A Theory of the Origins of Human Knowledge

John R. Anderson
Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA 15231, U.S.A.

ABSTRACT

The PUPS theory and its ACT predecessor are computational embodiments of psychology's effort to develop a theory of the origins of knowledge. The theories contain proposals for extraction of knowledge from the environment, a strength-based prioritization of knowledge, knowledge compilation mechanisms for forming use-specific versions of knowledge, and induction mechanisms for extending knowledge. PUPS differs from ACT* basically in its principles of induction which include analogy-based generalization, a discrimination mechanism, and principles of making causal inferences. The knowledge in these theories can be classified into the knowledge level, algorithm level, and implementation level. Knowledge at the knowledge level consists of information acquired from the environment and innate principles of induction and problem solving. Knowledge at the algorithm level consists of internal deductions, inductions, and compilation. Knowledge at the implementation level takes the form of setting strengths for the encoding of specific pieces of information*

1. Introduction

This paper is an attempt to establish a perspective on the sequence of theories I have been associated with. Anderson and Bower [9] wrote a book about a system called HAM which was a theory of human declarative knowledge. This was succeeded in 1976 by ACT [2] which augmented the declarative knowledge component with a procedural component in the form of a production system. In 1983 [5] ACT evolved into ACT* which had an elaborate theory of the acquisition of production rules. More recently, Anderson and Thompson [12] have developed the PUPS system which is an attempt to remedy deficits in the ACT* theory.

These theories have been iterations in an attempt to evolve a theory of the origins of human knowledge. Before going into the claims of these theories it is important to establish the larger historical perspective in which this research takes place. Therefore, I will review the significant issues in the psychological attempts to develop a theory of the origins of human knowledge. The major factor that has caused our work to develop from these historical roots is the emphasis that our theories have a firm computational foundation.

0004-3702/89/$3.50 © 1989, Elsevier Science Publishers B.V. (North-Holland)

After this historical perspective there will be an analysis of ACT* and PUPS. Finally, I will try to identify where these theories stand on the issue of the origins of knowledge. Despite the considerable technical work that has been done on the theories we have never made them face up to this fundamental issue.

2. Origins of Knowledge: A Psychological Perspective

2.1. Philosophical origins

Of course, the issue of the origins of human knowledge has been an issue of philosophical debate for centuries. While the philosophical subtleties were many, there were essentially three positions about the origins of any piece of knowledge:

(1) *Nativism*. The person was born with that piece of knowledge or it appeared in the mind according to some predetermined maturational process. A common candidate for innate knowledge is our knowledge about the causal structure of the world. This is the claim that we are born knowing that there are causes and effects in the world even if we have to learn what causes what. Another frequent suggestion in cognitive science is that there is innate knowledge about the syntax of natural language (Chomsky [18]).

(2) *Empiricism*. The knowledge was planted in the mind by experience. A common candidate for such knowledge is our knowledge about what words mean. These word meanings are arbitrary associations we have to commit to memory.

(3) *Rationalism*. The knowledge originated by the person engaging in some reasoning process. More generally, we may think of rationalism as the position that knowledge appears as the result of internal computation, without making the commitment that this computation deserves the categorization of "reasoning." Mathematical knowledge is an example of something that is often thought to arise this way.

These three categories presumably exhaust the plausible sources of knowledge. Therefore, one litmus test for the adequacy of any theory of human knowledge is that it be able to categorize knowledge into these three categories. If it cannot it is not coming to grips with the issue of the origins of knowledge. Philosophers, being philosophers, were inclined to argue that all or most knowledge was of one kind or another. However, even if one takes a less extreme position, the above provides a useful categorization of possible origins of knowledge and one can certainly argue about the origins of any particular piece of knowledge. Fundamentally, such an issue is a scientific issue and not one to be settled by introspection and logical argumentation.

2.2. Behaviorism and a science of human knowledge

Psychology started out in the late 1800s as a science with one of its main goals to settle empirically the issue of the origins of knowledge. The field had a great deal of difficulty initially in making any headway on the issue. The problem is that scientists need to have agreed-upon data and it took psychology a while to establish consensus about what its data were. The introspectionists (see Boring [14]) took as their data self-observations of the content of thought. The problem was that different researchers with different theories would observe different things about their internal thoughts that confirmed their different theories. For instance, some introspectionists claim they could have thought devoid of sensory content while others claimed they could not. Because of irresolvable controversies like this, it became clear that a more objective data source was required.

This was one of the stimuli for the behaviorist movement that began around 1920, a movement much misunderstood, particularly by many of the behaviorists who practiced it. The behaviorist movement began with the observation that the prime source of objective data was recordings of the behavior of people. There were other possible sources of objective data such as physiological recordings but these turn out to be much harder to obtain than behavior. There are two essential features that distinguish behavioral data from introspective data. The first is that it is equally available to all scientists and not the private domain of the scientist who is having the introspection. Second, the psychologist is not constrained as to how to theoretically interpret the data. Thus, if a subject of a psychological experiment says "I have a visual image of a cat", the scientist is free to propose any theory that will produce that verbal protocol and is not required to propose visual images as part of the theory. Because of the similarity of verbal protocols to introspective reports, many behaviorists have refused to admit verbal protocol data. However, as Ericsson and Simon [25] correctly argue, verbal protocols are very appropriate and powerful sorts of behavioral data, when treated as behavioral data and not as introspective data.

However, there was a second point of motivation in stressing behavior as the measure by which a theory of knowledge will be assessed. This arose out of an emphasis on the functional nature of human knowledge. There was no point in making distinctions about knowledge that did not have consequences for behavior. If the person behaved the same whether he possessed knowledge X or not, in what sense does he really know X? If two pieces of knowledge result in the same behavior in what sense are they really different? Such arguments should be quite familiar to the AI community where we commonly talk about equivalence among different knowledge representation schemes. This point of view also anticipates the Turing-type tests for deciding if a system is intelligent.

The behaviorists frequently argued that there was no such thing as knowledge in the abstract; when we speak of someone having certain knowledge we mean that the person has certain behavioral potentials. This led to prohibitions against discussing mental structures and a claim that an objective science should only talk about behavior. Here we see a basically correct observation being carried to unfortunate extremes.

Behaviorism can be separated into a methodological and a theoretical position:

(1) *Methodological*. Behavioral data is the major data for deciding among theories of knowledge. Different theories that imply no difference for the behavior (including verbal) of the system might as well be regarded as notational variants.

(2) *Theoretical*. The terms of a theory should be behavioral. Since only external behavior counts, a theory should not make reference to underlying mental structures.

The fundamental error in behaviorism comes from extending the methodological prescription to the theoretical prohibition. Given that they were prohibited from theorizing about mental structure it is not surprising that the behaviorist theories tended to take strong empiricist stands and claim all knowledge arose directly from experience. A methodology that denied the existence of a mind denied the possibility of a contribution of the mind to knowledge.

The behaviorists were very fundamentally concerned with issues of learning. The first half-century of American psychology was the era of the grand learning theories (see Bower and Hilgard [15] for a review of these). There were many variations but they all proposed that experience implanted various connections among stimuli and responses which would play themselves out as adaptive behavior under appropriate circumstances. Many of the issues that separated the various theories were really concerned with how motivational variables influenced the acquisition of knowledge and its later appropriate execution.

The behaviorist learning theories of the first half-century, while they were rigorous in their choice of data, were fundamentally sloppy in their theorizing. They never really showed that the mechanisms of their theories could be put together to account for the complexities of human behavior. The advent of more computationally oriented theories in the 1950s provide the basis for exploring the issues of what could actually be computed in these frameworks. It became clear that there were serious inadequacies. Chomsky's [17] criticism of Skinner's [55] account of language was the most dramatic instance of such analysis, but there were many others. The learning theorists had insisted that they could account for behavior with theories whose only terms were objectively observable stimuli and responses. It became apparent that one needed to make reference to non-observable mechanisms to account for human behavior.

2.3. Cognitive psychology and computational concerns

Cognitive psychology as a field of endeavor really began with the demise of behaviorism. The development in cognitive psychology has closely paralleled work in artificial intelligence, both contributing to that work and borrowing from it. One of the major contributions of AI work was to provide demonstrations that theories of internal structure and process could be scientifically rigorous. This was constantly used to fend off the behaviorist critics of cognitive psychology.

In the 1960s and 1970s there appeared in cognitive psychology a set of theories that were sufficiently well specified that they could be simulated on a computer. Sometimes these theories even were. These theories tended to focus on the performance of acquired skills. This nonlearning emphasis may have arisen because of the perceived difficulty of addressing learning and because some of the most telling criticisms of behavioristic theories were focused on their inability to perform at adult-level competences, not on the issue of whether these competences could be acquired. The standard remark of the time was that we needed first to understand the system that learning produced before we could try to understand how learning worked. Of course, a similar attitude existed in artificial intelligence about learning research.

The emphasis in psychology became not what we know and how we acquire it but rather how we implement what we know. One of the premier issues of the time was whether various computations are performed in parallel or serial. One domain where this was debated was the Sternberg task [56]. In a prototypical experiment a subject was first told to keep in mind a set of one to six digits and then asked whether a particular probe digit was in the memory set. It is generally found that subjects take about 40 milliseconds longer to make this judgment for each digit that is in the set. The serial model for this task proposed that subjects serially matched the probe digit against each item in the memory set taking about 40 milliseconds to make each comparison. The parallel model for this task proposed that the digit was compared in parallel against all members of the memory set but that there was a fixed capacity for this parallel computation and the more comparisons were being performed the slower any particular comparison was made. It was finally shown that in general such parallel and serial models could be made mathematically equivalent (Townsend [57]). Researchers were forced to make their arguments on vaguer claims such as whether the brain really could perform serial operations at the rate of one per 40 milliseconds (J.A. Anderson [1]).

