DOCUMENT RESUME

ED 217 874 IR 010 260

AUTHOR Neches, Robert
TITLE Simulation Systems for Cognitive Psychology.
INSTITUTION Pittsburgh Univ., Pa. Learning Research and Development Center.
SPONS AGENCY National Inst. of Education (ED), Washington, DC.
PUB DATE 82
NOTE 56p.

EDRS PRICE MF01/PC03 Plus Postage.
DESCRIPTORS Artificial Intelligence; *Cognitive Processes; *Computer Programs; *Computers; Hypothesis Testing; Models; *Programing Languages; *Psychological Studies; Reading Processes; *Simulation; Theories; Word Recognition
IDENTIFIERS Collaborative Activation Based Production System; Heuristic Procedure Modification; LISP Programing Language; PRISM Programing Language; READER Model

ABSTRACT

Opening with a discussion of the role of computer simulation in cognitive psychology, this paper proceeds to examine the major perspectives on computer simulation as a tool for theoretical research on cognitive processes. The claim that computer simulation enforces rigor in theory specification is analyzed; the use of simulation as a method for exploring or validating theories is discussed, and empirical analyses of several programs developed for hypothesis-testing are reviewed; and simulation is considered as a source of new ideas about cognitive processing mechanisms, with specific attention devoted to the Heuristic Procedure Modification (HPM) program, the Collaborative Activation-based Production System (CAPS), and the READER model. Psychological simulation languages are then discussed, as are aspects of programming environments which facilitate simulation work. In closing, a new simulation language, the Program for Research into Self-Modifying Systems (PRISM), is described in detail. Eight figures and a 50-item reference list accompany the text. (Author/JL)

SIMULATION SYSTEMS FOR COGNITIVE PSYCHOLOGY

Robert Neches

Learning Research and Development Center

University of Pittsburgh

1982

To appear in Behavior Research Methods & Instrumentation, J. B. Sidowski (Ed.), 1982, in press. Reprinted by permission.

The research reported herein was supported by the Learning Research and Development Center, supported in part as a research and development center by funds from the National Institute of Education (NIE), Department of Education. The opinions expressed do not necessarily reflect the position or policy of NIE and no official endorsement should be inferred.

Abstract

Three views of the function of computer simulation in cognitive psychology are analyzed. The strong view that computer simulations will produce more rigorously specified theories is seen to be overstating the case. Two more pragmatic views are supported. One looks at computer method as a means of exploring or validating psychological theories. The other looks to computer simulation as a source of useful concepts. Several recent simulation efforts are presented as illustrations of these latter views. After establishing some perspective on the uses of simulation, the discussion turns to psychological simulation languages, and to aspects of programming environments which facilitate simulation work. A new simulation language, PRISM, is described. PRISM's design is intended as a response to some of the issues raised in this paper.

SIMULATION SYSTEMS FOR COGNITIVE PSYCHOLOGY

Robert Neches

Learning Research and Development Center

University of Pittsburgh

1.0 OVERVIEW

Although the primary purpose of this paper is to discuss simulation systems, how we view simulation as a methodology strongly affects our perceptions of what constitutes a useful simulation system. Therefore, the first part of this discussion considers several common views of the role of simulation in cognitive psychology. In the process of evaluating each of these views, I will be making some assertions about useful principles of simulation, and reviewing instances of simulation work which illustrate those principles. Once some perspective is established regarding simulation's uses, I will turn to a discussion of where I believe simulation work is heading. That discussion will consider the rise and fall of some past psychological simulation languages, as a means of focusing attention on aspects of programming environments that facilitate simulation work in general.

Finally, I'll close with a discussion of a particular class of psychological simulation languages, production systems. That discussion will focus on the design of a new production system language called

PRISM, which is being developed in collaboration with Pat Langley of Carnegie-Mellon University (Langley & Neches, 1981).

2.0 SIMULATION AS POLICEMAN OF THEORETICAL RIGOR

I'd like to start by exorcising a ghost, in the form of an extreme argument for simulation that was propounded rather vigorously in the late 1960's and early 1970's. This was the claim that computer simulation was a superior formalism for enforcing greater rigor in theory specification.

2.1 Five Claims For Computer Simulation

A strong example of this particular argument appears in Gregg & Simon's (1967) article using concept formation as a demonstration domain for information processing models. Embedded in that article were five claims for the advantages of requiring that running computer programs be associated with psychological theories:

- Inconsistencies would be prevented by the need to specify a particular set of operations in order to implement a hypothesized psychological process. The same set of operations would have to suffice for all cases in which that process was evoked.

- Untested implicit assumptions would be rendered impossible by the need to specify a complete set of processes. A program which does not specify processes completely could not run.

- Overly flexible theories which could too easily fit data would be prevented by the fact that computer programs contain no numerical parameters.

- Untestable theories would be eliminated by virtue of the specific sequence of operations generated by a program, which could be treated as predictions about intermediate processes. These predictions could be compared against process tracing data, such as verbal protocols or eye movements, thus allowing much more specific tests of a model (1).

- The need for a program to operate upon specific data would prevent finessing critical questions about encoding and representation.

There are some positive examples supporting these claims. John Anderson, one cognitive psychologist clearly influenced by the simulation approach (Anderson, 1976), has produced a very detailed theory which is often relatively specific in its claims. His work has stimulated a number of studies, both supporting and opposing.

However, in spite of positive examples such as his, it is hard to say that simulation was the causal factor in the development of a detailed model. Certainly the history of psychology contains a number of comprehensive theories not cast in a computational formalism.

Footnote 1: This, and the preceding point, is particularly important if one adopts Popper's (1959) view of science. Popper suggested that the dominant goal is to refute theories rather than support them, with a theory being "accepted" only so long as no evidence can be found counter to it. In that view, a theory is best if it is highly specific and therefore amenable to disconfirmation. In that case, either the cause for its disconfirmation leads to a new and better theory, or the failure to disconfirm lends credence to it.

2.2 Six Problems With The Five Claims

Furthermore, experience with simulation since the early days of Gregg & Simon (1967) has shown that there are a number of ways to avoid rigor while doing simulation work:

- A formal specification of a model needn't imply a comprehensible presentation; since programs are rarely presented in full with accompanying documentation, we remain dependent on verbal descriptions of the model. This can raise problems in determining whether the program performs as it does for the reasons claimed by its author. For example, see Hanna & Ritchie's (undated) analysis of Lenat's (1976, 1977) AM program, a system which has received a great deal of attention in the Artificial Intelligence community for its apparent ability to re-discover a number of interesting mathematical theorems. Hanna and Ritchie suggest several points that contribute to its performance, but where the actual program appears inconsistent with the general principles Lenat presented. They also raise instantiations of four of the five potential problems listed below.

- Programs frequently involve simplifying assumptions in order to facilitate implementation. These simplifications, however, cause the program to diverge from the theory it supposedly represents.

- Programs can be written to work only for a restricted set of examples, those presented in the write-up of the research. In the absence of some analysis of the formal properties of the domain, there is no automatic guarantee that the examples presented are representative of the domain, or that the principles required to

handle a given set of examples are sufficient to account for the entire domain.

- The inputs or database for the program can be structured in ways that simplify its task, but which are not necessarily psychologically plausible. That is, the real work of performing a task may be done before the program is started.

- Data or procedures supplied to the program to define different examples for it to handle may, in fact, constitute non-numerical parameters that give the program considerable flexibility in fitting psychological data. Newell & Simon (1972, page 56), for example, admit that the operators and table of differences supplied to GPS constitute such parameters.

- The programmer may hold back data or procedures that would have confused the program had it been available. That is, the program may appear to perform well not because it has the capacity to choose the correct action from all possibilities, but rather because the difficult choices are not offered to it.

For all the above reasons, there is no immediate assurance that a program's consistency with psychological data means the program is of psychological significance. Nor, on the other hand, is an inconsistency necessarily a sign of failure. For example, Newell & Simon (1972, page 472) admit to a number of exceptions to GPS' account of protocols obtained from subjects solving logic problems.

Although Newell and Simon are fond of claiming that the test of a a theory is a running program, this is no more true than claiming that the true test of an experiment's validity is a 0.05 significance level. The real question is how and why a particular result was obtained. The claim that computer simulation will necessarily lead to clearer and more rigorous psychological models does not hold up.

It is perhaps better seen from a historical perspective, as an argument stemming partly from the days of simpler programs, but primarily from a need to make a case for the respectability of simulation methodology compared to established mathematical modelling and experimental approaches. Unfortunately, the proponents of simulation approaches have, if anything, damaged the credibility of their case by overstating it.

3.0 SIMULATION AS A METHOD OF EXPLORING OR VALIDATING THEORIES

Therefore, I'd like to turn to some less ambitious views of simulation, in which a computer implementation is viewed not as a necessary formalism for expressing a model, but rather as simply one of several means for gathering information about it. Even this more restricted view may still be controversial.

3.1 The Significance Of A Running Program

One of the issues in the controversy is the significance of the fact that a program runs. L. Miller (1978) does a very nice job of summarizing the debate, which he suggests stems from alternative assumptions about the difficulty of theory validation. One side, he claims, believes that theories are easy to generate but difficult to

test. The other believes that a good theory is a significant and difficult accomplishment, and is accordingly more impressed by a demonstration of a model's sufficiency through the successful implementation of a computer program.

A related question has to do with the ultimate discriminability of psychological models. Anderson (1978), for example, has claimed that many different models can produce empirically identical predictions, and has even gone so far as to suggest that it is futile to try to distinguish which alternative is correct by experimental methods. Naturally, this claim has been disputed. Hayes-Roth (1979) has offered one of the more detailed responses, basically arguing that if two sets of processes are not identical, then it should be possible to find some form of process tracing data for which the two sets make different predictions. Without taking a firm position on the ultimate resolution to these questions, we still can say that simulation gives a means of exploring the plausibility of models where theoretical sophistication exceeds the state of the art in empirical testing.

