A GENERAL THEORY OF DISCRIMINATION LEARNING

Pat Langley
Irvine Computational Intelligence Project
Department of Information & Computer Science
University of California, Irvine, CA 92717

Technical Report 86–18

September 20, 1986

To appear as a chapter in D. Klahr, P. Langley, and R. Neches (Eds.), *Production System Models of Learning and Development.* Cambridge, Mass.: MIT Press, 1987.

I would like to thank David Klahr and Stephanie Sage for comments on an earlier version of this paper. Stephanie Sage also carried out the experiments described in the sections on concept learning and heuristics learning. The research described in this paper was carried out during 1980 and 1981, while the author was based in the Department of Psychology at Carnegie-Mellon University.

ABSTRACT

One important component of learning is the ability to determine the correct conditions under which a rule should be applied. We review a number of systems that discover relevant conditions through a generalization process, and discuss some drawbacks of this approach. We then review an alternative approach to learning through discrimination, in which overly general rules are made more conservative when they lead to errors. Unlike generalization-based programs, a discrimination-based system is able to learn disjunctive rules, discover regularities in errorful data, recover from changes in the environment, and learn useful rules despite incomplete representations. We show how our theory of discrimination learning can be applied to the domains of concept attainment, strategy learning, first language acquisition, and cognitive development. Finally, we evaluate the theory along the dimensions of simplicity, generality, and fertility.

Introduction

One of the major goals of any science is to develop general theories to explain phenomena, and one may ask what general mechanisms have so far been uncovered to explain the nature of intelligent behavior in man and machine. Early research in cognitive science and artificial intelligence focused on this issue, and systems like the General Problem Solver (Newell, Shaw, and Simon, 1960) were the result. These programs employed techniques like heuristic search and means-ends analysis to direct attention down promising paths, and such methods have proved to be very general indeed. In contrast to these early efforts, recent work has emphasized the importance of domain-specific knowledge (Feigenbaum, Buchanan, and Lederberg, 1971; Pople, 1977), and much of the current research focuses on expert systems for particular fields of knowledge. While these systems perform admirably in their areas of expertise, they fare very poorly on the dimension of generality.

Hopefully, our final theory of intelligence will consist of more than a few basic search techniques, along with the statement that domain-specific knowledge can be used to direct one's search. In addition, Langley and Simon (1981) have proposed that we look for generality in the *learning mechanisms* through which such domain-specific knowledge is acquired. They have argued that because learning theories should be able to account for both novice and expert behavior, as well as the transition between them, such theories are inherently more general than performance theories alone. Moreover, there is the chance that a single learning theory can be used to explain the transition process for a number of different domains. Given this generality, learning has a central role to play in any theory of cognition, and in this chapter we present a general theory of learning that is stated in the language of adaptive production systems.

We will not attempt to justify our choice of production systems as the formalism in which to cast our models of learning and development, since that issue has been addressed elsewhere in this volume. As we have seen, the natural way to model learning in a production system framework is by the creation of new condition-action rules or productions. The appropriate actions of these new rules can often be determined rather easily, since analogous actions can be observed in the environment. However, the conditions under which these actions should be applied in seldom so obvious. In this chapter we address the issue of how one determines the correct conditions on productions. We review some earlier research that has been concerned with condition-finding, and outline a theory of discrimination learning that begins with overly general rules and that creates variants of these rules with additional conditions when these rules lead to errors. After this, we consider how the theory can be applied to explain learning and development in four rather different domains; these include concept attainment, strategy learning, first language acquisition, and development on a Piagetian task. Finally, we evaluate the theory along a number of dimensions, including its generality.

Previous Research on Condition-Finding

The problem of determining the correct conditions on a rule is not limited to the production system framework, and much of the research on learning in both artificial intelligence and cognitive science has focused on condition-finding methods. Below we review some of

the earlier work on this topic. We begin by describing some systems that learn through a process of generalization, after which we review some limitations of the generalization-based approach. Finally, we consider a number of systems that learn through an alternative process of discrimination.

Learning Concepts through Generalization

The majority of research on condition-finding has incorporated techniques for *generalization*, and the bulk of work on generalization has focused on the problem of learning the definition of a concept from a set of positive and negative examples. In this paradigm, the learning system begins by assuming that all aspects of the first positive instance are relevant to the concept, and systematically removing conditions as they fail to occur in new examples. The basic insight is that one can determine the relevant conditions on a concept by finding those features or structures that are held in common by a set of positive examples. Below we review a number of systems that learn by generalization, and we will see a few of the many variations that are possible within this basic framework. In each case, we describe the learning task, and discuss two key features of the particular learning method – the manner in which the space of rules is searched, and the use that is made of negative instances.

Bruner, Goodnow, and Austin (1956) carried out one of the earliest studies of concept learning, working with concepts that could be represented in terms of simple attribute-value pairs. Although their primary concern was with human behavior on concept learning tasks, they completed detailed task analyses that could have been easily cast as running programs. Many subjects employed a *focusing* strategy for determining the conditions defining a concept.* This approach started with a positive example of a concept, such as a *large blue square*, and initially assumed that all features were relevant to the concept's definition. As new positive instances were encountered, they were used to eliminate some of the features. For example, if a *large red square* were also an example of the concept, then the *color* dimension would be deemed irrelevant. Such comparisons continued until each of the attributes had been varied; those attributes whose alteration led to negative instances were retained, and all other were eliminated. Although this strategy works well with conjunctive concepts like *large and square*, it cannot be used to learn disjunctive concepts like *large or square*. An important feature of this strategy was that for concepts defined as conjunctions of attribute-value pairs, no real search was involved in their determination. Given the current hypothesis as to the concept's definition (some features of which were known to be relevant, and others still in question) and a new positive instance, only a single generalization could result, leading to the removal of the varied attribute from the hypothesis. We shall see that in learning tasks involving more complex representations, life is not so simple, and negative instances play a more important role than in Bruner et al's concept attainment tasks.

