schedule the sequence of computations. A PSIM program consists of a set of variables, each of which has an S-expression which determines its value. E.g.:

```
(Z (AND X (NOT Y)))
(X (POISSON 0.1))
(Y (POISSON 0.1))
```

The POISSON expressions generate Poisson-distributed pulse trains with mean frequency 0.1 pulses per second. Whenever a variable, such as X, changes the variables which depend on it are automatically updated. A graph of X, Y, and Z versus simulated time will look something like this:



PSIM also handles the complications which arise when the goal tree is circular; in this case, an iteration procedure is used to calculate the GPR, GC, and WANT variables.

H. PRDLRN (Probablity-Delay Learner)

Conditional probabilities and time delays are estimated by a rather orthodox coefficient learning procedure (Minsky,). Suppose there is a link between A and B. Whenever A occurs, followed within five seconds by B, $Pr\ (B \mid A)$ is incremented by an amount θ (1-old value of $Pr\ (B \mid A)$), and Delay $(A \rightarrow B)$ is incremented by θ(actual delay - old estimate of delay). If A occurs, but not B, $Pr\ (B \mid A)$ is decremented by an amount θ (old value of $Pr\ (B \mid A)$) and Delay $(A \rightarrow B)$ is incremented by θ (5 seconds - old estimate of delay). It can be shown that this procedure gives an unbiased estimate of $Pr\ (B \mid A)$ and Delay $(A \rightarrow B)$, with an exponential weighting such that old occurrences of A affect the estimates less than new ones. θ, the delay coefficient, is currently 0.1.