In such cases, there are a number of ways that modelling can aid our thinking. The demonstration that a theory is sufficiently powerful to guide implementation of a working program is certainly encouraging for its credibility. Efforts to produce working programs can also lead to a better understanding of the computational requirements of a task, which in turn can help to constrain the set of plausible theories.

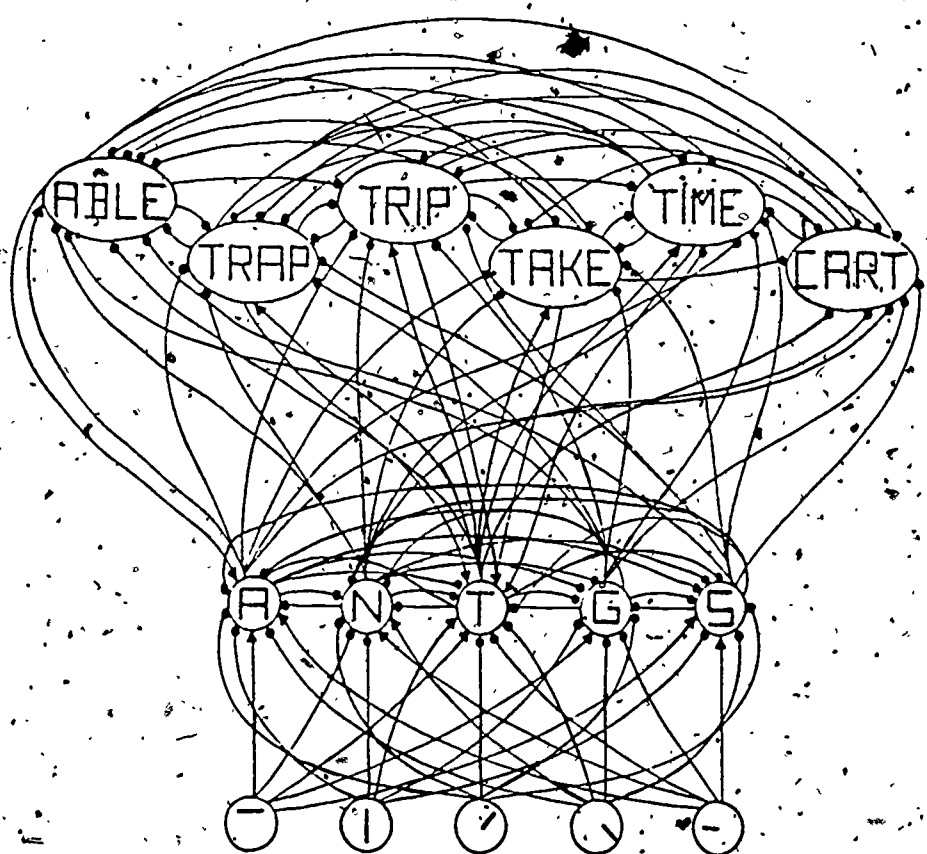
3.2 Empirical Analyses Of Programs

Another important contribution of simulation comes from our greater freedom to perform psycho-surgery on a program, since no clearance from a Human Subjects Committee is required in order to modify a computer simulation. This permits use of simulation for experiments that would be unethical or impossible with human subjects, experiments that can help in understanding the interactions between components in complex models. I'd like to offer McClelland & Rumelhart's (1981) model of word perception as an interesting example of this.

McClelland & Rumelhart (1981) were concerned with explaining a number of phenomena in the perception of words and letters in tachistoscopically presented displays. Among their key concerns were: (a) modelling the process of recognizing words and letters within words; (b) explaining the facilitating effect of pseudo-words for letter recognition; (c) explaining the sensitivity of the pseudo-word effect to expectations about what will be presented; and, (d) explaining the differential effects of various kinds of masks.

The model which they built assumed a highly-linked structure of nodes, representing hypotheses at various levels about what stimulus was presented. An example of such a structure is illustrated in Figure 1. Each node has an activation level associated with it, which represents the model's confidence at the current time in the hypothesis represented by the node. Hypothesis nodes vary in their baseline activation level.

Figure 1



Each node has a large number of weighted links to other hypothesis nodes. Excitatory links send activation to hypotheses consistent with a node. Inhibitory links decrease activation of inconsistent hypotheses.

The activation of a node at any point in time is a function of its baseline activation and the excitatory and inhibitory activation received from related hypothesis nodes. The function used modulated the activation level to keep it within a restricted range and allow for time decay. Activation reverberates through the network, and at some point in time whichever hypothesis is most active at that point is accepted as true.

In this model, the word superiority effect and the facilitating effect of words on letter recognition were explained in terms of activation flows to and from nodes at the word hypothesis level. The facilitating effect of pseudo-words on letter recognition could be understood as an outcome of partially activated word hypotheses reinforcing the letters. For example, the pseudo-word "TROP" contains letters which would activate hypotheses such as, "TRIP", "TRAP", and "PROP"; these, in turn, would send activation back to the hypotheses for the letters "T", "R", "O", and "P". Finally, the effects of various kinds of masks were explained in terms of the relative times at which activation for the mask grew to levels sufficient to interfere with activation for a target.

This last effect deserves discussion in some more detail, because it nicely illustrates some of the advantages obtained through computer modelling. The general phenomena which McClelland and Rumelhart tried to capture was as follows. When a tachistoscopically presented target

display is followed closely by presentation of a mask display, a number of factors affect the extent to which the mask will interfere with recognition of the target. The basic findings of interest involve comparing letter and word recognition for three different kinds of masks: feature masks consisting of letter-like geometrical shapes, letter masks consisting of non-word letter strings, and word masks. A number of studies have shown that letter recognition is about equally affected by all three kinds of masks, while word recognition is markedly less affected by feature masks than by letter or word masks.

Given the formulation of their model, the uniform effects of the three different kinds of masks on letter recognition are easily understood. All three kinds of masks quickly engender competing hypotheses at the letter level. These can depress the correct hypothesis' activation through their inhibitory links before that hypothesis can reach its peak activation level.

In the case of word recognition, the difference in effects between feature masks and others is somewhat more complicated to understand. McClelland & Rumelhart, in spite of a long and fairly detailed discussion of their model, do not make it clear why it produces the desired effect. (This is worth noting, in the light of Gregg & Simon's claims that computer simulation would eliminate exactly this kind of uncertainty.)

It appears their explanation is that random feature displays weakly activate many different letter hypotheses, rather than strongly activating a few. Thus, none of the competing alternatives have enough strength for their inhibitory links to have an immediate effect on the activation for the correct hypothesis. One indication that this is

indeed the intended explanation comes from their report that the program was very sensitive to the degree of similarity between features in the mask and the target.

This is an interesting point, because we see here that the program is perhaps just as complex for an outsider to understand as a verbally stated model. However, there are some real differences in the value of a program over a verbal model in situations where the complexity of a theory obscures its implications. With the program -- unlike a verbally expressed theory -- it is possible to perform manipulations to help understand exactly what factors contribute to its performance. For example, having determined that the program was sensitive to similarities at the feature level, McClelland and Rumelhart set out to equate their stimuli in order to eliminate that confounding factor. Doing that required coming up with feature, letter, and word masks which all three had just as many features same/different with respect to the target display. Worse yet, to properly equate the stimuli, the equivalences had to hold letter-by-letter, for each letter position in a four character string.

This would be a rather daunting task if the stimuli had to be created for human subjects in an experimental design of any statistical rigor. It is difficult to create even one grouping of a target word and three masks which would satisfy these criteria. Fortunately, in evaluating the performance of the program, one is all that is needed. Since the program is a deterministic entity, there is no concern of statistical error. When running experiments with a program, the only concern is with finding a range of inputs that verify the generality of the results. The need to be concerned with noise, or the statistical

reliability of measurements of the program's performance, is eliminated.

Even with the statistical issue of noise eliminated, though, it is still difficult to construct stimuli in this particular case. McClelland & Rumelhart's ability to do so illustrates yet another virtue of models implemented as running programs, the ability to turn thought-experiments into real tests of a theory. To create stimuli meeting the desired criteria, they simply modified the knowledge base of their program. For example, they selected as a target string the word, "MOLD". As a letter mask, they selected the string, "ARAT". In the specialized character font used in the experiments simulated, the letters of "ARAT" and the letters of "MOLD" had, respectively, 2 similar features in the first letter position, 3 in the second, 2 in the third, and 2 in the fourth.

It was easy to produce a feature string with the same number of similarities to the target string "MOLD". Where the constraints upon the stimuli become tricky is in finding a common four-letter word which also has the same pattern of similarities. However, because a program can be much more easily modified than a human mind, McClelland & Rumelhart were able to sidestep the constraint. After obtaining the results of running their program with the letter string "ARAT" used as the mask for "MOLD", they simply modified the program's database so that "ARAT" was now represented as a known word. When they then ran the program again, the results of the new run could be interpreted as representing a word mask rather than a letter mask. Thus, they were able to explore the effect of top-down knowledge about words without the confounding effects of feature differences due to different letter

strings.