* Some of Bruner et al's subjects employed other strategies that did not rely on the "common features" approach. We have not discussed these strategies here because we are mainly concerned with the origin of ideas on generalization-based learning.

Winston (1970) extended this approach to learning structural concepts such as *arch*. His system was given a sequence of structural descriptions, each with an associated classification, with the goal of determining the conditions that would allow the proper classification of new examples. On first encountering a positive instance of a concept, the program included every aspect of the description in its initial definition of the concept. For example, the initial definition of *arch* might state that two standing blocks supported a third block, that the two standing blocks were not in contact, and that each block had certain features of its own. When the system was given another positive example of the same concept, it would find a mapping between the current definition and the new example; this specified the structures held in common by the definition and the example, and the concept was redefined in terms of these common structures.

For instance, if the original arch had a brick as its top block, while another arch used a pyramid in this position, then the shape of the top block would be removed from the definition. This approach was very similar to that described by Bruner et al., except that because of the more complex representation being used, in some cases more than one generalization was possible. Since the program considered only one of these possibilities, it can be viewed as carrying out a *depth-first* search through the space of possible concepts. And since there was no guarantee that the program would always select the right generalization, it required the ability to backtrack through this space as well, and it was in such cases that negative instances came into play. When the definition of a concept led the system to incorrectly predict that a description was an example of that concept, Winston's program backtracked to an earlier, more specific concept definition. The system also used negative instances to identify 'emphatic' conditions such as 'must support.' To ensure that the system did not also have to search for the conditions it should add, Winston presented it only with *near misses*; these were negative instances that differed from positive exemplars by only a few features.* Although Winston's learning heuristics were potentially very general, he never seems to have extended them beyond his blocks world.

Hayes-Roth and McDermott (1976) have described SPROUTER, another program that learned concepts from a series of exemplars. This system used a technique called *interference matching* that was similar to Winston's method for finding common properties between positive examples. However, where Winston used relatively ad hoc rules for selecting the appropriate generalization, SPROUTER carried out a systematic *beam search* through the space of possible mappings. This was a version of breadth-first search in which multiple mappings were entertained simultaneously; when the number of alternative generalizations exceeded a reasonable bound, the search tree was pruned and only the best hypotheses were retained. In making this decision, generalizations that contained few nodes but many relations between them were preferred to generalizations that consisted of less dense structures. SPROUTER focused on positive instances, but negative instances also had an important role in directing the search process. If a generalization was formed that covered nonexam-

* Thus, one can view Winston's system as relying on a simplistic version of discrimination to avoid overgeneralizations. However, since the system's learning method was centered around generalization, and since its version of discrimination could deal with neither far misses nor disjunctive rules, we have chosen to include the work in the present section.

ples of the concept in addition to positive examples, then it was considered overly general, and that hypothesis was dropped from consideration. As a result whole branches of the search tree could be abandoned, and attention focused in more profitable areas. The system also searched the space of examples, selecting those most likely to provide new information. SPROUTER showed its generality by learning a number of concepts, including the structural description for *chair*.

One advantage of breadth-first generalization strategies is that they need not retain any positive instances of a concept, since they need never backtrack. However, negative instances must still be retained if they are to be used in rejecting overly general rules. Mitchell (1977) has done away with the need for negative information as well with his *version space* technique. In addition to maintaining a set of generalizations or *maximally specific versions* (MSVs) of a concept, his system maintained a set of *maximally general versions* (MGVs). New positive instances led to more general MSVs with fewer conditions, which corresponded to the process of generalization in earlier programs. New negative instances led to more specific MGVs with additional conditions.* If a negative instance was encountered that an MSV successfully matched, that MSV was removed from further consideration (much as in SPROUTER). Similarly, an MGV was removed from the competition when a negative instance was found that it failed to match. The main use of Mitchell's system was within the meta-DENDRAL program, in the discovery of conditions for predicting peaks in mass spectrograms. However, the learning heuristics were implemented in a very general manner, and were applied to Winston's arch-learning task as well.

The programs described above certainly do not exhaust the generalization learning paradigm. The interested reader is directed to Vere (1975) for an excellent formal treatment of breadth-first strategies for generalization. Mitchell (1979) provides an interesting overview of work in this area, relating the various approaches and evaluating them in terms of their search strategies and memory requirements. Although most work on condition-finding has been concerned with concept learning, Hedrick (1976) has employed generalization techniques in modeling language acquisition, Vere (1977) has considered the generalization of procedures stated as condition-action rules, and Mitchell, Utgoff, and Banerji (1981, 1982) have applied the version space method to learning heuristics for directing search.

Drawbacks of the Generalization Approach

There is little doubt that progress has been made since the early work of Bruner, Goodnow, and Austin. The most recent generalization learning systems are considerably more general than the earliest programs, in that they can deal with structural and relational representations as well as attribute-value pairs. In addition, they are more robust than their ancestors in that they take advantage of negative information to reduce the search that results from more complex representations, and they organize this search more efficiently. However, this basic approach has a number of drawbacks that limit its value as a path

* Although this aspect of Mitchell's system bears some resemblance to the discrimination learning method, it differs in its continued reliance on finding features held in common by positive instances.

to knowledge acquisition. First, because they examine features that are held in common between examples, generalization-based strategies do not lend themselves to the discovery of *disjunctive* concepts (such as *large or red*). When confronted with positive examples of disjunctive rules, these systems overgeneralize (removing those features not held in common by the disjuncts) and cannot recover. Iba (1979) has extended Winston's depth-first search approach to handle disjunctive concepts, but this method is very costly in terms of computation time.

Second, generalization-based learning systems have difficulty handling errorful data. As before, this results from their dependence on finding commonalities in a number of examples. If even one of these examples is faulty, then the entire learning sequence is thrown into confusion. Again, we are not claiming that one cannot in principle modify generalization-oriented strategies to respond to noise; rather, we would claim that these strategies do not lend themselves to handling noisy environments. For example, Mitchell (1978) has proposed an extension to his version-space technique that can deal with isolated errors. However, such a system would pay a high price for maintaining the additional hypotheses that would be necessary to recover from even a single faulty piece of information.