Another example of a research issue of this type concerned the existence of a separate short-term memory. It was observed that when we are told something we remember it well initially but rapidly forget much of it. So, for instance, we can remember a phone number for a few seconds but an hour later we are lucky to remember any of the number. One model (Atkinson and Shiffrin [13]) proposed that there was an initial short-term memory in which information could be held for a while. However, if the information decayed from short-term memory or was pushed out by interfering information, it would be lost unless it was transferred to long-term memory. The contrary position (e.g., Wickelgren [59]) was that there was just a single long-term memory and this initial rapid decay of information was continuous with the general forgetting curve for long-term memory which is characterized by being quite negatively accelerated. Again, the field reasoned to the conclusion that these positions were essentially indistinguishable (Crowder [20]).

I think these two episodes and many others like them were symptomatic of the fact that the field was theorizing at too concrete a level. It was trying to resolve distinctions which fundamentally could not be resolved on the basis of behavioral data. As a further observation, even if these issues could have been resolved, their resolution would have shed no light on fundamental issues of the origin of human knowledge which is our concern in this paper if not always in cognitive psychology. For instance, these issues said nothing about the nativist-empiricist-rationalist controversy we highlighted at the beginning of this review.

Establishing computational rigor was a major methodological contribution of this era in cognitive psychology. The major theoretical contribution of this era to the issues of the origin of knowledge was the work of Newell and Simon [46] providing a coherent analysis of human problem solving. The basic concepts of this analysis are extremely familiar to the AI community and do not need to be repeated here. However, the essential feature from the point of view of psychology is that it defined a mapping from goals and knowledge to behavior. This provided the kind of computational basis for a theory of human motivation that had not been available to the learning theorists. It was now possible to rigorously address issues about the functional character of human knowledge.

2.4. Desiderata for a theory of human knowledge

From this brief survey of how psychology has got to where it is today we see a number of criteria by which to judge a psychological theory if it is to address issues of the origin of knowledge.

(1) That it in fact address the issue of the origin of knowledge. A litmus test of this is that the theory have something to say about the nativist-empiricist-rationalist controversy. This need not mean that it choose among the positions, but it should at least to able to say why a choice would not be meaningful.

(2) That it be cast fundamentally as a scientific theory with testable predictions about human behavior.

(3) That it address not just knowledge in the abstract but the functional consequences of the knowledge for human behavior.

(4) That it be cast with the precision that it can be a runnable simulation of human behavior.

(5) That it be capable of being cast at a level of abstraction that ignores irrelevant and often undecidable issues about how that knowledge is actually implemented in the human head.

The series of theories that I have been associated with were motivated by this agenda, although I must admit that I have only recently articulated to myself the last criterion above. Certainly, my theorizing has not always satisfied the last criterion as will be apparent from the review of that theory to follow. Nonetheless, I will try to focus on the significant abstract claims of the theory.

3. The ACT Learning Theory

The goal of this paper is to analyze the knowledge acquisition processes in the PUPS theory (Anderson and Thompson [12]), which is currently our best model of human knowledge acquisition. However, PUPS really is just a revision and embellishment of the ACT theory. Therefore, the next section of the paper will present a development of the ACT theory and the subsequent section will note the modifications involved in the PUPS theory.

3.1. The ACT architecture

Figure 1 illustrates the basic ACT production system architecture as developed by Anderson [5]. There are three memories: a working memory, a production memory, and, a declarative memory. Production rules had their conditions matched to the contents of working memory and their actions could add to the contents of working memory. This is the standard production system interpretive cycle of the variety proposed by Newell [44]. Where the ACT series differs

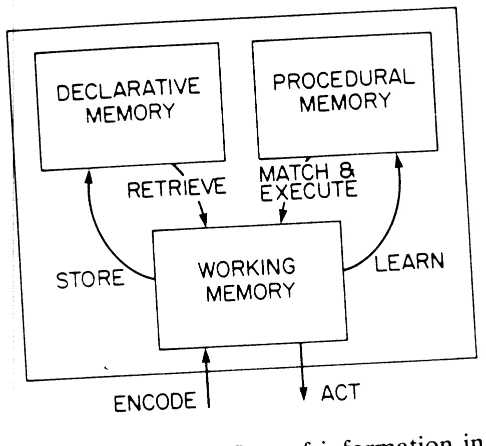

Fig. 1. A representation of the general flow of information in the ACT* architecture.

from Newell is in the postulation of a separate declarative database. Declarative memory consisted of factual knowledge not committed to a particular use. Stored in declarative memory might be an abstract fact of the form, "Two angles whose measure sum to 180 degrees are supplementary angles."

In contrast, in production memory knowledge is encoded in a use-specific way. For instance, we might have the following production rule which is derived from the above definition:

IF one wants to prove that the measure of angle 1 equals the measure of angle 2
and angle 1 is supplementary to angle 3,
THEN try to prove that angle 2 is supplementary to angle 3.

The procedural-declarative distinction has not had a happy history in artificial intelligence. Therefore, it is important to emphasize that there is a substantial psychological literature pointing to the existence of the two kinds of knowledge in the human case. It also has a somewhat different definition in psychology than artificial intelligence. For psychologists procedural knowledge is distinguished from declarative knowledge by the criteria that it is not reportable and that it is available in a use-specific form rather than in a general form. A number of experiments have been done showing that people can acquire knowledge in procedural form but not declaratively. For instance, Cohen [19] has shown that certain amnesic patients can learn to solve the Tower of Hanoi problem but have no reportable memory of seeing the problem.

As Fig. 1 illustrates, all knowledge from the environment comes into working memory first. The assumption is that various perceptual systems leave descriptions of experience in working memory. A permanent residue of these experiences can be deposited into declarative memory. These memories can be brought back into working memory by a spreading activation process by which associated concepts retrieve these facts. For example, the supplementary angle definition might be brought back into working memory at a later date if the student comes upon a problem that mentions supplementary angles. Thus, working memory contains both encoding from the immediate environment and associated past memories. The contents of working memory result in behavior by means of being matched to the condition side of production rules. Productions can, in their action sides, add goals and facts to working memory. These facts can themselves be stored in long-term memory. We thus remember our past inferences.

An important feature of this architecture is that there is no way by which experience can directly create new productions. Productions are created by a learning system inspecting the trace of past production firings and creating new ones. Thus, procedural learning is a matter of learning by doing.

Another important feature of the ACT architecture was its use of activation-based computations to implement the information processing implied by this architecture. While the details would be a paper in themselves, Anderson [5] went through a concerted effort to establish that the symbolic processing involved in the theory could be implemented in terms that were plausible given what is known about the nature of brain computations on neural activation levels. Working memory is actually the portion of the knowledge structure currently active. Activation has a rapid decay process so that without some source of maintenance any particular piece of knowledge would rapidly decay. Spreading activation, the retrieval process, was a mechanism for making active the related portions of declarative memory. The process of matching production patterns and selecting from among related productions (i.e. conflict resolution) was implemented by excitatory and inhibitory computations taking place on the production rule conditions.

Given this architecture, learning will involve either changes to declarative memory or production memory. There are four basic kinds of learning mechanisms in ACT. We will review them in rough order of increasing complexity.

3.2. Declarative recording

Information from the environment is deposited in a declarative form in working memory. Declarative information in working memory may be permanently recorded in a long-term memory. Once in long-term memory the spreading activation process can retrieve the information for later use. In the ACT* theory the recording was a probabilistic process in which information in working memory would only sometimes be permanently recorded. See [5] for a review of the psychological literature that points to such a simple probabilistic encoding process.

This is a particularly simple process but absolutely key to ACT's theory of knowledge acquisition as developed by Anderson [5]. It provides the empiricist component of the theory in that this is the only way for knowledge to enter from the outside into the system.

3.3. Strengthening

Both in declarative and production memory, knowledge is stengthened every time it is used. Strengthening a piece of knowledge will make it more likely to be selected at a later point in time. A production is considered to be used whenever it is selected by conflict resolution and fired. A declarative fact is considered to be used whenever it is matched to the condition of a production rule that fires. As developed in [5] these strengths exert their influence primarily through the conflict resolution process, which decides which instantiations of productions to fire. Strengthening is an interesting case of learning in that it in no way changes the knowledge that is encoded in the system but does change what knowledge will actually be manifested in behavior. Thus, the strengthening mechanism is basically a means by which the system concentrates processing resources on knowledge that is frequently used.

There is also a decay process that weakens the strength of knowledge if it is not used for a while. The formula for the strength of a particular item is:

$$S = \sum_{i=1}^{n} t_i^{-d} ,$$

where the summation is over the n uses of the item and each t_i is the time since that use of the item. Basically we have a sum of a set of strengthenings each of which is decaying as a power function (d is the exponent) of time. This is a particularly straightforward process and one for which there is an abundance of psychological evidence (for a review read [5]). What is interesting is the amount of "wisdom" built into this equation about what knowledge is likely to prove useful.

(1) The equation has a strong recency component built into it in that it effectively puts recent events into a "cache memory" from which they can be retrieved.

(2) The equation uses frequency of occurrence information and prefers knowledge that has proven useful multiple times in the past.

(3) The equation lets long-term tendencies dominate in the long term. If there are two pieces of knowledge currently equal in strength but one whose strength is based on a recent accumulation of strength and the other whose strength was built up over a longer period, then in the future the more established piece of knowledge will be preferred because its strength is decaying less rapidly.

Anderson [6] has shown that this strengthening function can be derived as an optimal solution to the information retrieval demands facing the human.

3.4. Knowledge compilation

In ACT an extended computation can be replaced by a single production rule. Often this new production rule will have built into it knowledge that had been only in declarative memory before. To return to the supplementary angle example given earlier, the definition of supplementary angle might have been used by some general inference productions to come to the conclusion that it would be a good idea to try to prove angles 2 and 3 supplementary. The production given above would then be produced by the knowledge compilation process as a summary of this process. Knowledge compilation is thus the way knowledge can transform declarative to procedural knowledge.

In defining more precisely the knowledge compilation process it is necessary to distinguish between two types of compiling mechanisms, one of which is called composition and the other proceduralization. The description below describes these two components of knowledge compilation as they are implemented in the GRAPES production system (Sauers and Farrell [53]), which is intended to embody certain aspects of the ACT* theory.