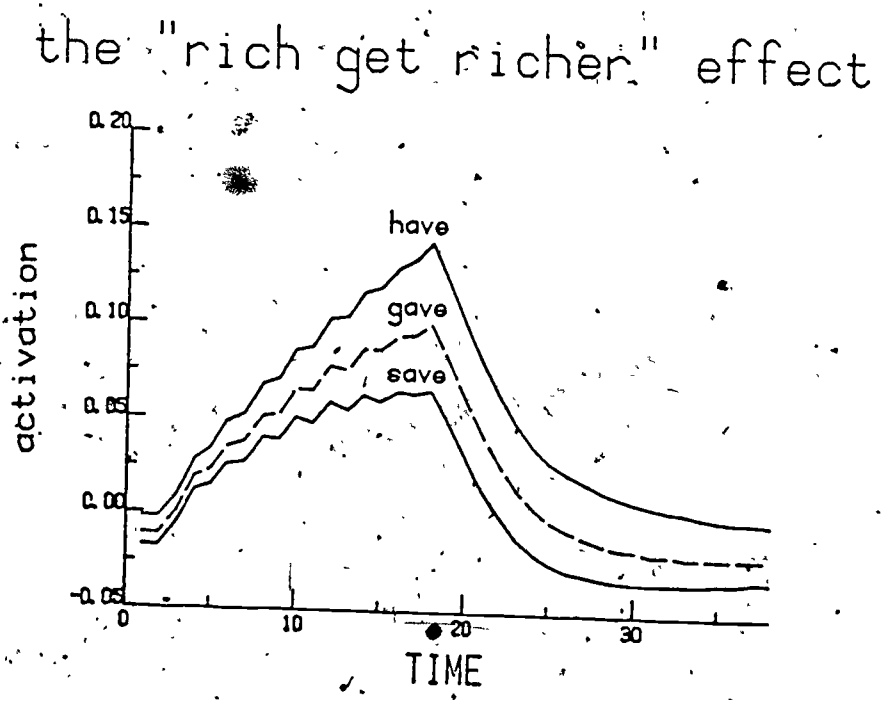
To see where some of those confounding effects could be produced, and to see another virtue of analyzing the performance of a computer model, we need to consider some other observations made by McClelland & Rumelhart.

Since programs can be modified at any point, it is possible to insert code to record virtually any kind of data about its run-time characteristics. This can permit one to make observations about implications of a model which might not come out nearly as clearly otherwise. For example, tracing the time course of activation flow enabled McClelland & Rumelhart to analyze three different factors influencing activation level.

The first they called the "friends and enemies effect". Activation is clearly going to depend on the number of excitatory and inhibitory links from other active nodes. Thus, the likelihood of a hypothesis being accepted, whether correct or not, is partly dependent on the relative amount of knowledge which the system has stored about it.

The second effect they called the "rich get richer" effect, the empirical observation that feedback loops inherent to the structure greatly accentuate over time any initial differences in baseline activation levels. This is one of several aspects of the model which offer accounts of expectation effects. In particular, by making baseline activation encode word frequency, they were able to simulate common frequency effects. Figure 2 illustrates this, by showing how small initial differences in activation due to differing frequency were

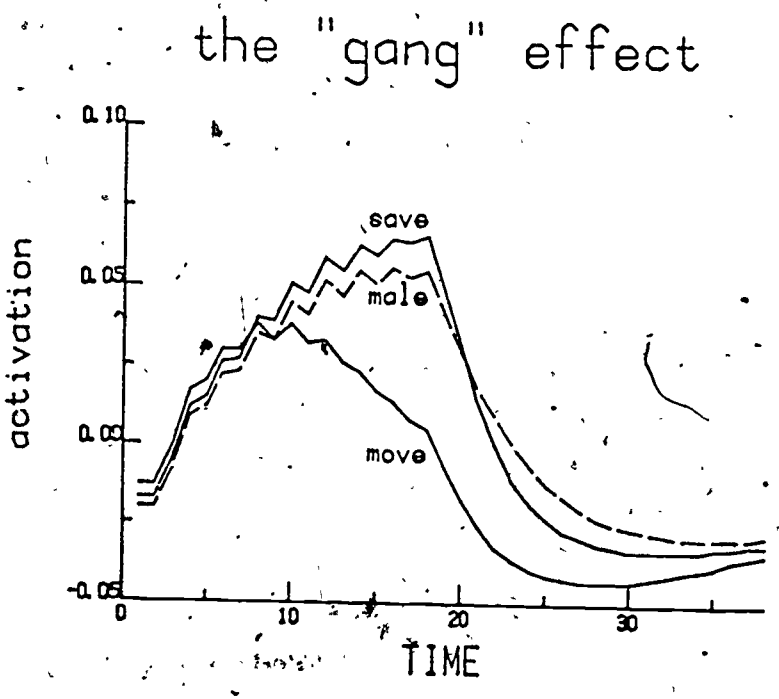
Figure 2


the "rich get richer" effect
activation
0.20
0.15
0.10
0.05
0.00
-0.05
0
10
20
30
TIME
have
gave
save

enhanced over time for three alternative hypotheses entertained by the program when presented with the string "MAVE". Note that all three hypotheses have three letters in common with the presentation string, and thus all receive equal bottom-up support.

The third effect was called the "gang effect". Observation of the program showed that strong hypotheses at a given level indirectly reinforced a subset of their competitors at the same level, those that depended on the same supporting evidence. This is because a hypothesis node sends activation to lower-level nodes, which in turn send increased activation not only back to that node, but also to all other higher-level nodes to which they are linked. Figure 3, for example, shows how three additional hypotheses fare over time in response to the same presentation string, "MAVE". Once again, all three alternatives have three letters out of four in common with the string actually presented, and so start out with initial bottom-up activation. However, "SAVE" indirectly receives activation from five other word hypotheses that boost the activation of the letters "A", "V", and "E" (e.g., "HAVE" and "GAVE"). Similarly, the program had stored five other words involving the letters "M", "A", and "E", and those alternative word hypotheses boosted the activation levels for "MALE" by way of those three shared letter hypotheses. On the other hand, there were no other hypotheses involving "M", "V", and "E" to indirectly support the hypothesis that the word seen was "MOVE". Thus, its activation is markedly lower than for the other alternatives.

Figure 3


the "gang" effect
activation
0.10
0.05
0.00
-0.05
0
10
20
30
TIME
save
male
move

3.3 There Are No Simple Standards

It is interesting to note that this simulation does not at all fulfill the promised advantages of simulation outlined by Gregg & Simon (1967), but instead illustrates the objections to their claims outlined in section 2.2. We were promised specificity through parameter-free models; McClelland and Rumelhart present a full-page table listing parameters; and vary the settings in simulating different experiments. We were promised deeper concern with encoding and representation; they present a system which pre-codes information about letter position (and which requires creating such a large number of links for exciting consistent hypotheses and inhibiting inconsistent alternatives that one has to wonder about the psychological processes required to add a new piece of knowledge). Finally, we were promised extensibility to related tasks; they presented a program which could not even easily be modified to handle five-letter words.

However, these objections really do injustice to what we instinctively know is a respectable piece of work. The problem is with the standards offered by Gregg & Simon, which basically amount to a promise that we will never again have to think hard to understand or evaluate someone else's work. Those standards do not fully capture what can be gained by simulation.

McClelland & Rumelhart's observations about interactions between components of the model are significant because of their implications for other work, a point which I'll return to below. What is of interest for the moment, though, is that the ability to perform empirical analyses of a program has enabled them to provide greater insight into the implications of their model. In addition to

information about how well the model accounts for a body of data, the capacity to perform experiments and make observations on the program means that we can also get information about why the model succeeds or fails.

4.0 SIMULATION AS A SOURCE OF NEW IDEAS

Another view of simulation is as a source of new ideas about information processing mechanisms, which implies a close partnership between cognitive psychology and artificial intelligence. Psychology, in spite of recent claims to the contrary, has made several contributions to AI. Among them are the notions of means-ends analysis embodied in GPS (Ernst & Newell, 1969; Newell & Simon, 1972), of discrimination nets (Feigenbaum, 1961; Simon & Feigenbaum, 1964), and of various semantic network representations (e.g., Kintsch, 1974; Norman & Rumelhart, 1975; Anderson, 1976).

Psychology has certainly been influenced by AI. Winograd's (1972) SHRDLU, for example, was considered of sufficient importance to have an entire issue of Cognitive Psychology devoted to it. Another important, although perhaps not as well-known, example is the HEARSAY speech understanding system (Erman & Lesser, 1975). That system introduced notions of a central memory structure shared by co-operating parallel knowledge sources; these notions have influenced psychologists in topics ranging from models of reading processes (Rumelhart, 1977) to planning (Hayes-Roth & Hayes-Roth, 1979). Scripts (Schank & Abelson, 1977), frames (Minsky, 1975), or schemata (Bobrow & Norman, 1975) have generated a number of lines of research, as has the work on story grammars (Rumelhart, 1975; Mandler, 1977; Thorndyke, 1977).

Although the examples just mentioned are all cases where ideas about processes have been transferred fairly directly, simulation work can have a much more subtle impact on psychological thinking. This is because solutions to sub-problems encountered in the course of implementing a program can turn out to have implications for psychological issues that the program was not originally intended to address. Often, this can help us gain a teleological understanding of mechanisms, by making us aware of constraints that necessitate their existence or force them to operate in a particular way.

All computer programs are fundamentally concerned with issues of control and focus of attention (or, to put it less elegantly, getting the right things done at the right time). Thus, the process of developing a simulation can suggest domain-independent mechanisms which other researchers can apply in developing models of behavior in quite different topic areas.

To illustrate these rather abstract claims, I will first discuss some of my own work on a learning simulation called HPM, then describe a simulation of eye fixations in reading (Thibadeau, Just, & Carpenter, 1981), and briefly return to McClelland & Rumelhart's (1981) word perception model. I will try to show how these disparate systems contribute a model of sloppy errors in algebra problem-solving.