One final drawback is closely related to the issue of noise. Any program that learns through generalization would have serious difficulty responding to an environment in which the conditions predicting an event actually changed over time. For example, if a tutor decided to modify the definition of a concept in the middle of a training session, a system that searched for common features would rapidly become very confused. Of course, one would not expect a learning system to note such a change immediately, since it must gather evidence to determine whether its errors were due to random noise or to an actual shift. However, the ability to recover gradually from changes in its environment would be a definite advantage in real-world settings, where such changes are all too common. In summary, generalization-based approaches to condition-finding have a number of disadvantages. Below we review five systems that learn through discrimination, an approach to knowledge acquisition that has the potential to overcome these difficulties.

Finding Conditions through Discrimination

Although the majority of research on condition-finding has employed generalization-based methods, some research has been carried out with techniques for discrimination. Members of this class of learning methods start with very general rules containing few conditions, and introduce new constraints as the need arises. Positive and negative examples are used in a quite different manner from the way they are used in generalization-based strategies. Rather than looking for features held in common by all positive instances, these methods search for *differences* between positive and negative instances. These differences are then used to further specify the conditions under which a concept or rule should apply. Let us consider review five systems that learn in this manner.

Feigenbaum (1963) has described EPAM, a computer simulation of verbal learning behavior in humans. This model learned to associate pairs of nonsense syllables such as DAX and JIR through a process of discrimination; using the same mechanism, it was also able to memorize sequences of nonsense syllables. EPAM learned by constructing a discrimination net for sorting different stimuli and responses. Tests bifurcated the tree and specified which path should be taken, while nonsense syllables were stored at the terminal nodes of the tree. This tree was initially very simple, but new branches and their associated tests were inserted as new syllables were encountered. For example, JIR would first be stored below a branch taken if J were the first letter of a syllable; however, once the syllable JUK was encountered, a new branch would be introduced that tested whether the third letter was R, and JIR would be restored on this branch. The particular letter EPAM focused on was determined by the position that had been most recently found to be useful in discriminating between syllables. Associations between stimuli and responses were encoded by storing enough information at a stimulus' terminal node to allow retrieval of the associated response. EPAM's discrimination process accounted for a number of well-known verbal learning phenomena, including stimulus and response generalization, oscillation and retroactive inhibition, and the forgetting of seemingly well-learned responses.

Hunt, Marin, and Stone (1966) described a set of programs that applied EPAM-related ideas to the task of concept learning. They focused on concepts such as *large and red*, in which instances could be represented as attribute-value pairs. Concepts were represented as discrimination nets, with each branch in the tree testing the value of one of the attributes and each terminal node stating whether or not the instance was an example of the concept. The basic learning strategy was to search for attribute-value pairs common to all positive or negative instances. If this failed, then the attribute-value pair occurring most frequently in either the positive or negative class was chosen as the test for a new branch in the tree for the concept.* The same heuristic was then applied recursively to the instances satisfying this test, until a tree was constructed that divided the instances into sets that were exclusively positive or exclusively negative. The researchers carried out a number of experiments with this approach. For example, they found minimal effects as the number of irrelevant attributes and values were increased, but significant effects as the complexity of concepts grew. Limiting memory for past instances increased the number of instances required to determine the correct rule, but an overall saving in computation time resulted, since less information had to be taken into account in forming the discrimination tree. Another interesting finding was that the ability to select instances intelligently did not result in a significant improvement over random selection. Because in general the expression *not(A and B)* is equivalent to the expression *not(A) or not(B)*, Hunt, Marin, and Stone's systems could learn disjunctive concepts simply by viewing positive instances as negative instances and vice versa. Since

* It is not clear whether this work should actually be included as an example of discrimination learning, since it differs considerably from the other methods that we will consider under that topic. However, the representation of concepts as discrimination nets, combined with the fact that common features were not exclusively required, has led us to include the work in this section. Quinlan (1983) has recently extended Hunt, Marin, and Stone's method to learning rules for chess end games, but we do not have the space to discuss his work in detail here.

their approach did not exclusively rely on finding common features, it should also have lent itself to dealing with noisy data, but they did not address this issue.

Langley (1978, 1979) has described BACON.1, a discovery system that operated in the domains of concept attainment, sequence extrapolation, and the induction of mathematical functions. The central heuristic of this program was to search for constant values of dependent terms, and then attempt to determine the conditions under which these constancies held. For example, when presented with a standard Bruner et al concept attainment task, BACON.1 would systematically vary the independent attributes (such as color, size, and shape) and observe the feedback associated with each combination of values. Upon encountering two cases in which the feedback was *yes* (i.e., two situations that were examples of the concept), the system would immediately conclude that *all* combinations would lead to this feedback (i.e., that everything was an example of the concept). However, it would continue to gather additional data, and if it came upon a combination with an unpredicted feedback (such as *no*) it would realize that it had overgeneralized and attempt to correct its error through a simple discrimination process. BACON.1's recovery heuristic was stated as a condition-action rule that searched for some attribute that had the same value in two of the correctly predicted situations, but a different value in two of the incorrectly predicted cases. When such a difference was discovered, the distinguishing attribute-value pair was included as an additional condition on the hypothesis. Since it based its discriminations on only *two* good and bad instances, BACON.1 was able to learn disjunctive concepts like *large or red* as well as conjunctive rules like *large and red.*

Brazdil (1978) has discussed ELM, a PROLOG program that learned from sample solutions in the domain of simple algebra and arithmetic. The system began with a set of rules for associativity, for adding the same number to both sides, and so forth. It was then given a set of practice problems, along with their solution paths. For each problem, ELM went through each step of the solution, comparing the step it would have made with the corresponding step of the known solution. Since the system had no priority ordering for its rules at the outset, it tried all rules that were applicable to the current problem state. However, only one rule application agreed with the solution trace, so the corresponding rule was given priority over its competitors; in the future, this rule was selected in preference to the others. In this way, ELM established a partial ordering on its rule set. Difficulties arose when one problem suggested a certain ordering, and another problem suggested a different one. In such cases, Brazdil's system invoked a discrimination process to create more constrained versions of the competing rules with additional conditions on their application. The new conditions were selected by finding predicates that were true when the rule should have been applied, but false when another rule should have been preferred. The new rules were added to the priority ordering above the rules from which they were generated, so the more conservative rules would be preferred in the future.