3.4.1. *Proceduralization*

Proceduralization assumes a separation between goal information and context information in the condition of a production. Consider the following production:

```
G1    IF the goal is to create a structure
          and there is an operation that creates such a structure,
      THEN use that operation.
```

This is a classic working-backwards operator, an instance of a general, weak, problem solving method. This production might apply if our goal was to insert an element into a list and we knew that there was a LISP function, CONS, that achieved this goal. In this production, the first line of the condition describes the current goal and the second context line identifies relevant information in declarative memory. Proceduralization eliminates the context lines but gets their effect by building a more specific goal description:

```
D1    IF the goal is to insert an element into a list,
      THEN use CONS.
```

The transition from the first production to the second is an example of the transition from a domain-general to a domain-specific production.

To understand in more detail how domain-general productions apply and how proceduralization occurs, one needs to be more precise about the encoding of the production, the goal, and the knowledge about CONS. With respect to the production, we have to identify its variable components. Below is a production more like its GRAPES implementation, where terms prefixed by "=" denote variables:

```
G1'   IF the goal is to achieve =relation on =arg1 and =arg2
          and =operation achieves =relation on =term1 and =term2,
      THEN use =operation.
```

When this production applies in the CONS case, the goal is "to achieve insertion of arg1 into arg2," and our knowledge about CONS is represented as "CONS achieves insertion of argument1 into argument2." The production G1' applies to the situation with the following binding of variables:

=relation:	insertion
=operation:	CONS
=arg1:	arg1
=arg2:	arg2
=term1:	argument1
=term2:	argument2

The actual execution of the production requires that the definition of CONS be held active in working memory and be matched by the production. This can be eliminated by proceduralization, which builds a new production that has embedded in it the information that was retrieved by matching to the CONS definition. This is achieved by eliminating the parts of the production that matched to the definition (i.e., =operation achieves =relation on =term1 and =term2) and replaces the variables from this part by their bindings (i.e., =relation gets replaced by *insertion* and =operation by CONS). The proceduralized production that would be built in this case is:

D1′ IF the goal is to achieve insertion of =arg1 into =arg2,
THEN use CONS.

In general, proceduralization operates by eliminating reference to the declarative knowledge of the domain used for problem solution by the weak method productions and building the consequences of the knowledge into domain-specific production rules. In a production system architecture like GRAPES, where this access to the declarative knowledge only occurs through the matching of production conditions, the variable replacement description scheme above is adequate to implement proceduralization. When we talk later about the analogy-based processing in PUPS, it will be necessary to perform different computations to decide how to eliminate reliance on working memory structures. The general concept of proceduralization is to be understood in terms of the elimination of the reliance on declarative, working memory elements.

In ACT* proceduralization would only eliminate matches to declarative knowledge that was permanently encoded in long-term memory. The claim was that, since the knowledge was there, all that proceduralization did was eliminate the need for its retrieval. Proceduralization did not fundamentally change the behavior of the system.

Since 1983 there has been a fair amount of data (e.g., [19, 29, 47]) which suggests that procedural knowledge can be acquired without there being first a long-term declarative representation. The most dramatic data comes from patients who display poor long-term declarative memory for events after the establishment of some neural impairment. Many of these are Korsokof patients who suffer effects of severe alcoholism. However, one of the most impressive patients is HM who had surgical removal of part of his hippocampus. While he can remember events from his childhood, and early adulthood, he has total amnesia for any events after the operation. This patient has been shown nonetheless capable of learning to solve problems such as the Tower of Hanoi puzzle. When presented with the puzzle he will swear he has never seen it before but nonetheless proceeds to solve it perfectly (which he could not do initially). While all these patients do not have such complete declarative impairment many have shown an ability to acquire skills from practicing a task while they cannot remember their experiences on the task.

While these patients have impaired long-term memories they have perfectly functional short-term declarative memories. That is, while solving the problem they can remember the instructions. The memory deficit only shows up when they return to the task. This suggests that ACT* was wrong in the claim that long-term declarative memory was a prerequisite to procedural knowledge. Any declarative representation, even if it is only transient, seems adeauate. This is the implementation of proceduralization within PUPS.

3.4.2. *Composition*

Composition (Lewis [35]) is the process of collapsing multiple productions into single productions. Whenever a sequence of productions apply in ACT and achieve a goal, a single production can be formed that will achieve the effect of the set.

The original Lewis proposal was to collapse pairs of productions that occurred in sequence. A problem with this is that often contiguous productions have little to do with each other. We use the goal structures in ACT* to decide what productions to compose. Basically, ACT composes productions that generated a sequence of subgoals and actions that led to the satisfaction of a particular goal.

While many times composition applies to sequences of more than two productions, its effect on longer sequences is just the concatenation of its effect on shorter sequences. Thus, if S1, S2 is a sequence of productions to be composed and C is the composition operator, C(S1,S2) = C(C(S1), C(S2)). So all we have to do is specify the pairwise compositions.

Let "IF C1 THEN A1" and "IF C2 THEN A2" be a pair of productions to be composed, where C1 and C2 are conditions and A1 and A2 are actions. Then their composition is "IF C1 & (C2 − A1) THEN (A1 − G(C2)) & A2." C2 − A1 denotes the conditions of the second production not satisfied by structures created in the action of the first. All the conditional tests in C1 and (C1 − A2) must be present from the beginning if the pair of productions are to fire. A1 − G(C2) denotes the actions of the first production minus the goals created by the first production that were satisfied by the second.

As an example, consider a situation where we want to insert the first element of one list into a second list. One production would fire and write CONS, setting subgoals to code the two arguments to CONS. Then a second production would fire to code CAR and set a subgoal to code the argument to CAR. Composing these two together would produce the production:

C3 IF the goal is to insert the first element of =arg2 into =arg3,
THEN code CONS and then code CAR and set as subgoals to
(1) code =arg2
(2) code =arg3.

Composition differs from chunking as developed in SOAR (Laird, Rosenbloom and Newell [34]) in that it is defined on the productions involved whereas the SOAR mechanism is defined on the working memory elements involved. In SOAR the same variable is introduced for the same working memory token wherever it appears in the chunk, leading to certain accidental restrictions on the production created because the same variable appears unnecessarily in the same place. Composition, being defined on the production rules themselves can avoid such accidental coincidences of variable bindings. Explanation-based generalization also contrasts with SOAR in this respect (Rosenbloom and Laird [50]).

The weakness of the SOAR variabilization mechanism can be seen in a case of its application to algebraic problem solving (Golding [27]). It tries to chunk the steps that transformed $Y * R = S$ into $Y = S/R$. Note that it is moving R to the *right* of the equation and R appears in the *right* of the expression $Y * R$. It responds to this coincidence of position and builds a rule which has a single variable for these two position descriptors, hence requiring them to be the same. ACT*, by inspecting the actual productions, would recognize this as an accidental coincidence and not build this constraint into the composed rule.

The major effect of composition is to create macro-operators which encode sequences of production rules that frequently follow one another. Again, there is evidence in the human case (McKendree and Anderson [41]) for the existence of such macro-operators. Composition and proceduralization together lead to the development of a rich set of domain-specific operators which encode the special problem structure of that domain.

3.5. Generalization and discrimination

The final category of learning mechanism in the ACT theories has been a mechanism concerned with inductive learning—i.e. learning that goes beyond the received knowledge to infer new knowledge. Clearly, such a learning mechanism is absolutely critical in domains such as language acquisition. The major issue in the evolution from ACT* to PUPS has concerned the mechanisms of induction. Generalization and discrimination were the two inductive mechanisms in the 1983 ACT* theory. They were defined on production rules and produced new production rules in response to the history of old production rules.

The mechanisms of generalization and discrimination can be nicely illustrated with respect to language acquisition. Suppose a child has compiled the following two productions from experience with verb forms:

IF the goal is to generate the present tense of KICK,
THEN say KICK + S.

IF the goal is to generate the present tense of HUG,
THEN say HUG + S.

The generalization mechanism would try to extract a more general rule that would cover these cases and others:

IF the goal is to generate the present tense of =X,
THEN say =X + S.

where =X is a variable.

The production rule formed is the maximally specific generalization of the two productions allowed in the description language for the production system. Given that the description language only allows conjunctions of variabilized clauses, generalizations take the form of deleting clauses and replacing constants by variables.

Discrimination deals with the fact that such rules may be overly general and need to be restricted. For instance, the rule above generates the same form independent of whether the subject of the sentence is singular or plural. Thus, it will generate errors. By considering different features in the successful and unsuccessful situations the discrimination mechanisms would generate the following two productions:

IF the goal is to generate the present tense of =X
  and the subject of the sentence is singular,
THEN say =X + S.

IF the goal is to generate to present tense of =X
  and the subject of the sentence is plural,
THEN say =X.

If there are multiple potential discriminating features, one is chosen at random with a probability that is a function of its level of activation. Over multiple opportunities a search is in effect performed over possible discriminations. The strengths of individual production rules serve as evaluations of how well various rules do and successful rules will be strengthened to the point where they come to dominate. These learning mechanisms have proven to be quite powerful, acquiring, for instance, nontrivial subsets of natural language [5].

These discrimination and generalization mechanisms are very much like similar knowledge acquisition mechanisms that have been proposed in the artificial intelligence literature (e.g., [28, 42, 58]). In particular they can be considered syntactic methods, in that they only look at the form of the rule and the form of the contexts in which it succeeds or fails. There is no attempt to use any semantic knowledge about the context to influence the rules that are formed. A consequence of this feature in the ACT theory is that generalization and discrimination are regarded as automatic processes, not subject to strategic influences and not open to conscious inspection. The reports of unconscious learning by Reber [49] seem consistent with this view. He exposed subjects to examples of strings generated by an unknown finite-state grammar. He found that these subjects were able to judge whether new strings were consistent with the rules of the grammar without ever consciously formulating the rules of the grammar.