4.1 HPM: An Example Of A Spin-off Discovery

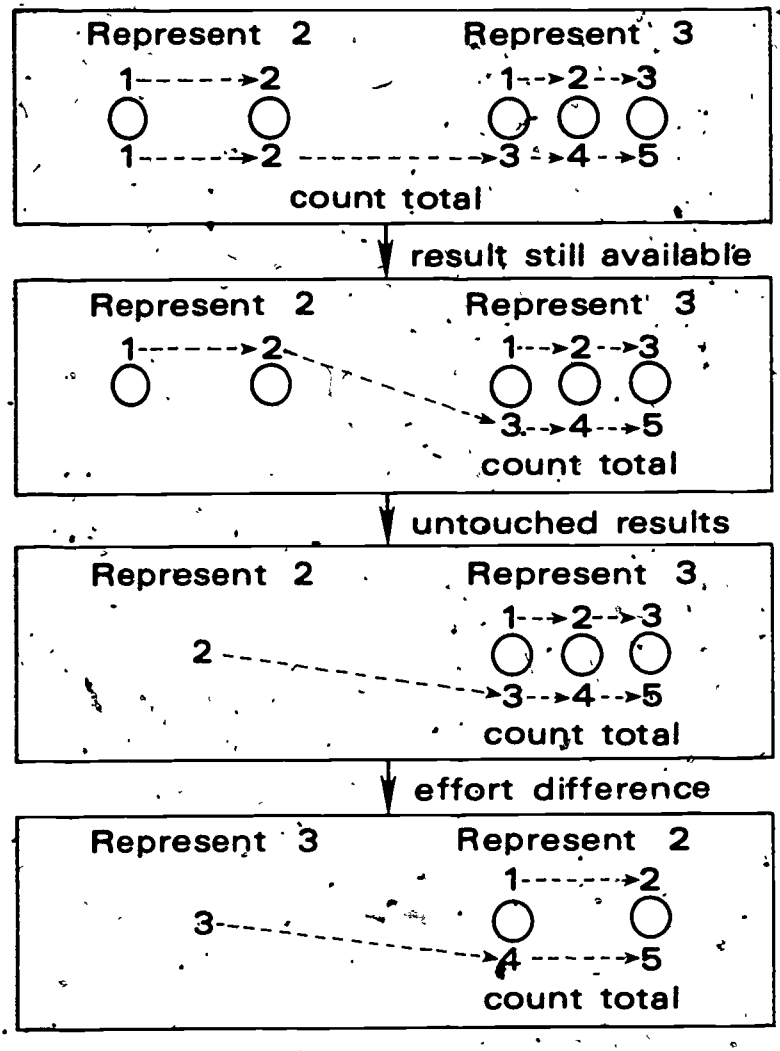
The HPM (for Heuristic Procedure Modification) program is a model of learning through the incremental refinement of procedures (Neches, 1981a, 1981b). Although primarily concerned with learning, it turns out to provide a new explanation for an old observation from the days

of gestalt psychology called the Zeigarnic effect. (For an English description of this effect, see Lewin, 1935, pages 243-244.) The effect, which Gestaltists interpreted as illustrating the phenomenon of "closure", boils down to the observation that delayed recalls of a task are richer and more detailed when subjects were stopped part-way through the task than when they were allowed to carry the task through to completion.

In order to make clear HPM's account of this phenomenon, it is necessary to provide some background about the program. HPM is a production system, which means that it belongs to the class of programming languages in which procedures are specified as a set of condition-action rules and data is represented as propositions in a working memory. The system runs through a cycle of finding the set of productions whose conditions are satisfied by the current contents of working memory, selecting a subset of those rules for execution, and modifying the contents of working memory according to the actions specified by the rules selected for execution.

The program was inspired by protocol studies by myself (Neches, 1981b) and others (e.g., Anzai & Simon, 1979) indicating that people use a number of common-sense heuristics to improve their procedures on the basis of experience applying them to a task. Most of the simulation work has concentrated on getting the system to acquire an addition strategy similar to that used by many second-graders, given a simpler strategy employed by most pre-schoolers. Figure 4 shows the heuristics which seem to be most relevant to this task, along with a sequence of strategies that the system discovers. The initial strategy adds two numbers by counting out a set of objects corresponding to each

Figure 4


Represent 2
Represent 3
1 2
1 2 3
1 2 3 4 5
count total
result still available
Represent 2
Represent 3
1 2
1 2 3
3 4 5
count total
untouched results
Represent 2
Represent 3
1 2 3
2
3 4 5
count total
effort difference
Represent 3
Represent 2
1 2
3
4 5
count total

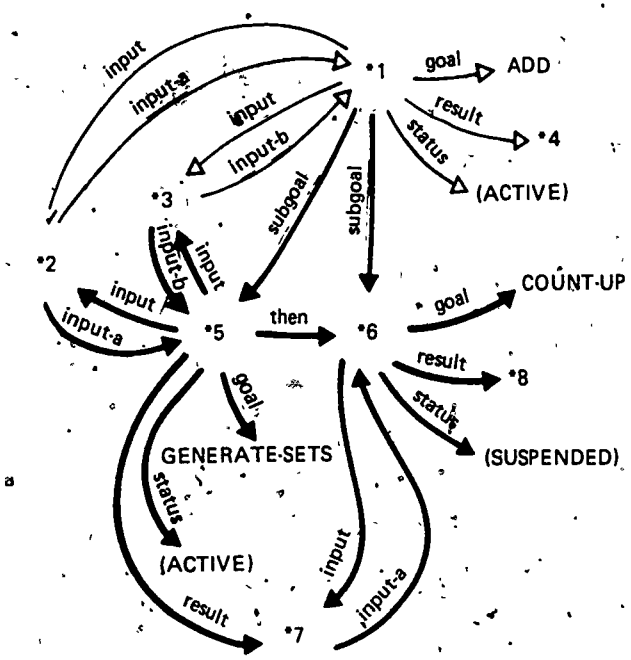
addend, combining those two sets, and counting the total set. The final strategy adds the numbers by incrementing the larger addend a number of times given by the smaller addend.

HPM was designed as a vehicle for exploring the problem of operationalizing heuristics such as those in Figure 4. Thus, the kinds of questions I was concerned with were ones like, "What sort of information about a procedure is necessary in order to apply heuristics like these?".

The answer embodied in HPM involves solving problems by setting up a hierarchical goal structure not unlike Sacerdoti's (1977) planning nets. Productions in HPM respond to nodes in a partially-constructed goal structure by adding propositions that further elaborate the goal structure. Whenever a production fires, a linkage is established between the propositions which satisfied its conditions (i.e., caused its firing), and the propositions which were added as its actions. This information allows HPM to implement heuristics like those of Figure 4 as sets of productions which look for configurations in goal structures indicative of inefficiencies. The program represents learning by using the information to construct new productions, with conditions that cause them to fire in circumstances when the inefficiency is likely to be repeated. The information allows the productions to construct actions for the new productions that cause the system to sidestep the inefficiency.

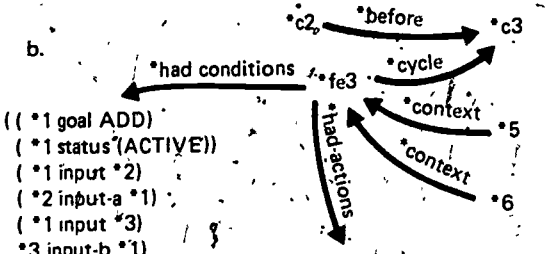
Figure 5 illustrates the structures in HPM's memory after executing its first production for addition in response to an externally supplied goal to add two numbers. When we remember that the semantic network shown in this figure represents only knowledge about

Figure 5



a.

b.




```
(( *1 goal ADD)
 ( *1 status (ACTIVE))
 ( *1 input *2)
 ( *2 input-a *1)
 ( *1 input *3)
 *3 input-b *1)
)

    (( *5 goal GENERATE-SETS) ( *1 subgoal *5)
     ( *5 status (ACTIVE)) ( *5 input *2)
     ( *2 input-a *5)  (*5 input *3)  (*3 input-b *5)
     ( *5 result *7)  (*5 then *6)
     ( *6 goal COUNT-UP)  (*1 subgoal *6)
     ( *6 status (SUSPENDED))  (*6 input *7)
     ( *7 input-a *6)  (*6 result *8)
    )
```

the first of a large number of steps to be taken, it is easy to see that a huge body of information must be retained in order for the system to represent a complete problem-solving sequence. (For an explanation of the necessity of the information retained, see Neches, 1981b, section 5.2.)

From both the computational consideration of minimizing the size of the database to be searched, and the psychological consideration of limited short-term memory, it was essential to have some mechanisms in the system which would cut down the number of propositions required for consideration without eliminating any critical information.

The mechanism adopted in HPM assumed an extremely rapid decay of working memory contents; propositions drop out of working memory unless used within two processing cycles. The propositions in working memory consisted of those required to specify the current goal, plus a set brought in from long-term memory by a spreading activation process. To reduce the number of propositions brought in from long term memory, activation was assumed to spread unevenly through the semantic network, with the primary direction in which it spread being dependent on the processing status of the current goal.

Specifically, when a new goal is initiated, HPM sends activation down through the network to retrieve information most likely to be helpful in deciding how to process the goal. When an old goal is terminated, HPM sends activation up the hierarchy towards higher goals and sideways towards planned successor goals, thus retrieving information most likely to be helpful in deciding what action to take next. Although this part of the model was developed in response to computational overloads produced by large semantic structures, it turns

out in retrospect to provide a psychologically plausible account of the Zeigarnic effect. In this account, the effect is an outcome of associative retrieval processes primarily intended to minimize the size of working memory needed for processing goal structures.