Anderson, Kline, and Beasley (1978, 1980) have described ACT, a production system formalism that has been used to model a wide range of cognitive phenomena. Anderson and his colleagues have applied their theory to many aspects of learning, and we cannot review all of them here. Instead, we will focus on a paper by Anderson and Kline (1979), in which they summarize ACT's learning mechanisms and their use in simulating the process of

concept attainment. The authors divided their learning model into three major components. The first of these consisted of a generalization process much like those reviewed above, in which positive instances of the concept were compared, and more general rules (in this case stated as productions) were created with conditions removed or with constants replaced by variables. However, Anderson and Kline also recognized the possibility that this process could lead to overgeneralizations, and included a discrimination mechanism to direct the recovery process. This discrimination technique compared a good application of a rule (when the correct classification was made) to a bad application (when an incorrect application was made) and, based on the differences it found between these two cases, constructed a less general version of the rule by replacing variables with constants.* Finally, the model employed a strengthening process that incremented the strength of a production if it was applied correctly or if it was relearned. Along with a bias toward more specific productions, the strength of a rule was used in deciding which production should be selected at any given time. Taken together, the three ACT learning methods accounted for data from various concept learning experiments with humans. In addition, the discrimination process let the model acquire disjunctive concepts as well as conjunctive ones.

An Overview of the Discrimination Learning Theory

In the previous section, we examined a number of systems that learned through generalization. In finding the definition of a concept, these programs initially assumed that all conditions were relevant, and systematically removed these conditions when they failed to occur in new examples. However, we also saw that an alternative approach to condition-finding is possible. Instead of starting with all conditions, one initially assumes that *none* of the potential conditions are relevant. If the resulting rule is overly general and leads to errors, then one inserts new conditions that make the rule more conservative. This *discrimination learning* approach provides an interesting alternative to the more traditional generalization paradigm. As with generalization-based condition-finding, a discrimination-based strategy can be implemented in many different ways. Below we outline a theory of discrimination learning that we feel has considerable potential, beginning with a discussion of PRISM, the formalism in which the theory is stated. While our approach shares some features with the earlier work on discrimination, it has some important differences as well.

---

* In another paper, Anderson, Kline, and Beasley (1980) describe a different version of the discrimination process in which a new condition was added to lessen the generality of the discriminant production. This approach is very similar to the one taken in the current paper, and not by coincidence, since we were involved in early discussions with Anderson and his co-workers about discrimination learning. In fact, the initial version of Anderson's discrimination process generated variants whenever a rule was applied, taking only positive instances into account. Although this approach would in principle lead eventually to a rule including all the correct conditions, the search for these conditions would be very undirected and could take a very long time. Based on the condition-finding heuristic that was then being implemented in BACON.1, we suggested that the process instead construct variants based on differences between positive and negative instances, and this proposal was later incorporated into ACT.

The PRISM Formalism

Out theory of discrimination learning is implemented in PRISM, a production system language that was designed as a tool for exploring different architectures and their relation to learning mechanisms. PRISM is a very flexible language through which the user can create any of a large class of production system architectures. Thus, the particular architecture in which the discrimination learning theory has been implemented is only one of many possible PRISM-based schemes. However, since we have found this particular framework to be the most useful for building actual learning systems, we will focus on its characteristics here.

We will not review the basic production system cycle, since that has been covered elsewhere in this volume. However, the PRISM conflict resolution procedure is worth some discussion, since it interacts with the various learning mechanisms. On every cycle, PRISM applies the following (ordered) sequence of conflict resolution principles:

1. *Refraction.* Eliminate from consideration every instantiation that has already been applied; this lets the system avoid simple loops by forcing it to focus on new information.
2. *Production strength.* From the remaining instantiations, PRISM selects those matched by productions with the greatest *strength*; this serves to focus attention on rules that have been successful in the past or that have been learned many times (see below).
3. *Recency.* From the revised set, PRISM selects those instantiations that match against the most *recent* elements in working memory; this serves to focus attention on recently established goals in preference to older ones.
4. *Random selection.* If multiple instantiations still remain at this stage, PRISM selects one of them at random. Thus, one and only one production instantiation is applied on each cycle.

These particular conflict resolution strategies are far from new. For example, Forgy (1979) has used both refraction and recency in his OPS4 production system language, and Anderson et al (1980) have relied on strength in their ACT models. The above combination of strategies was determined partially by analysis, and partially by trial and error, to be the most useful in constructing robust adaptive production systems. Refraction is foremost because trivial looping must be avoided. Strength is second in line because some measure of a rule's usefulness or success is essential to direct search through the large space of productions that could conceivably be constructed. Recency is essential for domains such as strategy learning and language acquisition, where the order in which goals have been added to memory can be used to order behavior. Finally, random selection is required if one wishes to retain the standard assumption of the serial application of productions.