There are now a number of reasons for questioning whether the ACT* theory is correct in its position that inductive learning is automatic. First, there is evidence that the generalizations people form from experience are subject to strategic control [24, 33]. In a prototype formation experiment, Elio and Anderson [24] showed that subjects could adopt either memorization or hypothesis formation strategies, and the two strategies led them to differential success, depending on what the instances were. Second, Lewis and Anderson [37] found that subjects were able to restrict the application of a problem solving operator (i.e., discriminate a production) only if they could consciously formulate the discrimination rule. Indeed, there is now reason to believe that even in the Reber unconscious learning situation, subjects have conscious access to low-level rules that help them classify the examples (Dulany, Carlson, and Dewey [23]). They do not form hypotheses about finite-state grammars but do notice regularities in the example sentences (e.g., grammatical strings have two Xs in second and third position).

Another interesting problem with the ACT generalization mechanism is that subjects often appear to emerge with generalizations from a single example [24, 32], while the syntactic methods of ACT* are intended to extract the common features of a number of examples. Both of these difficulties with the ACT induction mechanism, conscious access to inductions and single-example generalizations, are dealt with in the PUPS theory.

4. The PUPS Learning Theory

The PUPS theory (Anderson and Thompson [12]) basically continues the assumptions of the ACT* learning theory except on three scores:

(a) As noted, proceduralization does not require working memory structures to be permanently recorded in long-term memory in order to delete matching to them in production conditions.

(b) The inductive mechanisms of generalization and discrimination which operated on productions have been replaced by inductive mechanisms of analogy and discrimination which operate on declarative knowledge structures.

(c) There has been the introduction of mechanisms of causal inference.

The most fundamental change has been the development of mechanisms for analogy. The two observations, that subjects seemed to consciously formulate their inductions and that subjects could form inductions from single examples, led to a formulation of analogy-based induction in the new PUPS simulation. The basic idea is that students work from declarative representations of solutions to past problems and try to map these onto new solutions. This example-based learning also fits well with our observations of students learning in domains like LISP and geometry [5, 10, 48] where they seem to place a heavy emphasis on using examples of problem solutions.

4.1. Knowledge representation

Analogical problem solving in PUPS makes strong assumptions about the representation of examples. Specifically, it assumes that problems are represented as forms achieving particular functions under certain preconditions. We can encode this information about examples in schema-like structures where function, form, and precondition are three slots among others in the schema-like representation. Below is a representation we might want to impose on the example LISP code (+ 2 3):

```
structure1
  isa: function-call
  function: (add 2 3)
  form: (list + 2 3)
  context: LISP
  medium: CRT-screen
  precondition: context: COMMONLISP
```

It is represented as a function call that adds 2 and 3 in the context of COMMONLISP and which was executed on a CRT screen. The form information states that the example is a list structure consisting of the symbols "+," "2," and "3." The precondition information states that it is essential that the context be COMMONLISP for this form to achieve its function. It would not succeed in INTERLISP, for instance.

4.2. The no-function-in-identity principle

There is a basic semantics underlying the relationship among the form, function, and precondition slots. This semantics is basically that jointly the form and the precondition imply the function. We can represent the example above, for instance, by the following implication:

```
IF goal is to achieve the function (add 2 3)
   and the context is COMMONLISP,
THEN use the form (list + 2 3).
```

However, this is a very specific rule. Suppose we wanted to achieve the function of (add 6 8) in the context of COMMONLISP. This production will fail because of the mismatch of the constant 6 to 2 and the constant 8 to 3. Faced with such a mismatch PUPS will try to variabilize the rule embedded in the example so that it will make the current problem. The production rule we want in the current case is:

```
IF the goal is to achieve the function (add x y)
   and the context is COMMONLISP,
THEN use the form (list + x y).
```

PUPS could always variabilize the rule implicit in an example so that it would match in the current context but at the cost of spuriously extending examples. There has to be reason to suppose that the example above can be appropriately variabilized without spuriously extending it. Note that in the example above 2 and 3 appear in both the function and the form. This is critical to the induction that replaced them by x and y. The inductive principle underlying this is called *the no-function-in-identity principle*. The basis idea is that it is not an accident that 2 and 3 appear in both function and form. It is merely their *positions* in the form, and not their identities which achieve their positions in the function. The analogous function would be achieved by any elements that appeared in the form. Thus, the "no-function-in-identity" principle allows us to respond to the appearance of a term in both form and function of the example by replacing it everywhere by the same variable.

To summarize, PUPS searches for some substitution of variables for constants such that:

(a) each constant in the substitution must appear both in the condition and the action.

(b) The production rule after the substitution matches in the current context.

(c) No more substitutions are used than is necessary to achieve (b). Thus, if both the example and the problem share a constant a variable is not introduced.

This production rule which PUPS extracts to solve the new problem is stored away for future use—i.e., it becomes an explicit rule unlike the implicit rule in the original example.

In extracting this production rule to apply to the next problem, PUPS has basically computed an analogy from the example to the problem defined by the mapping:

$$2 \rightarrow 6\,,$$
$$3 \rightarrow 8\,.$$

All current analogy systems (e.g., [16, 26, 60]) involve putting one structure in correspondence with another such that certain elements map to other elements. These theories differ in terms of their criteria for allowing a correspondence. The "no-function-in-identity principle" is the principle in PUPS. Like all analogical principles it is basically inductive in character.

4.3. The principle of functional elaboration

The example above is simple in that it can be achieved by variabilizing terms that appear only in the example. However, in most interesting cases of analogy, the terms that appear in the function do not directly appear in the form. It is necessary to elaborate the form and/or function descriptions to come up with a representation that can be variabilized. For instance, suppose what we wanted to achieve was to multiply 6 and 8. The above variabilized production would have a mismatch between "add" and "multiply." PUPS can get over this hurdle if it has encoded that "+" implements "add" and "*" implements "multiply" encoded in the following PUPS structures:

```
+: isa: LISP function
   function: (implements add)
   form: (TEXT +)

*: isa: LISP function
   function: (implements multiply)
   form: (TEXT *)
```

One can embellish the representation of the plus example by including this information about the function of the + symbol:

```
IF goal is to achieve the function (add 2 3)
   and the context is COMMONLISP,
THEN use the form (list =function 2 3)
      where =function implements add.
```

Note that the "+" has been replaced by the variable "=function" and "=function" has been given the functional description of "+." This reflects the second inductive principle in PUPS analogy, which we call *the principle of functional elaboration*. The inference is that any term which achieves the function of the replaced symbol will do. There now is a variabilization of this production rule which will extend it to the problem of multiplying 6 by 8:

```
IF the goal is to achieve the function (=op =x =y)
   and the context is COMMONLISP,
THEN use the form (list =function =x =y)
      where =function implements =op .
```

The constraint "=function implements =op" can be satisfied either by inserting a term which satisfies this function or setting a subgoal to find such a term. PUPS performs a search over functional elaborations of terms in the condition and action side of the original production (implicit in the example) looking for some embellished representation where the no-function-in-identity principle can apply—i.e., where points (a)–(c) specified under that principle can be satisfied.

Note that in extracting this production rule from the example and matching it to the problem we have in fact calculated the following mapping of the example onto the problem:

$$\begin{array}{lcl} \text{add} & \rightarrow & \text{multiply}, \\ + & \rightarrow & *\,, \\ 2 & \rightarrow & 6\,, \\ 3 & \rightarrow & 8\,. \end{array}$$

Thus, we have performed the analogy defined by this mapping. However, I prefer to talk about rules and not the analogical mapping because PUPS does store the production rule underlying this analogy. These productions are basically proceduralizations in the sense defined earlier. They eliminate the need to make reference to a declarative structure to repeat a computation. In this case, the data structure eliminated is the PUPS encoding of the example. Subjects do show a dramatic improvement in their problem solving after using a single example and tend to drop out reference to an example in the subsequent problem solving episodes [48].

An interesting question concerns how PUPS selects an example from which to make an analogy. If it selects an example which will not work (because it cannot find a variabilization satisfying the no-function-in-identity principle), it simply tries another. Unguided it may have to go through a great many examples before coming up with a successful one. However, PUPS uses a spreading activation scheme and selects the most active example. This means that it will tend to select a recent example and one that overlaps a lot with the features of the current problem. These biases correspond to the biases in human selection of examples for analogy [51].

4.4. Form to function analogy

One can use the same analogy mechanism to infer the function of a novel form. The following example, adapted from the dissertation research of Shrager [54], shows analogy operating in this fashion. Subjects were presented with a toy tank that had the keypad in Fig. 2. They determined that the key labelled with the up-arrow moved the tank forward and they had to figure out what the keys with the down-arrow and left-arrow did. Below we have PUPS structures that purport to represent their states of knowledge:

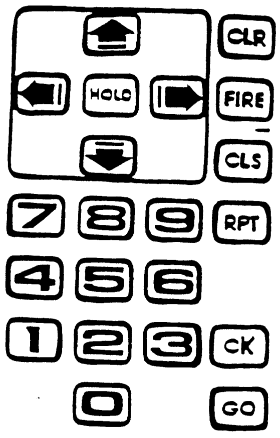

Fig. 2. An example of the device used by Shrager [54].

```
example
  isa: button
  function: (MOVE forward)
  form: (LABELLED up-arrow)

up-arrow
  isa: symbol
  function: (DENOTE forward)

problem1
  isa: button
  function: ?
  form: (LABELLED down-arrow)

down-arrow
  isa: symbol
  function: (DENOTE backward)

problem2
  isa: button
  function: ?
  form: (LABELLED left-arrow)

left-arrow
  isa: symbol
  function: (DENOTE leftward)
```

The example is encoded as an up-arrow with the further information that an up-arrow is a symbol which conventionally means forward. The functions of the other two buttons are not represented but we have represented the conventional knowledge that down-arrows symbolize backward and left-arrows left.