Assume that, as in HPM, a goal structure is built as a task and is carried out in which goal nodes are represented as either active or completed. In the case where the task is interrupted before completion, the rapid decay process causes their loss from active memory; they are, however, retained in long term memory. The instruction to give a recall causes retrieval of some of the higher-level nodes in the goal structure, since these are the nodes that define the task. Because these goals are represented as active, their return is treated as a re-initiation, and activation is sent down the network according to the processes outlined above. This retrieves a set of nodes which contains more detailed information about the task, since it consists of the more specific sub-goals set up to perform the task, along with information about the operands of those goals.

On the other hand, if the task is allowed to go through to completion, the goal nodes are all represented as completed when they return to long term memory. If the same higher-level nodes are retrieved due to a recall instruction in that case, HPM will try to send activation up and sideways through the network. Since the goals it works from are already near the top of the structure, there is simply not much up to go. HPM therefore retrieves a smaller set of propositions, which furthermore consist of more general and abstract propositions because they are drawn from near the top of the goal structure.

The significant point of this example is that the demands of formalizing a model in computational terms led to new ideas about issues not initially seen as related to modelling learning processes. HPM, although basically a model of learning, led to development of a notion of directed activation -- a distinct variant upon current notions of unfocused spreading activation (Collins & Loftus, 1975; Anderson, 1976). An additional property of the simulation is that it gives us some insight into the teleological role of activation in an information processing system. The simulation suggests that it should be viewed not only as a mechanism for focus of attention or information retrieval, but also as a component of a larger mechanism for minimizing working memory loads. In that larger mechanism, activation may serve to enable relatively drastic measures for eliminating propositions from active memory, by providing an assurance that critical propositions will return when needed.

4.2 READER And CAPS: An Example Of Concern With Control Processes

It is worthwhile to consider another example of directed activation, Thibadeau's READER model, which develops the notion in a much more sophisticated way. Thibadeau (1981; Thibadeau, Just, & Carpenter, 1981) has developed a production system language called CAPS in order to implement the READER model. CAPS is a programming architecture of some interest, only in part because it illustrates another useful property of simulation research: the development of general notions of control and focus of attention.

READER's mission is to account for gaze duration data from eye movement studies of reading. It is similar in some respects to McClelland & Rumelhart's word perception model, but differs in implementation and models a broader range of processes. The similarities stem from the notion of nodes representing hypotheses with activation levels representing confidence in the correctness of the hypothesis, excitatory relations to other hypotheses consistent with a given hypothesis, and inhibitory relations to others which are inconsistent. Rather than doing parallel processing on a feature array representing a four-letter character string, as McClelland and Rumelhart's program did, READER sequentially processes a string of letters and spaces representing a paragraph of text. Hypotheses in READER are maintained at the letter-cluster, word, syntactic, and semantic, levels. The system tries to do as much as possible at all levels before moving on to the next input element. These properties allow the model to explain gaze durations in terms of the time required for hypotheses to rise above the threshold for acceptance and thus allow the system to move on.

The READER model offers explanations for a number of effects. For example, at the word encoding level, the sequential processing of the input string causes the system to take more time to activate longer words, reproducing the linear increase in gaze duration found in data from human subjects. Gaze duration also turns out to be a log function of word frequency, a phenomenon modelled in READER as essentially similar to McClelland & Rumelhart's "rich get richer" effect on baseline activation levels. At the syntactic parsing level, the system displays a number of effects similar to those observed in the human data, most of which occur because of the way that interacting semantic

and syntactic processes contribute to activation levels of syntactic hypotheses.

Among other things, the collaboration between semantic and syntactic processes allows the system to parse difficult noun phrases like, "the greater the mass" (det adj det noun). It also produces the negative correlation observed in humans between the number of modifiers in a noun phrase and the fixation time for the head noun. The more modifiers there are, the more semantic constraints imposed, thus pre-raising the activation levels for likely candidates for the noun itself, and thereby decreasing the time required to raise the correct alternative above the threshold for acceptance. Much the same process underlies READER's ability to duplicate human subjects' tendency to skip over function words entirely.

Finally, the processing structure of the READER system, which enables it to do as much processing as possible at all levels before moving on to the next input, allow it to reproduce several effects at the semantic level, such as increased gaze durations at the first mention of a topic and at the end of sentences.

Thibadeau has found himself in the enviable position for a modeller of having an extremely rich body of data against which the performance of his program can be evaluated (cf., Just & Carpenter, 1980). And, in fact, the program does quite reasonably; without special tuning of parameters, Thibadeau, Just, & Carpenter (1981) claim that READER accounts for 79% of the variance in their data, in contrast to the 72% accounted for by the model offered by Just & Carpenter (1980).

However, the principles embodied in the program are of even greater interest than its account of the data, because Thibadeau has done an especially impressive job of embedding his model of performance at a particular task within an information processing architecture of great potential generality. To see this, we need to look more closely at CAPS (Thibadeau, 1981), the interpreter for the language in which READER was implemented.

CAPS, which stands for "Collaborative Activation-based Production System", is a LISP interpreter for a language oriented towards concurrent processing of hypotheses at multiple levels. Its fundamental processing units are productions, independent condition-action rules. Its fundamental data objects are propositions, consisting of node-relation-node triples with an associated activation level. Activation represents the system's current confidence or certainty that the proposition is correct. The conditions of productions specify some set of propositions, along with threshold activation levels for each, below which the production will not be eligible for execution. CAPS executes all productions whose conditions are satisfied. Once a production becomes eligible for execution, it continues to fire on each processing cycle until some event occurs that causes it to stop. The primary action of a production is altering the activations of specific propositions by some proportion of the activation of one of the production's evoking propositions.

Figure 6 illustrates this by showing the general form of CAPS productions, and a hypothetical example paraphrased into English. The example can be paraphrased further as saying, "If you think you're seeing the letter T, but only if you think it's starting a new word,

Figure 6

GENERAL FORM OF CAPS PRODUCTIONS

(p *production-name*
(*propositions to send activation*
context in which to send
conditions for starting firing
conditions for stopping firing
-->
(<spew> *from sending propositions*
to target propositions
and side-effect propositions)))

EXAMPLE (PARAPHRASED INTO ENGLISH)

(p Letter-to-word
(*the letter seen was "T", activation 0.2 or greater*
the letter begins a new word; activation 0.3 or greater
the word seen is "THE", activation 0.01 or greater
the word seen is "THE", activation 0.999 or less
-->
(<spew> *from the letter seen was "T"*
to the word seen is "THE")))

and you also think that the word might be THE, then increase your certainty that the word in fact is THE by a proportion of your certainty that you've seen a T." Note that the conditions are specified in such a way that the production would begin to fire when the hypotheses first began to be entertained, and would stop firing when the target hypothesis is either accepted (activation greater than .999) or rejected (activation drops to zero).

In actual CAPS productions, the proportion of activation transmitted is specified in the production, but that proportion is actually a multiplier for a global parameter which can be adjusted by an action of productions called "<REWEIGHT>". This is one of a number of actions that allow the system to modify the rate at which activation flows from one hypothesis to another, along with thresholds for acceptance or rejection.

In short, Thibadeau has built not just a model of reading, but a very general processing language for implementing a large class of models based on a common theoretical framework. His work is a very nice example of how a concern with control processes and focus of attention can pay off.

4.3 Sloppy Errors: An Example Of Transfer To New Domains

There are many similarities between READER and McClelland & Rumelhart's model, and many complementary features as well. Thibadeau offers a model of parsing processes and a general control structure. McClelland and Rumelhart provide an analysis of interactions in the transmission of interaction under this sort of control structure -- namely, the "friends and enemies" effect, the "rich get richer" effect, and the

"gang" effect. They also offer some mechanisms for explaining how expectations come into play: context-dependent adjustments of weights on links between hypotheses at different levels. Thibadeau, in turn, provides in CAPS processing mechanisms such as <REWEIGHT> that make it possible to model those adjustment processes.

Together, they set the stage for a simulation of a seemingly very different topic, "sloppy" errors in algebra problem solving, which I am now working on in collaboration with James Greeno and Michael Ranney. Greeno has collected a large body of protocols illustrating a common and persistent problem. Novices make a large range of seemingly random errors, which they themselves can sometimes detect as errors if asked to review their own work. These errors occur with much greater frequency in novices than experts. It is not that the subjects have missing or incorrect rules for solving the problems, since they can identify their own errors. Nor is it that they have buggy rules (Brown & Burton, 1978), since they can identify the correct actions and since the errors do not consistently occur.

The model we are developing to account for these observations postulates an activation-based parsing process, like in Thibadeau's READER, that is trying to build an internal representation of an input algebra expression. The effects that McClelland & Rumelhart outlined can cause the system to mis-rate some of its hypotheses about the content of expressions. If one of the wrong hypotheses is accepted before the correct hypothesis has time to gain sufficient strength, an error will occur through the system applying correct algebra rules to incorrect data. In our model, learning to avoid errors has two components: learning the appropriate thresholds for accepting

hypotheses of various types, and learning the correct weights to be used in taking one hypothesis as supporting another.

What these examples illustrate is one of the most important properties of the simulation approach: the development of general concepts of information processing mechanisms. Regardless of the particular topic area, all simulation systems must solve the same problem: specification of control processes that will produce appropriate focus of attention. That is, whatever the program is to do, ensuring that it actually does it requires specifying mechanisms that will select appropriate actions in the proper sequence. Since all psychological simulations share the concern of modelling an intelligent system, general concepts about these control mechanisms may be developed which have applications in areas far removed from their origin.