PRISM also includes a number of mechanisms for modeling learning phenomena. The most basic of these is the *designation* process, which allows the creation of a new production as the action of an existing rule. By including the general form of a rule in its action side, a designating production matching against a particular situation can carry over variable bindings to its action side, and in this way creates a specific rule based on the general form. A second process lets PRISM create generalizations of existing rules when it discovers two productions in its memory that have isomorphic structures; the resulting rule includes variables

in places that the two specific rules had differing constants, and so it can match in situations that the less general versions could not. (The PRISM-based learning systems that we will be discussing in later sections do not take advantage of this capability.) PRISM also has the ability to strengthen or weaken productions and so affect their order of preference during the conflict resolution stage. Rules are strengthened whenever they are recreated by any of the learning mechanisms, but they can also be strengthened (or weakened) explicitly by the action side of an arbitrary production. Finally, PRISM incorporates a *discrimination* process that can be used to recover from overly general rules; we discuss this learning mechanism in some detail below.

Finding New Conditions

We will begin our discussion of discrimination learning in terms of a simple example, and introduce complexities as we proceed. Consider a concept learning task like that studied by Bruner et al, in which we have four potentially relevant attributes – size (either *large* or *small*), color (*blue* or *red*), shape (*circle* or *square*), and thickness of the lines making up the shape (*thick* or *thin*). Also suppose that we start with a rule that predicts that every combination of values is an example of the concept:

> If you have a combination of values,
> and you have not yet made a prediction,
> then predict that the combination is an example.

Clearly, this overly general rule will lead to many errors, but suppose that our concept is *large and square*, and that initially the system is presented with the combination *large thick red square*. In this case, the rule will produce a correct prediction, and this information would be stored for future reference. Now suppose that the next combination is *large thick red circle*, which is not an example of the desired concept. In this case our initial rule will make a faulty prediction, indicating that it lacks one or more relevant conditions.

When such an error is noted, the discrimination process is evoked to produce more conservative variants on the original rule. To accomplish this, the learning mechanism retrieves information about the situation in which the faulty rule was last correctly applied, and compares it to the current situation, which led to an incorrect application. The goal is to discover differences between the good and bad situations, and in this case only one difference is noted: in the good case the shape was square, while in the bad case the shape was circle. Accordingly, the discrimination process constructs a variant on the original rule with a new condition:

> If you have a combination of values,
> and you have not yet made a prediction,
> **and the shape is square,**
> then predict that the combination is an example.

This rule is guaranteed to match in the correct situation, but will fail to match in the incorrect one, avoiding the error made by its predecessor. In order to select between competing productions, each rule is given an an associated *strength*. When a variant rule is

first created, it may be weaker than the more general one from which it was created, but if the same rule is learned many times, it will eventually come to exceed its ancestor. The initial strength of a rule is controlled by a user-modifiable parameter, as is the amount by which rules are strengthened upon recreation. In addition to strengthening a rule each time it is relearned through discrimination, a faulty rule is weakened when it leads to an error. Since stronger rules are preferred to their competitors, these strengthening and weakening strategies bias the learning system in favor of more successful productions, so that useful variants eventually come to mask the rules from which they evolved.*

Although this second rule is better than the original version, it still does not represent the entire concept *large and square.* Before the complete concept is acquired, another error must occur. Suppose our second rule has gained sufficient strength to mask its ancestor, and the combination *small thick red square* is presented. In this case, the variant rule will apply, incorrectly predicting that this combination is an example of the concept. Again discrimination is evoked, this time comparing the combinations *large thick red square* and *small thick red square.* As before, only a single difference is noted, and a still more conservative rule is created:

If you have a combination of values,
  and you have not yet made a prediction,
  and the shape is square,
  and the size is large,
then predict that the combination is an example.

Once this rule has been learned enough times to mask its predecessors, our system will always correctly predict whether a combination is an instance of the concept. Although it should have helped to clarify the basic nature of discrimination learning, this example oversimplifies the process along a number of dimensions. For example, it assumes an attribute-value representation rather than more general (and complex) relational structures. Also, we have ignored the possibility of learning negated conditions through discrimination, and we have omitted the details of learning from far misses, in which one must consider a number of alternative variants. Below we discuss each of these complications on the basic discrimination learning method.

Finding Negated and Complex Conditions

One extension of the basic method concerns the acquisition of rules with negated conditions. In the above example, including the condition that *the size is not small* would be equivalent to the condition *the size is large.* However, this results from the fact that each attribute took on only two values; if we had allowed the size to take the value *medium* as well, then the two conditions would have quite different meanings. The discrimination learn-

* For those who dislike notions of strength, one may instead view this number as a measure of the rate of success for the rule, with bias being given to rules with a more successful history. Of course, one might also delete overly general rules, but later we will discuss some reasons for retaining them.

ing mechanism can be easily extended to allow the discovery of negated conditions. In the previous example, if a fact was present during the good application and absent during the bad application, then it was included as a positive condition in a new variant rule. But the reverse reasoning holds equally well: if a fact was present during the bad application but absent during the good application, then it should be included as a *negated* condition in a new rule. This means that variants with positive and negative conditions may be created in response to the same error, but since one cannot distinguish between them on the basis of a single example, this is a reasonable decision. If one of the rules is more useful than the other, then it should be created more often and eventually come to mask its competitor.

A second extension allows the discovery of many conditions at the same time. This can occur in the context of structural or relational representations, when a single fact is not sufficient to distinguish between a good and bad application. As an example, suppose that our system is learning the concept *uncle*, but starts with an overly general rule that believes the uncle relation holds between any two people. Suppose further that it correctly predicts that *Joe* is the uncle of *Sam*, but incorrectly predicts that *Jack* is the uncle of *Steve*. Upon examining the information associated with each of these situations, it finds that both *Sam* and *Steve* have a parent; thus, this condition in itself is not enough to predict the uncle relation in one case but not the other. But on continuing its search, the discrimination process finds that *Joe* is the brother of *Sam*'s parent, while *Jack* is not the brother of *Steve*'s parent. Thus, the *conjunction* of the parent and brother relations is sufficient to tell between the two instances, and these two relations would be included as conditions in a variant on the original rule. Analogous situations can lead to a conjunction of negated conditions; a rule containing such a conjunction will match if any subset of its negated conditions match, but not if all of them are true simultaneously. In principle, this approach may be used to find variants with an arbitrary number of new conditions; in practice, the search must be constrained to a reasonable depth, allowing no more than four or five conditions to be found simultaneously.