We can represent the knowledge encoded by the example by the following variabilized production:

```
IF there is a structure with form (LABELLED =symbol)
   and =symbol denotes =direction,
THEN the structure has the function (MOVE =direction).
```

This production can be extracted from the example using the "no-function-in-identity" principle and the principle of "functional elaboration" and just switching the form to the condition and the function to the action side of the production. This production enables us to infer that the function of the problem1 button is to move backwards. Similarly we can infer that the function of the problem2 button is to move left. As it turns out only the first inference was correct. The left-arrow button did not actually move the tank in the left direction but rather only turned it in that direction. This is an example of where the "no-function-in-identity" assumption was violated. Some buttons moved the tank in the specified direction and some turned. One simply had to learn which did which. The actual identity of the direction determined the function of the button. This just proves that analogy has the danger of any inductive inference of coming to conclusions which are not in fact correct. The important observation is that human subjects also made this misanalogy. Later we will discuss discrimination mechanisms in PUPS for avoiding such over-generalizations.

4.5. Function to function analogy

Function slots in PUPS can have multiple values corresponding to the basic observation that an object can have multiple functions. The potential for multiple function slots creates a third type of analogy. One can take an example serving two functions and if one of these functions is analogous to the function of a current problem, one can infer that another way to characterize the function of the problem is in analogy to the second function of the example. The following example illustrates this. One might have as a goal to calculate the second element of the list:

```
Goal
  isa: lisp-code
  function: (second lis)
  form: ????
```

Even if one knew what the functions *car* and *cdr* did, one might not see their applicability to this goal because one represented the function of *car* as calculating *first* and *cdr* as calculating *rest* and these relationships do not make immediate contact with the relationship of *second*. However, one might have an example of calculating the second element of a list in some other domain (e.g., the second card in a deck) where that was represented as the first of the rest of the list:

```
example
  isa: card
  function: (second deck1)
            (first deck2)

deck2
  isa: deck
  function: (rest deck1)
```

From this example the following rule can be extracted:

```
IF there is a structure with function (second =lis1),
THEN the structure also has function (first =lis2)
      where =lis2 is the rest of =lis1.
```

This analogical elaboration then allows one to refine the goal function so that the *car* and *cdr* productions can apply.

4.6. Relationship to other work on analogy

There has been a fair amount of research on mechanisms of analogy and it would be useful to relate this current system to other proposals:

4.6.1. *Explanation-based generalization*

Explanation-based learning (DeJong [21], Mitchell, Keller and Kedar-Cabelli, [43]) has some interesting similarities with our analogy mechanisms. Kedar-Cabelli [31] has extended Michell's system to analogy which makes the similarities all the more apparent. In explanation-based learning, the system is given a high-level description of the target concept (the *goal concept*), a single positive instance of the concept (the *training example*), a description of what an acceptable concept definition would be (the *operationality criterion*), and a list of facts about the domain. Included in these facts are abstract rules of inference about the domain. EBG (explanation-based generalization) algorithm tries to find a proof that the training example satisfies the goal concept. To do this it simply expands the terms in high-level description until all the terms in the description meet the operationality criterion. After a proof is generated that the training example satisfies the goal concept, the proof is generalized to form a rule which is capable of matching any instance of the goal concept which meets this same low-level description.

The expansions done by the EBG method are not unlike the functional elaborations done by the PUPS system. The essential difference is that, while the EBG system expands until it either reaches a dead end or the (apparently ad hoc) operational criterion is met, the PUPS system has an implicit operational criterion, which is that the expansion is sufficiently elaborate for the no-function-in-identity principle to apply. A second difference is that PUPS need not be given abstract rules of inference for the domain. It tries to infer these directly from its encoding of examples. Thus, EBG starts out with a strong domain theory and essentially composes new rules; while PUPS discovers the rules hidden in its examples. A third difference is that the EBG method simply characterizes the way in which a single object instantiates a concept, while PUPS draws analogies in order to further problem solving efforts.

4.6.2. *Winston's analogies*

Winston's ANALOGY system [60] is very similar to the work of Mitchell and Kedar-Cabelli discussed above. The major difference is that the rules of inference are stored with the examples and not stored separately and thus the example serves the additional function of providing rules of inference. One of the big differences between ANALOGY and PUPS is that ANALOGY seems only capable of filling in *function* slots (i.e., doing form-to-function mapping). That is, ANALOGY would not be able to generate an example which served a specific function (i.e., do a function-to-form mapping). Indeed, this observation could be made of all the examples we have looked at, except for Carbonell's work (which is only capable of finding form that fills a particular function, and not the reverse). PUPS is the only system we know of that can draw analogies in either direction.

4.6.3. *Gentner's structure mapping*

Gentner's [26] structure-mapping approach to analogies distinguishes between various types of features of the model. In particular, there are *attributes*, which are predicates of only one object, and *relations*, which are predicates of two or more arguments. In analogy, one is only concerned with mapping relations. From this assumption, she distinguishes in a natural way those features which should map when comparing the solar system to an atom. The method for selection of what features will map to the target domain involves a causal analysis of the domains. The *systematicity principle* says that those relations which are central to the functional description of the domain are much more likely to get mapped than those which are not. So, for instance, the fact that the sun is more massive than a planet in some way *causes* the planet to orbit the sun. Thus, this relation is more likely to get mapped to the domain of atoms than the assertion that the sun is *hotter* than the planets (which doesn't cause anything). The causal analysis is similar to Winston's model. The central idea is that if you cannot show a reason for a relation to get mapped, then you shouldn't map it.

The PUPS analogy mechanism does not require a commitment to which relationships will be mapped. It will recruit all and only those functional relationships that are required to enable the no-function-in-identity principle to map. However, causal relationships provide an important type of functional link as we will discuss.

4.6.4. *Carbonell's derivational analogy*

Carbonell's [16] work is different in kind than the systems so far discussed. His basic strategy is to take a worked-out solution for a problem and convert it to the current task. The problem solution may be represented at any level of abstraction (corresponding to various points along the problem solving continuum) as a list of operators along with an elaborate description of the dependencies among the operators and the parts of the problem domain. These dependencies are then evaluated with respect to the current problem, and various editing operations are performed to convert the solution to one appropriate for the current problem.

A major difference between Carbonell's work and our own is that he represents problem solutions as a whole, and requires that the entire solution be transported (modulo certain possible transformations) into a solution to the current problem. In our work, each operator application is done by a separate step (which may either be a learned rule or an analogy), and our solutions may therefore potentially borrow from many different examples. Also, since the generalizations we learn describe an individual step in a problem solution rather than the entire solution, these generalizations are more widely applicable (our theory predicts more transfer to novel problems). We think this more piecemeal approach is closer to the human use of analogy.

4.7. Discrimination

PUPS analogy is meant to replace the ACT* theory of generalization. The compilation of the analogy to produce a production rule amounts to a generalization. As in ACT*, this generalization mechanism needs to be supplemented with a discrimination mechanism which adds appropriate constraints to the generalizations. The precondition slot of a PUPS structure is supposed to encode the constraints on a form achieving its function. When a form fails to achieve its intended function, it is assumed that this is because some critical precondition is missing and an effort is made to find the precondition and encode it within the PUPS structures. Currently, there are two ways such precondition information can be added. Since such precondition information is declarative structure, it can be added by encoding linguistic instruction or by the output of other production actions. Alternatively, PUPS can engage in a comparison of successful and unsuccessful analogies much as ACT* does, looking for critical differences. Applied to the earlier misanalogy in the Shrager situation (Fig. 2) the difference uncovered might be that *left* and *right* require the tank to turn. However, even when PUPS engages in this compare and contrast strategy it differs from ACT* in that the precondition is added to a declarative data structure and so is available for inspection. This produces the phenomenon noted earlier that when subjects make successful discriminations they are able to articulate what the discriminating features are.

4.8. Causal induction

PUPS, being a developing theory, does not have a fixed set of learning mechanisms. Recently, influenced by the work of Lewis [36] we have become convinced that we need to work on principles of causal inference to make the theory of learning more complete. The basic idea is that people have a capacity to infer causal relationships among elements in their experience. Basically, in PUPS terms, a causal induction involves encoding as a function of form1 that caused form2. While we have experimented with mechanisms for causal induction in PUPS, there is not a stable implementation as there is in the case of analogy.

Lewis has identified a number of principles for causal induction, including the following two:

(1) *The identity heuristic*. This basically asserts that when two tokens of the same element appear in a sequence, the first token had a role in causing the second token. Thus, if we type (lis) into the computer and we see a message "lis: unknown function object" we can use the identity of lis between what we typed and the message to infer that our action caused the message.

(2) *Previous action*. This basically asserts that if an event has no other apparent cause, ascribe as its cause the immediately preceding action. Thus, if a computer responds immediately with "rubbish" to a symbol we typed we can infer our typing caused the message.

We have built these two mechanisms into PUPS and have been exploring their interaction with analogy. The following is an interesting example of how analogy and causal induction work in concert.