5.0 LANGUAGES FOR PSYCHOLOGICAL SIMULATIONS

So far, I've been talking about some simulations of interest and trying to suggest some principles which they illustrate. At this point, I'd like to shift gears a bit and consider the languages in which simulations are implemented.

Although many different languages have been used to write simulation programs for psychology, historically the three most important are probably IPL, SNOBOL, and LISP. These are the languages which introduced the key concepts of list processing, pattern matching, and function notation.

It's worth quoting two sentences about IPL-5 from Sammet's (1969) review of programming languages, because they capture some critical points about the fate of many special-purpose languages. The first quote reads, "The most significant property of IPL-5 is that it has a closer notational resemblance to assembly language than any other language in this book..." The second quote brings some other sad news, "The implementation and development of this line of language stopped with IPL-5 because the people most vitally concerned were more interested in the problems they were trying to solve than in further language development."

It is these two factors, ease of use and certainty of support, that suggest why LISP caught on to a much greater extent then IPL. By and large, it has been such pragmatic factors that have influenced attempts to develop simulation languages especially for psychology. It would be a little grandiose to count the languages just mentioned as strictly psychological, since their development fell more within the bounds of AI and since they have also been put to use by other cognitive scientists (such as the MIT linguists whose work with COMIT led to the development of SNOBOL).

5.1 The First Generation Of Psychological Simulation Languages

Therefore, the first generation of specialized languages should probably be considered to have arrived in the early '70's with Newell"s (1973) PSG production system, Norman & Rumelhart's (1975) MEMOD interpreter for the language SOL, and Anderson's (1976) ACT model. These are all systems in which a number of specific simulations have been implemented, but where the system itself was an object of

psychological interest because it was seen as an analogy to at least some global aspects of the human information processing system. Newell emphasized event-driven processing and working memory limitations. Norman & Rumelhart emphasized long-term memory and the notion of "active semantic networks". Anderson's system tries to integrate all of these concerns. I will refer to all such systems as, "whole-system" simulations; it is important to distinguish them from "special-purpose" programs intended to simulate performance in a particular domain.

It is worth noting that, although their developers are still active in simulation work, all three of the systems just named have been phased out. Their developers seem to have turned, instead, to special-purpose programs designed to explore restricted aspects of verbally specified theories. Rumelhart's model of word perception was implemented in a program that did only that (McClelland & Rumelhart, 1981). Rumelhart & Norman (1981) have developed a complementary model of typing; again, implemented in a special-purpose program. Anderson has implemented some of his recent ideas about knowledge compilation as a learning mechanism (Neves & Anderson, 1981) not in his own ACTF program, but in a simpler production system architecture which retained only those features of ACT deemed immediately relevant to the task at hand.

Their new work is quite consistent with their old, so the abandonment of the whole-system simulations cannot be taken as a rejection of the theories. Rather, it seems more a question of practical matters. I'd like to speculate on a number of factors that lead researchers to abandon large systems.

- The systems become slow and expensive to run; there is a feeling that the cost is not justified when portions of the system are not directly related to the current topic of interest.

- The problems of developing and debugging a system grow as it increases in complexity; trained psychologists may prefer psychological research to hardcore computer science.

- Demand from others for chances to use the system are generally low; many researchers, even if they have the facilities to bring up the program at their own site, are hesitant to do so due to the theoretical unwillingness to buy an entire set of assumptions, and to the pragmatic fear of poor maintenance.

- At the same time, the demands of the few who are interested in adopting the system can become burdensome; one hesitates to commit the resources required for documenting and extending a system in order to make it usable outside the lab. (Norman and Rumelhart, who produced a manual for their MEMOD system running over 100 pages, are a notable exception to this remark.)

There are a number of advantages of pre-existing languages like LISP that make these difficulties seem especially discouraging. LISP is available on a wide range of machines in more-or-less compatible dialects (e.g., DEC KL-10s and 20s, VAXes, IBM 360's). With the exception of MIT's MACLISP variant, reasonably clear documentation is readily accessible. The language is fairly well-structured, symbol-oriented, and has many list processing and string manipulation constructs. It is relatively easy to define new data structures. Last, but by no means least, most variants of LISP offer fairly useful

interactive debugging and trace mechanisms.

Thus, it may seem that the trends favor small special-purpose simulation programs. However, to balance the picture, there are two points to consider. First of all, there are new whole-system simulations being developed. Thibadeau's (1981) CAPS and my own HPM (Neches, 1981ab) are two examples of such systems. Second, the way that CAPS and HPM were developed show that there are some benefits to the whole-system approach in terms of generality and understanding of unexpected inter-relations between components of the information processing system.

Although it may turn out that the CAPS and HPM efforts are subject to the same pitfalls as previous whole-system simulations, there is another system under development which attempts to steer a middle course between the alternatives of special-purpose modelling and whole-system simulation. That system is called PRISM, for Program for Research Into Self-Modifying systems, and is being developed by Pat Langley of Carnegie-Mellon University and myself (Langley & Neches, 1981).

5.2 The PRISM Production System Architecture

PRISM is a production system interpreter implemented by augmenting LISP with a number of special functions. It owes a major debt to Forgy's (1979) OPS4, from which a large portion of its code is borrowed.

Production system programs are more difficult to follow than traditional programs, because of their many conditional rules and the absence of an explicitly specified order of execution for the rules. This has probably been a major factor in limiting their acceptance. Nevertheless, there are a number of attractive properties to production systems, as Newell & Simon (1972, pages 804-806) and Langley, Neches, Neves, & Anzai (1980) have pointed out. They can model both goal-driven and data-driven processing, the program organization offers a closer analogy to human memory limitations than other programming formalisms, and the relative independence of individual production rules gives programs a degree of modifiability which might facilitate models of learning processes.

The design philosophy underlying PRISM is that there are too many unresolved questions about the details of how a production system should work. Thus, it is premature to fix a particular set of choices and try to impose them upon users. Instead, PRISM seeks to identify the key choice points in specifying a production system architecture, offer plausible options at those points, and make it easy for sophisticated users to implement alternatives to those options. Thus, rather than being a whole-system simulation of a particular information processing theory, PRISM defines a class of theories, and leaves it to the user to specify the details.

In order to do this, PRISM expands somewhat upon the traditional view of a production system as consisting of a data memory and a production memory, with productions being selected and applied in a repeating "recognize-act" cycle. Figure 7 shows the general structure of the PRISM system. Fixed components are shown as rectangles, those

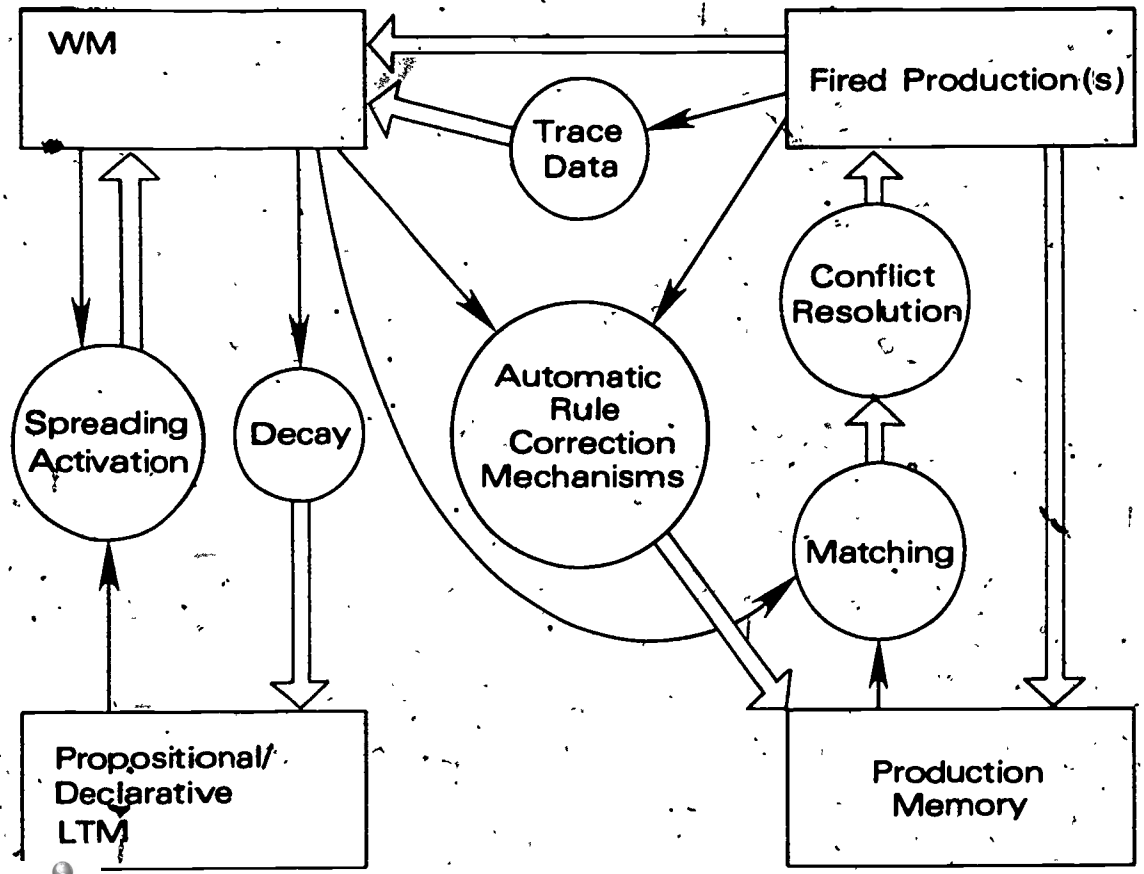

WM
Fired Production(s)
Trace Data
Conflict Resolution
Automatic Rule Correction Mechanisms
Spreading Activation
Decay
Matching
Propositional/ Declarative LTM
Production Memory

Figure 7

involving user-controlled options are shown as circles. Arrows indicate information flow.