Selecting between Alternative Variants

The reader may have noted the careful crafting of the above examples, so that only one difference occurred in each case. This meant that the relevant conditions were obvious, and the discrimination mechanism was not forced to consider alternative corrections. Unfortunately, one cannot always depend on a benevolent tutor to present an ideal sequence of examples. Accordingly, when an error is detected, the discrimination process considers all differences between the correct and incorrect situations, and constructs *all* of the corresponding variants. For example, suppose the good case was *large thick blue circle*, while the bad case was *small thin red square*. Here four* rules would be created, one with *large* as a new condition, one with *thick*, one with *blue*, and one with *circle*. Some of these differences may have nothing to do with the actual concept, but each of the rules is initially given a low strength. Only if the same rule is constructed many times will it have a chance to play a role in the decision process.

* Four additional rules would be created as well if we included negated conditions; these would include *not small*, *not thin*, *not red*, and *not square* as conditions.

The strengthening process is the key to focusing attention on promising variants. Even if a variant rule is still overly general, it will tend to be created more often than other variants in which the wrong conditions were inserted. This means that it will be selected in preference to its competitors, and lead to errors of its own. When this occurs, the discrimination mechanism will generate variations on this rule with still more conditions, which in turn may generate their own errors and lead to even better rules. Thus, though the discrimination process may generate many completely spurious rules, these tend to be ignored and only the most promising variants are considered further. The entire process may be viewed as a *beam search* through the space of rules, in which only a few nodes are chosen for expansion at any given depth. This strategy makes for much slower learning than occurs in generalization-based approaches, but the strategy is also much more robust than the more traditional methods, as we discuss below.

The Advantages of Discrimination Learning

In the previous section we discussed some drawbacks of generalization-based learning methods. We found that generalization ran into difficulties when confronted with disjunctive rules, noisy environments, or rules that change over time. In contrast, a discrimination learning technique like the one just outlined responds comparatively well in these situations. Because our discrimination process compares only a single good situation to a single bad situation, disjunctive concepts can be easily acquired. The disjunct that is learned in a particular case depends on the good situation that is examined, but since the most recent good situation is used, each of the disjuncts will eventually be found. The combination of discrimination and strengthening allows learning in noisy environments, since the occurrence of occasional errors will have little effect on a learning algorithm that sums across many different examples. Finally, the weakening of faulty variants provides a natural mechanism for backing up through the space of rules, should this ever be necessary due to a change in the environment. In summary, the proposed discrimination learning theory appears to embody a very robust approach to learning that deserves further exploration. In the following section, we propose some domains in which to test this approach to knowledge acquisition.

Relation to Earlier Research on Discrimination

Although the majority of condition-finding research has been concerned with techniques for generalization, we have seen that a few researchers have employed discrimination-based learning methods. At least some of these researchers have realized the potential of discrimination to deal with the issues we have discussed. Hunt, Marin, and Stone used their technique to learn disjunctive concepts by viewing positive instances as negative instances and vice versa. Anderson and Kline used discrimination to learn disjunctive rules when overgeneralizations occurred, and seemed to realize the importance of a strengthening component for dealing with noisy environments. However, to our knowledge they have not explored these issues in any detail. Thus, our theory should not be viewed as an entirely new approach to the task of determining relevant conditions. Rather, it is an attempt to extend a promising approach to learning that has received relatively little attention, and to cast this approach in sufficiently general terms that it can be applied to a variety of domains.

Since our learning method bears a close resemblance to that employed by Anderson and Kline, we should spend some time discussing the differences. First, the earlier researchers used discrimination mainly to recover from overgeneralizations, while we are exploring discrimination separately from generalization-based approaches. Second, Anderson and Kline's version of discrimination created only a single variant whenever it was evoked, while the proposed method constructs a different variant for each difference that is found. Thus, the earlier technique can be viewed as carrying out a depth-first search through the space of possible rules, while our method carries out a beam search that should be able to discover all useful variants. Finally, Anderson and Kline's variants differed from their predecessors by including only one additional condition.* In contrast, our discrimination mechanism can discover more complex differences that lead to the addition of multiple conditions, and to the addition of negated conjunctions that constrain the variant in complex ways. This allows the discovery of rules that, as far as we can determine, cannot be learned with any of the existing generalization-based or discrimination-based methods. Thus, while the new method has many similarities to Anderson and Kline's earlier technique, there are some important differences as well.

Learning Concepts through Discrimination

The majority of research on condition-finding has taken place in the context of concept attainment tasks. Therefore, this was a natural area in which to begin our exploration of discrimination learning. And because concept attainment tasks deal with condition-finding in its purest form (since no additional learning issues are involved), this was an ideal domain for testing our major claims about the new approach. Accordingly, we set out to construct a general system that could learn a variety of concepts when presented with examples and non-examples of those concepts. Below we present an overview of this system, after which we discuss some empirical explorations of the program's behavior.

An Overview of the Program

The program's task is to learn the conditions that correctly predict all positive instances of a concept, without predicting that any negative instances are examples. Since it must have data on which to base its learning, the system begins by selecting an example at random from user-supplied lists of good and bad instances. (The probability of choosing a good instance is 50 percent.) Once an instance is available, the system makes a prediction about whether that instance is good or bad. When no other rule is available or preferred,† a default rule cautiously decides that the instance is not an example of the concept. Next, the program

* In some cases, their system also replaced variables with constants; however, this can be simulated by the addition of a new condition that restricts the symbols that a variable will match against.