To set the scenario for this example, imagine a learner who neither knew English nor knew LISP, interpreting the following interactions involving an instructor and a computer user. First, an instructor says "Get the first element of (a b c)." Then a computer user, sitting at a terminal, types (car '(a b c)). Finally, the computer responds with just *a*. Figure 3(a) shows PUPS analysis of these three events which turns on these two principles for causal induction. Using the principle of previous action it infers that the first event caused the

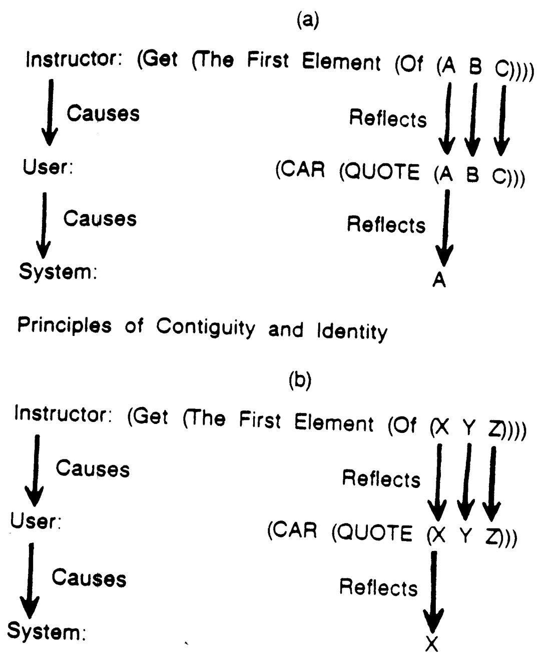
(a)
Instructor: (Get (The First Element (Of (A B C))))
Causes
Reflects
User:
(CAR (QUOTE (A B C)))
Causes
Reflects
System:
A
Principles of Contiguity and Identity
(b)
Instructor: (Get (The First Element (Of (X Y Z))))
Causes
Reflects
User:
(CAR (QUOTE (X Y Z)))
Causes
Reflects
System:
X

Fig. 3. (a) An example of how PUPS develops a causal analysis of a sequence of three events. (b) An example of how analogy extends to analysis in (a) to interpreting a new utterance.

second and the second caused the third. Using the first principle of identity it assumes that successive tokens of the letters reflect each other. These causal relationships are attached as functional elaborations of the events. For example, the instructor's sentence is represented as having the function of causing the user's input and the computer's *A* is represented as having the function of reflecting the user's *A*.

Suppose the learner now becomes the user and is presented with an example of the instructor saying "Get the first element of (X Y Z)." As can be seen in Fig. 3(b) PUPS fills in what the user will type and how the system will respond by analogy. There are two major steps in making this analogy. First, PUPS tries to figure out the function of the instructor's sentence and extracts the following rule:

```
IF the instructor utters =structure1 of form
      "GET the first element of (=X =Y =Z),"
THEN the function of =structure1 is to
            get the user to type =structure2
         and the function of =structure2 is
            to cause the system to produce =structure3
         and =structure3 reflects =component in
            the user's input
         and =component reflects =X .
```

Thus, PUPS infers that the sentence will cause the system to type something which will cause the system to produce something to reflect the X in the utterance.

The second major step of analogy by PUPS involves filling in the form of what the user typed. To do this PUPS extracts the following rule:

```
IF the user types =structure2
     and =structure2 is in response to the
        instructor's =structure1
     and =structure1 has the form
        "GET the first element of (=X =Y =Z),"
THEN the form of =structure2 is
            (CAR (QUOTE =X1 =Y1 =Z1)
         where the function of =X1 is to reflect =X
            the function of =Y1 is to reflect =Y
            the function of =Z1 is to reflect =Z.
```

Thus, PUPS figures out that the LISP code will be (CAR (QUOTE (X Y Z)) where the elements of list reflect the elements in the instructor's list.

Another principle of causal inference, the principle of minimal contrast, adds more potential to the combination of causal inference and analogy. Suppose our learner observes the following events.

(1) As before, the instructor says, "Get the first of (a b c)" and user types (car '(a b c)) and the system responds *a*.

(2) The instructor says "Get the tail of (a b c)" and the user types (cdr '(a b c)) and the system responds (*b c*).

The principle of minimal contrast will allow the learner to compare these two events and make some further causal inferences. This principle says that if two antecedent events are identical except for one place, and two consequent events are identical except for one place, then the thing in the antecedent place causes the thing in the consequent place. So, from the above the learner can infer "first" leads to *car* and "tail" leads to *cdr*. Also, the learner can infer that *car* causes the computer to respond with the first element of the list and *cdr* causes the computer to respond with the tail of the list. So, indirectly at least, the learner has connected the words "first" and "tail" with their meanings.

Now suppose the learner hears the instructor say "Take (m n o) and find its first element." (Note the form of this sentence is different than in (1) above. The user types (car '(m n o)) and the system responds *m*. Using the link between "first" and "car" it can infer that the "*first*" in this sentence form also caused the *car* to appear in the LISP code.

Now the learner is in a position to predict what will happen when the instructor says "Take (x y z) and find its tail." To do this it will use the analogical mechanisms illustrated in Fig. 3 plus the hooks it has inferred between "tail" and *cdr*, and between *cdr* and the relation "tail." Thus, the combinations of causal inference and analogy enable PUPS to understand a novel sentence form.

I think this example illustrates the tip of the iceberg with respect to the potential for interaction between causal inference and analogy. Analogy can extend established function-form relationships to new examples. However, something like causal inference seems the key to creating the function-form analysis of the initial example. For another example of the joint use of causal induction and analogy see Anderson [7].

5. Origins of Knowledge

This then concludes the review of the basic learning mechanisms in the ACT and PUPS theories. For more details and especially for analyses of how they apply to specific situations such as learning LISP, geometry, and language, I refer the reader to publications such as [3, 5, 7, 10, 12]. Now I would like to turn to the original goal of this paper, which is to extract the implications of these learning mechanisms for the issue of the origins of human knowledge. One step in identifying the epistemological claims of the PUPS theory is separating out three levels of analysis in a psychological theory: the knowledge level, the algorithm level, and the implementation level.[1] We need to understand the epistemological claims of ACT of each level.

5.1. The knowledge level

Newell [45] is responsible for the term "knowledge level" and Dietterich [22] has argued for its usefulness in analyzing issues of learning. It is the most abstract level and refers to what a person knows independent of how it is actually encoded and how it actually shows up in a piece of behavior. The distinction between what is known and what is represented is easy to make in the domain of logic. Suppose a system has encoded the logical formula p and p *implies* q. Then that system also knows q even though that is not directly represented. A system can be said to know all the entailments of what it has encoded. Two systems with different encoded information can be said to know the same knowledge if the entailments of these two sets of knowledge are identical. So, curiously, a system that has q and q *implies* p encoded knows the same things as the earlier system because the two sets of premises have identical entailments.

However, given the fundamental behaviorist insight it is a bit strange to talk about human knowledge in terms of its logical entailments. We do not see logical entailments; we see behavior. Therefore, Newell's development of the knowledge level hinges on the *implications* of the knowledge for behavior.[2] He introduces what he terms the "principle of rationality": "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select the action." Two encodings of knowledge are equivalent, then, if the principle of rationality maps them onto the same behavior. Thus, it matters not if a child has encoded "mother has baked oatmeal cookies" and "the cookie jar contains what mother has baked" or encoded "the cookie jar contains oatmeal cookies" and "whatever is in the cookie jar mother baked." In either case a polite oatmeal-loving child, who obeys the principle of rationality, should go to the cookie jar and thank mother for the cookies.

Clearly, learning at the knowledge level is critical to the original philosophical discussions mentioned in the introduction of this paper. They were intended to be about the knowledge level, but part of their confusion was that they mixed in issues of learning at the other levels.

[1] These three levels bear obvious relationships to Marr's [39] three levels of computational theory representation and algorithm, and hardware implementation. The major difference is that the knowledge level is fashioned somewhat differently to help us address the epistemological issues of concern here. The difference between the knowledge level and the other levels also has a clear similarity to Chomsky's [18] competence-performance distinction.

[2] As we will see there is a distinction between the actual behavior of a system and the implications of knowledge for the behavior. A system may not behave as its knowledge implies it should. The knowledge level is concerned with the implications and not the actual behavior.

It is also an interesting fact that the procedural-declarative distinction is a nondistinction when we talk about learning at the knowledge level. The procedural-declarative distinction turns on whether knowledge is prepared for a particular use (and hence is procedural) or not (and hence is declarative). The knowledge level is defined by the principle of rationality which says that if knowledge is there it will be used appropriately.

5.2. The algorithm level

While one might reasonably argue that learning at the knowledge level is the most critical to issues of knowledge, it is by no means the only level nor is the knowledge at this level the only type of knowledge we want to speak about (despite the term Newell gave it). That this is so can be seen by considering chess. One might argue that someone who knows the rules of chess knows how to play a winning game since winning performance follows logically from such knowledge. However, one does not necessarily have the means of turning the knowledge into winning performance. Another example is that the postulates of geometry should imply the ability to do proofs in geometry, but a great deal of learning must occur after students have learned the postulates of geometry before they are facile at generating proofs.

The algorithm level refers to the actual behavioral potentials of the system. The analogy here is to the algorithmic specification of a computer program. This is quite an abstract specification however. Just as a computer algorithm is not committed to the programming language many of the details of the PUPS implementation are irrelevant to understanding human knowledge at the algorithm level. The declarative-procedural distinction is a significant distinction at this level, however. Knowledge that can or cannot be used in a particular way will determine what the system can do.

In ACT* or PUPS, because any procedure must operate on the contents of working memory, there is a close relationship between the algorithm level and working memory. Any algorithm amounts to a specification for a series of transformations of the contents of working memory. Thus, there is also a close relationship between the algorithm level and protocol data because protocol data involves a reporting of the contents of working memory by behavior correlated with these contents. Thus, in addition to verbal protocols [25], protocols can include things like eye movements [30] where the assumption is that the eye is fixated on objects that are represented in working memory.

5.3. The implementation level

The program or algorithm for applying the knowledge leaves under-specified the actual performance of the system. This can be seen by analogy to computer programs. The speed with which a program will run depends on how cleverly it is compiled into code and on which machine it runs. The machine and compiler will determine whether the program will fit in memory, whether it will require swapping, and whether it can run at all. Similarly, in the human situation we can inquire as to the speed and efficiency with which the human program can run. There is evidence that the time and attentional costs of a human skill decreases continuously with practice [5]. In the ACT* theory this improvement with practice is a result of its strengthening mechanisms.

Improvement in such resource costs can actually change fundamentally what a person can do. If the resource demands of a particular algorithm are too high a person will simply fail to be able to successfully execute the algorithm. For instance, Anderson and Jeffries [11] argue that most novice errors in LISP programming result from working memory being overwhelmed by the problem demands. They also argue that experts' increased working memory for domain information means that they are able to cope with problem solving demands and so make fewer errors.