For example, PRISM divides the process of modifying memory into three components: add-to-wm, which puts propositions into working memory for temporary storage; add-to-net, which puts propositions into long-term semantic memory; and, add-connections, which ties propositions to others in a way that permits activation to pass between them. Almost all operations performed by PRISM can be specified by the user to be either default actions (performed on all propositions asserted as the action of a production) or special-case actions performed only on the propositions explicitly specified as their arguments. Thus, the user has case-by-case control over how these operations are applied.

Once a proposition enters working memory, it becomes subject to policies selected by the user for determining how long it will reside there. Among other things, users select a decay function to be used in computing how activation will decrease over time, along with a threshold below which propositions will be treated as inactive.

As Figure 7 shows, data can enter active memory from several directions. In addition to explicit assertions of new data, old data may return to active memory via a process of spreading activation, or associative retrieval. We have seen several examples in this paper illustrating why this is a useful component of a model. However, the details in those examples differed enough for it to be clear why options are worthwhile. PRISM offers three options.

The "Spread-to-depth" option assumes that activation is sent out only from a subset of active nodes, and travels with decreasing strength to all nodes within a specified distance. The "Spread-to-limit" option also assumes that activation travels with decreasing strength from a subset of the active nodes, but allows the activation to travel from node to node until it drops below a threshold level. The third option permits directed activation schemes similar to Thibadeau's (1981). Like all PRISM options, it is relatively easy to implement alternatives to those supplied, since all that is required is to provide the name of a function which will be executed by PRISM on the list of propositions from which activation is to spread.

That list of propositions is determined by choices made by the user; as with other functions, the associative retrieval functions may either be called as explicit actions of productions or specified as default actions to be applied to all propositions asserted by productions.

PRISM can operate with a wide range of policies for selecting productions for execution, a process also known as "conflict resolution". This turns out to be one of the key points of difference between various production systems offered in the past. Anderson's (1976) ACT, for example, fired some productions in parallel, but not all of those eligible for execution. The complex restrictions imposed by the system involved assumptions about varying lengths of time required to select different productions, about generalized and specialized variants of productions, and so forth. Allen Newell (1980) offered a model of the human information processing system designed to account for some effects in speech perception, in which he claimed that

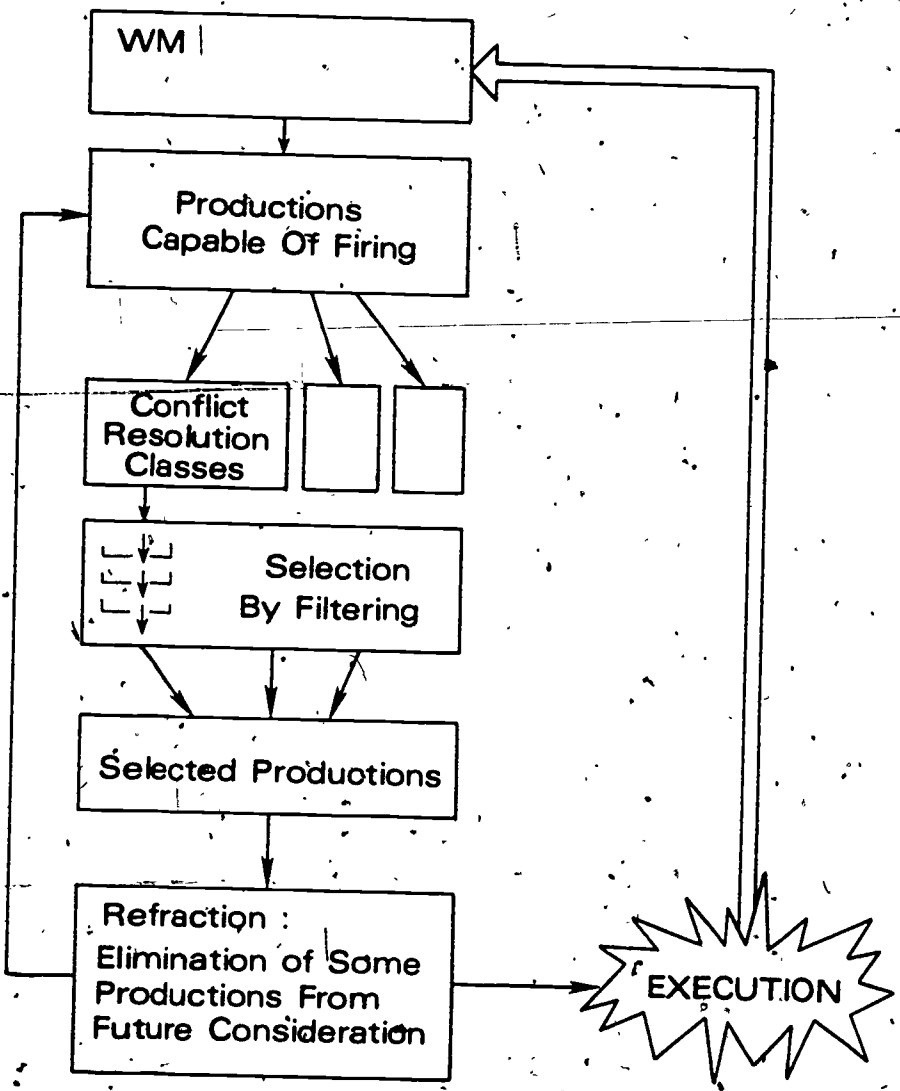
all satisfied productions containing constants could fire on a given cycle, but only one production involving variables in its conditions. Thibadeau's (1981) CAPS, on the other hand, allows all matched productions to fire. My own HPM (Neches, 1981a) divides productions into seven classes, with different rules for each class, and fires the union of the set of selections from each class.

PRISM's scheme for selecting productions for execution is shown in Figure 8. Like HPM, PRISM allows users to divide their set of production rules into independent classes which fire in parallel. In PRISM, users can specify one to infinity such classes, although the default is that all productions are placed in one common class. For each class that users allow, they define a "filter", or set of tests which must be passed for a production to be allowed to fire. Those productions passing the first test are sent on to the second, and so on. This allows the user to specify a wide range of conflict resolution policies.

PRISM also has a number of options related to modelling learning processes. In a production system, learning is mainly simulated by building new productions or by modifying pre-existing ones. (It is possible to also model learning in terms of changing or adding new declarative structures to long-term memory, of course, but there is no need to offer any special options in order for that to be done in PRISM.)

Note that the ability to model learning easily has long been a promise for production systems, ever since Newell & Simon (1972) started arguing for production systems as a formalism capturing key properties of the human information processing system. The argument

Figure 8


WM
Productions Capable Of Firing
Conflict Resolution Classes
Selection By Filtering
Selected Productions
Refraction : Elimination of Some Productions From Future Consideration
EXECUTION

has essentially been that learning models would be easier to implement than in traditional programming formalisms because of the modular properties of condition-action rules, with each production specifying the range of situations in which it's applicable, independent of all other productions (2). Up until quite recently, this promise was little more than just a promise. In the last few years, though, several different simulations have been developed in the formalism of self-modifying production systems (e.g., Anzai & Simon, 1979; Anderson, & Kline, 1979; Anderson, Kline, & Beasley, 1978; Langley, 1981; Neches, 1981ab; Neves, 1978; Neves & Anderson, 1981). The models which have been offered have incorporated several different features, and PRISM offers options related to each:

- Trace data: several learning models (e.g., Anzai & Simon, 1979; Langley, Neches, Neves, & Anzai, 1980; Neches, 1981ab) depend heavily on a system's memory for past actions. PRISM offers options that allow users to determine the form and content of the memory representation that is built after each production execution.

- Designation: since Waterman (1975), building new productions has been a staple feature of production system models of learning. PRISM contains a number of options governing the form of new productions constructed by pre-existing productions.

- Strengthening and weakening: PRISM offers options governing means for altering the likelihood of a particular production being

---

Footnote 2: This assumption puts a heavy burden on processes for selecting appropriate productions for firing, one reason why PRISM is designed with such a generalized view of conflict resolution.

selected for firing.

- Generalization: there are also options governing mechanisms for expanding a production's range of applicability through substitution of variables for constants in the production's conditions.

- Discrimination: there are a parallel set of options governing mechanisms for restricting a production's range of applicability through the insertion of additional conditions.

In summary, simulation work in PRISM starts with specifying a processing environment that controls how productions will be interpreted. The environment also includes long-term memory, active working memory, and processes which manage their contents, learning mechanisms. The system is built on top of LISP, and can therefore implement any knowledge representation which can be expressed as LISP data structures. PRISM can be thought of at two levels: either as a kit from which whole-system simulation packages can be assembled, or simply as a programming language which collects features found to have been convenient in other systems for cognitive simulations.

There are several motivations behind the development of the PRISM system. Production systems have been a useful simulation tool, but it is simply too early for any consensus to have arisen about the most useful form for a production system language to take. PRISM is intended to let researchers pick and choose the best combination of features for their particular purposes, without being forced to build a complete system from scratch. As I suggested in earlier sections, there is a strong gain from the exercise of trying to work within a

whole-system simulation. We hope that systems like PRISM, by encouraging researchers to specify whole systems, will promote a greater concern with the interactions between components -- that is, with the question of how the pieces of the puzzle are going to fit together. At the same time, PRISM's system of options, and the fact that it is built on top of a powerful programming language like LISP, are intended to make it relatively easy to modify and extend. This property of flexibility means, we hope, that models of particular tasks can be implemented within whole-system simulations without being forced into the Procrustean bed of a fixed system.