† As mentioned before, each rule has an associated *strength*. When an existing rule is reconstructed, its strength is incremented. However, until its strength exceeds that of the default rule, it will never be selected. Thus, the default rule's strength effectively acts as a threshold which other rules must exceed before they are considered.

compares its prediction to the correct answer. Errors of *omission* occur when the program fails to correctly predict a positive instance. In such cases, the system designates a new, very general rule that predicts *any* situation to be a positive instance. This will be the first learning response of the program, since it can only make negative predictions at the outset.

When the system correctly predicts a positive instance, it stores the instantiation of the rule that made that prediction. This information is used in recovering from errors of *commission*, which occur if the system predicts a positive instance when a negative instance actually occurs. When such an error is noted, the program evokes the discrimination process in an attempt to generate more conservative variants of the responsible rule (whose strength is decreased). As we have seen, this routine compares the bad instantiation of the rule that made the faulty prediction to the most recent good instantiation of the same rule. For every difference that is found, a variant of the overly general rule is created which includes that difference as an extra condition. When a variant is rebuilt, its strength is increased so that it has a greater priority. If any of these variants are still too general, they will produce their own errors of commission and lead to even more conservative variants. Thus, the discrimination process can be viewed as carrying out a breadth-first search through the space of rules, considering simpler variants before more complex ones.

To summarize, our concept learning system has six distinct components, each stated as a separate PRISM production. These components are:

- A production that selects an instance at random from the space of possible instances.
- A production that asks the tutor whether an instance is an example of the concept to be learned (i.e., a rule that gathers feedback);
- A default production that predicts that an instance is *not* an example of the concept when no other prediction has been made;
- A designating production that creates a rule for predicting that all instances are examples whenever an error of omission occurs;
- A production that notes when a correct prediction has been made, and that assigns credit to the responsible rule;
- A production that notes errors of commission and evokes the discrimination process in an attempt to construct more conservative variants of the responsible rule.

The production system itself is quite simple, since the real power of the program lies in the discrimination process that is called when overly general rules lead to faulty predictions. In the following pages, we see how the overall system learns different concepts under various conditions.

Learning Conjunctive Concepts

Since most research on concept learning has focused on conjunctive rules, it seemed reasonable to first see if the model could acquire such concepts. Rules stated in terms of attribute-value pairs are the simplest, so our initial runs were on concepts such as *large* and *large-and-thick*. The learning path in these runs was much as expected. Since no rule for

predicting examples of the concept existed at the outset, errors of omission led the system to create such a rule. Once this became strong enough, it led to errors of commission and a call on the discrimination process. As a result, variants with single conditions were constructed, with those containing useful conditions eventually exceeding threshold after they had been built repeatedly. When these overly general rules led to errors, productions containing a second condition were generated, again with useful variants eventually exceeding threshold. This process continued until the correct rule gained sufficient strength to be selected, after which no additional errors were made.

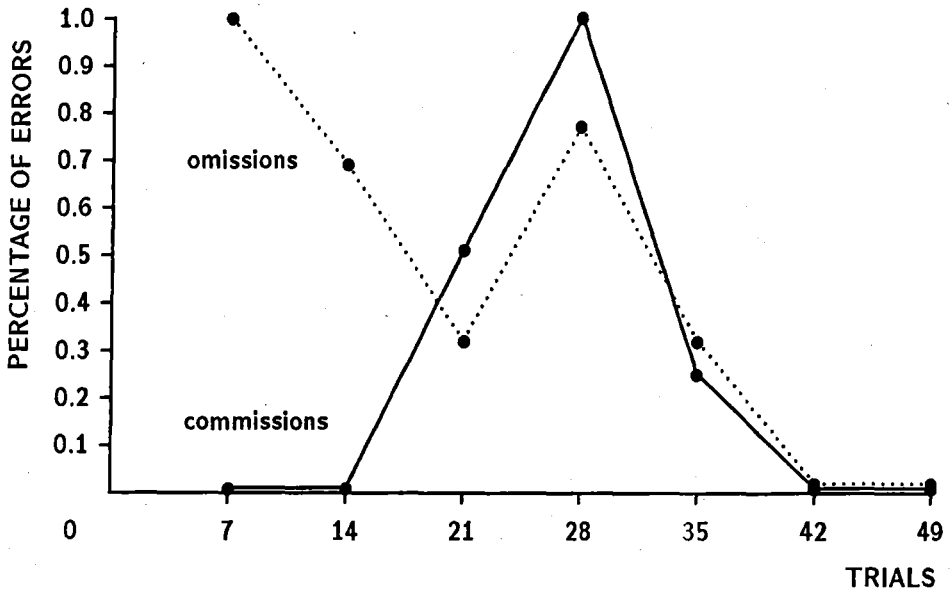

Figure 1. Errors in learning a single-condition concept.

Figure 1 presents the errors made by the system as a function of time, while learning the concept *blue*. As one would expect, errors of omission abound at the beginning of the run, but they become fewer as experience is gained (growing again at one point), until eventually they disappear entirely. Errors of commission are initially absent, but they grow to a peak at the center of the figure as overly general rules become stronger. However, as the dominant rules become more specific, such mistakes drop off and finally disappear as well. Figure 2 presents the trials to criterion (the number of examples until no errors occur) for conjunctive concepts of varying complexity. Since the discrimination process moves from general to more specific rules, simpler concepts with fewer conditions are mastered more rapidly than more complex ones.* Although one might expect a linear relationship, it does not occur. This results from the fact that as the system masters more of the conditions in a concept's definition, the chance for errors of commission decreases, since the dominant rules are more nearly correct. Since variants are created or strengthened only when errors of commission occur, the learning process slows down as the correct rule is approached. The effect is more

* Note that a generalization-based learning system would predict exactly the opposite trend, since it starts with very specific rules and removes conditions as it progresses.

pronounced when the correct rule includes many conditions, since a larger fraction of the instances are correctly classified when the concept is nearly learned than would be the case if a simpler concept were nearly learned. One solution to this problem is to let the program select its own instances, with a preference for those which it would predict as examples of the concept.