5.4. The relationship among the three levels

It is worth noting that there is a subset-superset relationship among knowledge at the three levels. Any knowledge at the knowledge level is also knowledge at the algorithm and implementation level and any knowledge at the algorithm level is knowledge at the implementation level. As one goes up the levels one loses distinctions. At the implementation level we have an actual specification of what someone will do including errors and actual timing. Anything that changes the behavior will count as learning at the implementation level. At the algorithm level we ignore differences among behaviors that amount to differences in exact time or differences due to working memory being overloaded. Thus, differences in knowledge states that show up only because of performance considerations are ignored at the algorithm level. On the other hand we do pay attention to differences in behavior that are due to fundamental differences in an algorithm. Thus, someone solving a geometry problem by backward search from what is to be proven knows an algorithm different from that used by someone searching forward from the premises. At the knowledge level however, there might not be a difference in the knowledge of these two people so long as their algorithms implied the same decision about whether the conclusion followed from the premises.

Thus, two systems which yield different behavior may not be distinguishable at the knowledge level but only the algorithm level. For instance, there may be no difference between a knowledge level analysis of a chess expert and a duffer. They may both know the same things about chess but the expert may possess better algorithms for realizing that knowledge and better implementations of those algorithms. Knowledge level refers to what the person in principle should be capable of doing, not what he actually does do—just as a proof system should in principle be capable of recognizing the truth of a theorem even if it cannot find the proof in allotted time. By Newell's own admission the knowledge level is a "radical approximation" to the behavior of most programs, "failing in almost all cases to explore the subtleties of the matter."

6. Classification of Knowledge in PUPS

This section will consider the various types of knowledge that PUPS proposes are in the human head and ask the question of how each got there—is it built in (nativism), is it acquired from experience (empiricism), or was it internally computed (rationalism)? To help sharpen the discussion I will consider these questions separately at the three levels of knowledge—the knowledge level, the algorithm level, and the implementation level. Table 1 provides a summary of the conclusions—we have classified knowledge into a three-by-three table according to its three possible origins and the three possible levels of analysis.

6.1. Declarative recordings of the environment

In the PUPS theory, declarative recording is closely associated with the knowledge level. Every time there is a declarative recording from the environment there is a change in the knowledge level. The reason that all declarative recordings from the environment constitute learning at the knowledge level is that they constitute some change in the rational behavior of the system—if only to the question "What happened at time X?" Thus, the situation stamping of events guarantees that the recording of any event is not redundant in its rational consequences with the current knowledge. Of course, learning at the knowledge level also implies learning at the algorithm and implementation levels because of the subset relationships that exist among the levels.

The only changes that occur at the knowledge level are those that derive from declarative recordings from the environment. One might think that it is possible to have rationalist learning at the knowledge level—that is, to have

Table 1
Classification of knowledge at three levels of abstraction according to the origins of the knowledge

	Knowledge	Algorithm	Implementation
Acquired (Empiricist)	Environmental recordings	—	—
Computed (Rationalist)	—	Deductions Inductions Compilation	Strengthening
Built in (Nativist)	Induction Causal inference Weak methods	—	Principles of operation (?)

the system compute knowledge that changes its own knowledge state. For instance, consider the cases in Fig. 3 where PUPS first inferred that the speaker's utterance caused the user's command and then later predicted that another utterance would cause a similar command. Why do these causal inference and analogical extensions not constitute rationalist learning at the knowledge level? The reason they are not is that when we dig down into what the system is doing, we see that it at least implicitly possesses knowledge of causation and induction and is just deriving the consequences of this knowledge with respect to the current events. In the case of causation this prior knowledge would include the identity heuristic and the principle of previous action. In the case of analogy this prior knowledge includes the *no-function-in-identity principle* and the *principle of functional elaboration*. It does not really matter whether the system has explicitly represented the knowledge of these principles and is reasoning in terms of this knowledge or if the knowledge is only implicit in the computations. As far as the principle of rationality is concerned, if the system is behaving as if it has the knowledge, then it does have the knowledge.[3]

6.2. Deductions and inductions

In PUPS there are three ways declarative facts can be added to memory. These facts can come from the outside which is the just analyzed case of learning at the knowledge level. Second, a production rule can add a declarative fact to memory. In this case we are just making explicit an implication already in the system and we have algorithm level learning. Such declarative learning we refer to as deductions.

The third possibility, is that inductive mechanisms of analogy, discrimination, or causal inference, might apply. By the preceding analysis such inductive learning is really another case of learning at the algorithm level because we are simply deriving the consequences of our principles of induction with respect to the current event. Indeed, it is unclear that inductive learning is any different than deductive learning under this analysis. That is to say, we could embed these inductive principles as production rules and the inductions would just be what is added by production actions.

6.3. Knowledge compilation

Knowledge compilation also involves learning at the algorithm level. This might seem peculiar in that knowledge compilation is often thought of as only improving the time or capacity demands of an algorithm. However, in addition

[3] Note that this position differs from that of Dietterich who views inductive learning as learning at the knowledge level. The view taken here is that such learning can be deduced from the inductive principles and the existing knowledge.

to these implementation consequences there are two ways in which the behavior of the system changes.

(a) Knowledge compilation in ACT* can actually change the direction of problem solving because of changes in the conflict resolution. In the terminology of Lewis [35] compilation in ACT* is "unsafe" in that it is possible that a compiled production will fire in situations when the production(s) from which it was compiled would be blocked. This is regarded as a feature, not a bug, in the theory because this allows the system to favor the more efficient rules it has formed. Also, when it does lock the system into an overly narrow type of behavior this corresponds to the Einstellung effect noted in human behavior (Luchins [38]) where subjects will produce a learned solution in a new situation when it is no longer optimal.

(b) Compilation will also change the step size of cognition and the contents of working memory. This can result in changes in what people report in concurrent verbal protocols or in other protocol methods of tapping the contents of their working memory such as eye movements.

6.4. Strengthening

The strengthening processes constitute learning at the implementation level in that they do not change the behavioral potentials of the system, only the capacity costs of performing these behaviors. Strengthening is a kind of rationalist learning. It is a case of a system mulling over its experience and deciding how to optimize its future processing of knowledge. It is not a case of the system strengthening all of the knowledge which is currently active; rather it is a case of the system deciding to strengthen that subset of the currently active knowledge which it judges to have contributed to the current line of information processing. The insertion of this judgment process is what makes it rationalist learning.

It is interesting to note that strength adjustments are basically the only learning mechanism underlying the connectionist models (e.g., McClelland and Rumelhart [40], Rumelhart and McClelland [52]), and such adjustments are said to produce learning that at least approximates learning at the knowledge and algorithm level. The claim of these connectionists is that our learning that Reagan is President is continuous with growth in the strength of our memory of this fact.

It is also of interest to note that these neural models are fundamentally empiricist in character. These changes in neural connections are thought to be passive reflections of current activation levels set up on the network in response to experience. In contrast, this is not the case in ACT* or PUPS, which provide a much more rationalist characterization of knowledge acquisition.

It is interesting to note that the actual implementation of ACT* is very much a matter of neural computation despite the fact that its epistemological assumptions are very different from connectionist theories. This reinforces a claim I have made elsewhere [8] that the fundamental issue between a "symbolic" theory like ACT and a "connectionistic" model is not the issue of neural implementation, where both basically agree, but rather the issue of learning.

6.5. Innate knowledge

To summarize the discussion of the preceding sections, at the knowledge level all learning is a matter of experience, by logical necessity. At the algorithm and implementation level all learning is rationalist, by theoretical choice. What remains to be specified is PUPS's position on innate knowledge.

It is an interesting question what the epistemological status is of innate processes in PUPS, such as those for spreading activation. Since these processes only influence the performance properties of the algorithms people possess, these processes must constitute innate knowledge at the implementation level if they are innate knowledge anywhere. However, even to speak of these as implementation knowledge is a bit strange. Innate knowledge seems only an appropriate concept at the knowledge level.

I think at the knowledge level we can say that there are at least three kinds of important innate knowledge in PUPS: knowledge about causal inference, the inductive knowledge behind analogy and discrimination, and knowledge of a set of weak problem solving methods.

The basic view of human behavior is one of a problem solver who has a set of operators for solving problems and set of methods for applying these operators. Such a system has to start out with a way of extracting operators from experience. In PUPS these are the mechanisms of causal inference, which basically infer that certain things cause certain other things. The actual acquisition of such causal relationships is rationalist learning as we discussed. The principles that underlie this rationalist learning are an important category of innate knowledge about the nature of the world.

In addition to having these operators, we must have some means of deploying them to solve new problems. These mechanisms must include methods which do not have built into them knowledge of the particular problem solving domain to which they apply. These are the so-called weak methods. Analogy, besides being an inductive mechanism, is one weak method which tries to map the solution of one problem to another. Its inductive components enable it to extend the solution to novel contexts but analogy is quite limited in its ability to generate a novel solution. Other weak methods like hill climbing or means-ends analysis do more to concatenate the steps in novel form. In the PUPS framework there is no way to acquire any of the basic weak methods. They have to be built in.

It is interesting to compare PUPS on this score with the SOAR model (Laird, Rosenbloom and Newell [34]) which is said to contain a single universal weak method and that other weak methods emerge in response to encoding of knowledge about a domain. We could do the same thing in PUPS using analogy as the universal weak method. If we encoded a solution to a problem as a means-ends solution we could generate a means-ends solution to an analogous problem. However, this ignores the question of where the original encoding came from. If one traces it back it has to be because the encoding mechanisms knew about means-ends solution. In the PUPS framework, all the weak methods must be traced to innate knowledge. It is not clear the same would not be true if we explored in SOAR where the encodings of domain knowledge came from.

7. Summary

In summary, the following are the assertions of the current PUPS theory on knowledge acquisition:

(0) The knowledge acquisition takes place within a production system architecture which assumes a rich enough pattern-matching ability to match operators to situations and to identify differences between goals and current states.

(1) The system begins with
- (a) a set of rules for inferring causal relationships in experience,
- (b) a set of inductive principles for extending or restricting the causal inference,
- (c) a set of weak methods for deploying these causal inferences in problem solving.