6.0 CONCLUSION

One of the most exciting things about simulation work is that, because of its necessary concern with control of processing and focus of attention issues, ideas can come out of a simulation project that are applicable in areas quite different from the domain in which the original work was done. I've tried to illustrate that point in the examples of simulation which I've presented. I have also tried to touch on a number of factors which are making simulation work easier and more accessible than ever before. One factor is the development of simulation languages, like CAPS and PRISM, which do not force their users to accept any single theory of the human information processing system, but provide frameworks in which models of the system, or components of the whole system -- can be developed and explored. Another factor is the development of lower cost machines, such as VAXes, with more powerful capabilities. A third factor is the increasing availability on these machines of core languages such as LISP, which facilitate direct implementation of special-purpose

simulations in addition to providing a foundation upon which simulation languages more specific to psychology can be constructed.

At the same time, though, I would like to avoid a presentation from the messianic genre. As we have seen, there are a number of advantages which have been claimed for the simulation approach that really do not hold up in actual practice. A computer simulation does not necessarily guarantee that a theory is more consistent or comprehensible. Nor does a program's successful performance guarantee that the theory is generalizable, or even that the causes for the success are those predicted by the theory. The psychological significance of a computer program can only be determined by close and careful examination of each piece of work on a case-by-case basis. There are also some practical limitations which will limit the spread of simulation work for some time to come. It is still time-consuming and hard to delegate. Interesting projects often have many of their payoffs only at the end, with fewer publishable milestones along the way. Computer hardware and software facilities are not always being planned with the potential for simulation work in mind.

These difficulties are due in part to the fact that the promise of simulation methodology -- the different levels at which it can stimulate thought about psychological issues -- is not as widely appreciated as it could be. I have tried in this paper to illustrate some of the ways in which simulations can aid us in thinking and reasoning about the human mind. They provide a tool for empirically analyzing theories to better understand their implications and predictions. They are a means of exploring interactions between components of complex models. They pose a practical challenge to

operationalize theoretical constructs, which can lead to incidental discoveries about related processes. And, finally, they engender a concern with issues of process control that contributes to the development of general principles with broad applications.

References

Anderson, J.R. Arguments concerning representations for mental imagery. Psychological Review, 1978, 85, 249-277.

Anderson, J.R. Language, memory, and thought. Hillsdale, NJ: Lawrence Erlbaum Associates, 1976.

Anderson, J.R., & Kline, P.J. A learning system and its psychological implications. In Proceedings of the Sixth International Joint Conference on Artificial Intelligence, 1979.

Anderson, J.R., Kline, P.J., & Beasley, C.M. A theory of the acquisition of cognitive skills (TR 77-1). New Haven: Psychology Department, Yale University, 1978.

Anzai, Y., & Simon, H.A. The theory of learning by doing. Psychological Review, 1979, 86, 124-140.

Bobrow, D.G., & Norman, D.A. Some principles of memory schemata. In D.G. Bobrow & A. Collins (Eds.), Representation and understanding: studies in cognitive science. New York: Academic Press, 1975.

Brown, J.S., & Burton, R.R. Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, 1978, 2, 155-192.

Collins, A.M., & Loftus, E.P. A spreading activation theory of semantic processing. Psychological Review, 1975, 82, 407-428.

Erman, L.R., & Lesser, V.R. A multi-level organization for problem-solving using many diverse co-operating sources of knowledge. In Proceedings of the Fourth International Joint Conference on Artificial Intelligence, 1975, 483-490.

Ernst, G.W., & Newell, A. GPS: a case study in generality and problem solving. NY: Academic Press, 1969.

Feigenbaum, E.A. The simulation of verbal learning behavior. In Proceedings of the Western Joint Computer Conference, 1961, 121-132.

Forgy, C.L. The OPS4 reference manual. Pittsburgh, PA: Computer Science Department, Carnegie-Mellon University, 1979.

Gregg, L.W., & Simon, H.A. Process models and stochastic theories of simple concept formation. Journal of Mathematical Psychology, 1967, 4, 246-276.

Hanna, F.K., & Ritchie, G.D. AM: a case study in A.I. methodology. Canterbury, CT2 7NT, Great Britain: Electronics Laboratories, University of Kent, undated.

Hayes-Roth, F. Distinguishing theories of representation: a critique of Anderson's "Arguments concerning representations for mental imagery". Psychological Review, 1979, 86, 376-382.

Hayes-Roth, B., & Hayes-Roth, F. A cognitive model of planning. Cognitive Science, 1979, 3, 275-310.

Just, M.A., & Carpenter, P.A. A theory of reading: from eye fixations to comprehension. Psychological Review, 1980, 87, 329-354.

Kintsch, W. The representation of meaning in memory. Hillsdale, NJ: Lawrence Erlbaum Associates, 1974.

Langley, P. Language acquisition through error recovery (CIP 432). Pittsburgh, PA: Psychology Department, Carnegie-Mellon University, 1981.

Langley, P., & Neches, R. PRISM reference manual. Pittsburgh, PA: Computer Science Department, Carnegie-Mellon University, 1981.

Langley, P., Neches, R., Neves, D., & Anzai, Y. A domain-independent framework for procedure learning. Policy Analysis and Information Systems, special issue on Knowledge Acquisition and Induction, 1980, 4, 163-197.

Lenat, D.B. An artificial intelligence approach to discovery in mathematics as heuristic search (Memo AIM-286). Stanford, CA: Computer Science Department, Stanford University, 1976.

Lenat, D.B. Automated theory formation in mathematics. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 1977, 833-842.

Lewin, K. A dynamic theory of personality. NY: McGraw-Hill, 1935.

Mandler, J.M., & Johnson, N.S. Remembrance of things parsed: story structure and recall. Cognitive Psychology, 1977, 9, 111-151.

McClelland, J.L., & Rumelhart, D.E. An interactive activation model of context effects in letter perception: Part I. An account of basic findings. Psychological Review, 1981, 88, 375-407.

Miller, L. Has AI contributed to our understanding of the human mind? A critique of the arguments for and against. Cognitive Science, 1978, 2, 111-128.

Minsky, M. A framework for representing knowledge. In P.H. Winston (Ed.), The psychology of computer vision. NY: McGraw-Hill, 1975.

Neches, R. HPM: a computational formalism for heuristic procedure modification. In Proceedings of the Seventh International Joint Conference on Artificial Intelligence, 1981a, 283-288.

Neches, R. Models of heuristic procedure modification. Pittsburgh, PA: Psychology Department, Carnegie-Mellon University, unpublished Ph.D. thesis, 1981b.

Neves, D.M. A computer program that learns algebraic procedures. In Proceedings of the Second Conference of the Canadian Society for Computational Studies of Intelligence, 1978.

Neves, D.M. & Anderson, J.R. Knowledge compilation: mechanisms for automatization of cognitive skills. In J.R. Anderson (Ed.), Cognitive skills and their acquisition. Hillsdale, NJ, Lawrence Erlbaum Associates, 1981.

Newell, A. HARPY, production systems, and human cognition. In R. Cole (Ed.), Perception and production of fluent speech. Hillsdale, NJ: Lawrence Erlbaum Associates, 1980.

Newell, A. Production systems: models of control structures. In W.G. Chase (Ed.), Visual information processing. NY: Academic Press, 1973.

Newell, A., & Simon, H.A. Human problem solving. Englewood Cliffs, NJ: Prentice-Hall, 1972.

Norman, D.A., Rumelhart, D.E., & the LNR Research Group, Explorations in cognition. San Francisco: W.H. Freeman & Co., 1975.

Popper, K.R. The logic of scientific discovery. London: Hutchinson, 1959.

Rumelhart, D.E. Notes on a schema for stories. In D.G. Bobrow & A. Collins (Eds.), Representation and understanding: studies in cognitive science. NY: Academic Press, 1975.

Rumelhart, D.E. Toward an interactive model of reading. In S. Dornic (Ed.), Attention and performance VI. Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.

Rumelhart, D.E., & Norman, D.A. Simulating a skilled typist: a study of skilled cognitive-motor performance (CHIP 102). La Jolla, CA: Center for Human Information Processing, University of California at San Diego, 1981.

Sacerdoti, E.D. A structure for plans and behavior. NY: American Elsevier Press, 1977.

Sammet, J.E. Programming languages: history and funddmentals. Englewood Cliffs, NJ: Prentice-Hall, 1969.

Schank, R., & Abelson, R. Scripts, plans, goals, and understanding. Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.

Simon, H.A., & Feigenbaum, E.A. An information-processing theory of some effects of similarity, familiarization, and meaningfulness in verbal learning. Journal of Verbal Learning and Verbal Behavior, 1964, 3, 385-396.

Thibadeau, R. CAPS manual. Pittsburgh, PA: Psychology Department, Carnegie-Mellon University, 1981.

Thibadeau, R., Just, M.A., & Carpenter, P.A. A model of the time course and content of human reading. Pittsburgh, PA: Psychology Department, Carnegie-Mellon University, manuscript submitted for publication, 1981.

Thorndyke, P.W. Cognitive structures in comprehension and memory of narrative descriptions. Cognitive Psychology, 1977, 9, 141-191.

Waterman, D.A. Adaptive production systems. In Proceedings of the Fourth International Joint Conference on Artificial Intelligence, 1975.

Winograd, T. Understanding natural language. Cognitive Psychology, 1972, 3, 1-191.

Young, R.M. & O'Shea, T. Errors in children's subtraction. Cognitive Science, 1981, 5, 153-177.