(2) Learning at the knowledge level is empiricist and consists of encoding declarative representations of experience as received.

(3) Learning at the algorithm level is rationalist and involves compiling rules that summarize existing computation, storing the declarative structures written into working memory by production actions, and adding the results of inductive procedures for causal inference and analogy.

(4) Learning at the implementation level is rationalist and involves strengthening the procedural and declarative knowledge as a reflection of its frequency of successful use.

ACKNOWLEDGMENT

I would like to thank Wayne Gray, Claudius Kessler, Clayton Lewis, Jean McKendree, Peter Pirolli, and Kevin Singley for their comments on the paper. Preparation of this paper was supported by contracts MDA903-85-K-0343 from the Army Research Institute and N00014-84-K-0064 from the Office of Naval Research.

REFERENCES

1. Anderson J.A., A theory for the recognition of items from short memorized lists, *Phychol. Rev.* **80** (1973) 417–438.
2. Anderson, J.R., *Language, Memory, and Thought* (Erlbaum, Hillsdale, NJ, 1976).

3. Anderson, J.R., Tuning of search of the problem space for geometry proofs, in: *Proceedings IJCAI-81*, Vancouver, BC (1981).
4. Anderson, J.R., Acquisition of cognitive skill, *Psychol. Rev.* **89** (1982) 369–406.
5. Anderson, J.R., *The Architecture of Cognition* (Harvard University Press, Cambridge, MA, 1983).
6. Anderson, J.R., A rational analysis of human memory, in: H.L. Roediger III and F.I.M. Craik (Eds.), *Varieties of Memory and Consciousness: Essays in Honor of Endel Tulving* (Erlbaum, Hillsdale, NJ, 1988).
7. Anderson, J.R., Causal analysis and inductive learning, in: *Proceedings Fourth International Workshop of Machine Learning*, Irvine, CA (1987).
8. Anderson, J.R., Methodologies for studying human knowledge, *Behav. Brain Sci.* (to appear).
9. Anderson, J.R. and Bower, G.H., *Human Associative Memory* (Winston and Sons, Washington, DC, 1973).
10. Anderson, J.R., Farrell, R. and Sauers, R., Learning to program in LISP, *Cognitive Sci.* **8** (1984) 87–129.
11. Anderson, J.R. and Jeffries, R., Novice LISP errors: Undetected losses of information from working memory, *Hum.-Comput. Interaction* **22** (1985) 403–423.
12. Anderson, J.R. and Thompson, R., Use of analogy in a production system architecture, in: A. Ortony et al. (Eds.), *Similarity and Analogy* (to appear).
13. Atkinson, R. and Shiffrin, R., Human memory: A proposed system and its control processes, in: K. Spence and J. Spence (Eds.), *The Psychology of Learning and Motivation* (Academic Press, New York, 1968).
14. Boring, E.G., *History of Experimental Psychology* (Appleton-Century-Crofts, New York, 1950).
15. Bower, G.H. and Hilgard, E.R., *Theories of Learning* (Prentice-Hall, Englewood Cliffs, NJ, 1981).
16. Carbonell, J.G., Derivational analogy: A theory of reconstructive problem solving and expertise acquisition, Tech. Rept. 85-115, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (1985).
17. Chomsky, N., Verbal Behavior (A review of Skinner's book), *Language* **35** (1959) 26–58.
18. Chomsky, N., The formal nature of language, in: E.H. Lenneberg (Ed.), *Biological Foundations of Language* (Wiley, New York, 1967) Appendix A.
19. Cohen, N.J., Preserved learning capacity in amnesia: Evidence for multiple memory systems, in: L.R. Squire and N. Ballers (Eds.), *Neuropsychology of Memory* (Guilford, New York, 1984).
20. Crowder, R.G., The demise of short-term memory, *Acta Psychol.* **50** (1982) 291–323.
21. DeJong, G., Generalizations based on explanations, in: *Proceedings IJCAI-81*, Vancouver, BC (1981).
22. Dietterich, T.G., Learning at the knowledge level, *Mach. Learning* **1** (1986) 287–316.
23. Dulany, D.E., Carlson, R.A. and Dewey, G.I., A case of syntactical learning and judgment: How conscious and how abstract?, *J. Experimental Psychol. General* **113** (1984) 541–555.
24. Elio, R. and Anderson, J.R., Effects of category generalizations and instance similarity on schema abstraction, *J. Experimental Psychol. Learning, Memory Cognition* **7** (1983) 397–417.
25. Ericsson, K.A. and Simon, H.A., *Protocol Analysis: Verbal Reports as Data* (MIT Press, Cambridge, MA, 1984).
26. Gentner, D., Structure-mapping: A theoretical framework for analogy, *Cognitive Sci.* **7** (1983) 155–170.
27. Golding, A., Analogical problem solving in SOAR, Unpublished Manuscript, Stanford University, Stanford, CA (1985).
28. Hayes-Roth, F. and McDermott, J., Learning structured patterns from examples, in: *Proceedings Third International Joint Conference on Pattern Recognition*, Coronado, CA (1976) 419–423.

29. Jacoby, L.L. and Witherspoon, D., Remembering without awareness, *Can. J. Psychol.* **36** (1982) 300–324.
30. Just, M.A. and Carpenter, P.A., The computer and eye processing pictures, *Behav. Res. Methods Instrumentation* **11** (1979) 172–176.
31. Kedar-Cabelli, S., Purpose-directed analogy, in: *Proceedings Seventh Annual Conference of the Cognitive Science Society*, Irvine, CA (1985).
32. Kieras, D.E. and Bovair, S., The acquisition of procedures from text: A production-system analysis of transfer of training, *J. Memory Lang.* **25** (1986).
33. Kline, P.J., Computing the similarity of structured objects by means of heuristic search for correspondences, Unpublished Doctoral Dissertation, University of Michigan, Ann Arbor, MI (1983).
34. Laird, J.E., Rosenbloom, P.S. and Newell, A., Chunking in SOAR: The anatomy of a general learning mechanism, *Mach. Learning* **1** (1986) 11–46.
35. Lewis, C.H., Production system model of practice effects, Doctoral Dissertation, University of Michigan, Ann Arbor, MI (1978).
36. Lewis, C.H., Understanding what's happening in system interactions, in: D.A. Norman and S.W. Daper (Eds.), *User Centered System Design: New Perspectives in Human-Computer Interaction* (Erlbaum, Hillsdale, NJ, 1986).
37. Lewis, M.W. and Anderson, J.R., Discrimination of operator schemata in problem solving: Learning from examples, *Cognitive Psychol.* **17** (1985) 26–65.
38. Luchins, A.S., *Mechanization in Problem Solving*, Psychological Monographs **54** (248) (1942).
39. Marr, D., *Vision* (Freeman, San Francisco, CA, 1982).
40. McClelland, J.L., Rumelhart, D.E. and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, II: *Applications* (MIT, Cambridge, MA, 1986).
41. McKendree, J.E. and Anderson, J.R., Frequency and practice effects on the composition of knowledge in LISP evaluation, in: J.M. Carroll (Ed.), *Cognitive Aspects of Human-Computer Interaction* (to appear).
42. Mitchell, T.M., Version spaces: An approach to concept learning, Doctoral Dissertation, Department of Electrical Engineering, Stanford University, Stanford, CA (1978).
43. Mitchell, T.M., Keller, R.M. and Kedar-Cabelli, S.T., Explanation-based generalization: A unifying view, *Mach. Learning* **1** (1986) 47–80.
44. Newell, A., A theoretical exploration of mechanisms for coding the stimulus, in: A.W. Melton and E. Martin (Eds.), *Coding Processes in Human Memory* (Winston, Washington, DC, 1972).
45. Newell, A., The knowledge level, *AI Mag.* **2** (1981) 1–20.
46. Newell, A. and Simon, H., *Human Problem Solving* (Prentice-Hall, Englewood Cliffs, NJ, 1972).
47. Nissen, M.J. and Bullemer, P., Attentional requirements of learning: Evidence from performance measures (to appear).
48. Pirolli, P.L., Problem solving by analogy and skill acquisition in the domain of programming, Master's Thesis, Carnegie-Mellon University, Pittsburgh, PA (1985).
49. Reber, A.S., Implicit learning of synthetic languages: The role of instructional set, *J. Experimental Psychol. Hum. Learning Memory* **2** (1976) 88–94.
50. Rosenbloom, P.S. and Laird, J.E., Explanation-based generalization in SOAR, in: *Proceedings AAAI-86*, Philadelphia, PA (1986) 561–567.
51. Ross, B.H., Remindings and their effects in learning in cognitive skill, *Cognitive Psychol.* **16** (1984) 371–416.
52. Rumelhart, D.E., McClelland, J.L. and the PDP Research Group, *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, I: *Foundations* (MIT, Cambridge, MA, 1986).

53. Sauers, R. and Farrell, R., GRAPES user's manual, Tech. Rept. 1, Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA (1982).
54. Shrager, J.C., Instructionless learning: Discovery of the mental device of a complex model, Doctoral Dissertation, Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA (1985).
55. Skinner, B.F., *Verbal Behavior* (Appleton-Century-Croft, New York, 1957).
56. Sternberg, S., Memory scanning: Mental processes revealed by reaction time experiments, *Am. Sci.* **57** (1969) 421–457.
57. Townsend, J.T., Issues and models concerning the processing of a finite number of inputs, in: B.H. Kantowitz (Ed.), *Human Information Processing: Tutorials in Performance and Cognition* (Erlbaum, Hillsdale, NJ, 1974).
58. Vere, S.A., Induction of relational productions in the presence of background information, in: *Proceedings IJCAI-77*, Cambridge, MA (1977) 349–355.
59. Wickelgren, W.A., Single-trace fragility theory of memory dynamics, *Memory Cognition* **2** (1974) 775–780.
60. Winston, P.H., Binford, T.O., Katz, B. and Lowry, M., Learning physical descriptions from functional definitions, examples, and precedents, in: *Proceedings AAAI-83*, Washington, DC (1983).