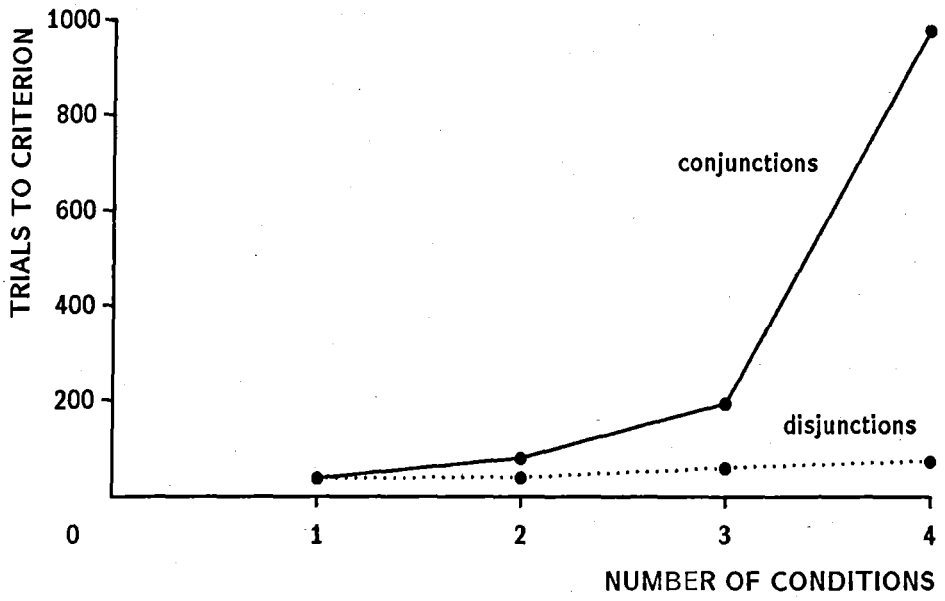

Figure 2. Learning speeds for conjunctive and disjunctive concepts.

The reader will recall that the discrimination process is not restricted to learning concepts based on attribute-value pairs. In addition, it can discover rules incorporating complex structures, such as *uncle-by-marriage* (i.e., the husband of a parent's sister). Rather than applying to a single object, this concept concerns a relationship between two people. Moreover, this concept cannot be described in terms of a simple conjunction of features; these features must be related to one another in certain ways as well. This means that the discrimination process will sometimes be required to discover complex differences between good and bad instances. We do not have the space to trace the system's evolution on such concepts, though it has successfully learned them.

Learning Disjunctive Concepts

In addition to learning conjunctive concepts, discrimination can also master disjunctive rules. This results from the breadth-first nature of the mechanism's search through the space of possible conditions. Let us trace the system's progress on a simple disjunct like *large or red*. Suppose the program first encounters a positive exemplar that is *large and blue* (along with a number of other features), and later encounters a negative instance that is *small and blue* instead. Upon noting the differences, the discrimination mechanism will construct a variant with *large* in its condition side (and perhaps others as well). Now suppose the system meets with another positive instance containing the features *small and red*, followed

by a negative example that is *small and blue*. In this case discrimination will create a variant with *red* as an extra condition. In summary, the method is capable of learning disjunctive rules because it focuses on a single positive and a single negative instance at a time. The particular pair it examines will determine which of the disjuncts will be found in that case. This is in sharp contrast to generalization-based approaches, which rely on finding features common to all positive instances to determine the appropriate conditions.

Besides summarizing the system's behavior on conjunctive rules, Figure 2 also graphs the complexity of disjunctive concepts against their learning time. This curve is also nonlinear, but it is considerably less steep than its conjunctive counterpart. The reason for this lies in the strategy used to search the space of concepts. Because disjunctive concepts are stated as separate rules, and because the space is searched in a breadth-first manner, disjunctive rules can be found more or less in parallel. However, since generally only one of the disjuncts can be strengthened when an error of commission occurs,* one might expect a linear relationship. But as with conjunctive concepts, fewer useful negative instances occur in later stages of the learning process. Of course, the system can also discover disjunctive relational concepts like *uncle*, which can be restated as *uncle-by-marriage or uncle-by-birth*. The complexity of such disjunctions cannot be treated as strictly cumulative, since they share some conditions, such as the fact that the uncle must be *male*.

Dealing with Noise

The strengthening process has proved very useful in directing search through the space of concepts, but one can imagine a discrimination-based learning system that makes no use of strength and is still capable of learning conjunctive and disjunctive rules. However, when one introduces noise into the environment, ordering rules along some measure of success becomes much more important. On reflection, it becomes apparent that two forms of noise are possible: a positive instance may be marked as a negative instance (positive noise), or a negative instance may be marked as a positive instance (negative noise). The effects on discrimination learning are different in these two situations. In the case of positive noise, a positive prediction is incorrectly marked as an error of commission; discrimination will be evoked, and since two positive instances are being compared, any variants are guaranteed to be spurious. A negative prediction in this case is less serious, since the system will simply fail to note an error of omission, and the general rule normally designated in such situations will not be strengthened. In the negative noise case, a negative prediction is incorrectly marked as an error of omission. The resulting construction of an overly general rule will in itself do little harm, but the "good" instantiation stored with it will actually be a bad instantiation. Accordingly, when the next error of commission occurs, two bad instantiations will be compared for differences, and again any resulting variants will be incorrect. Since a good instantiation remains with a production until it is replaced by another, this second type of noise may have cumulative effects. If instead the system makes a positive prediction, it will not be detected as an error of commission, and an opportunity for creating or strengthening useful variants will be lost.

* Occasionally more than one of the disjuncts is present during a positive instance; in such cases, variants containing each of the disjuncts are constructed following an error of commission.

Figure 3 presents the trials to criterion for a two-condition conjunctive concept with varying levels of noise. A noise level of zero represents no noise, while a noise level of 0.5 represents maximum noise or perfect randomness.* Separate curves are shown for the two types of noise. The system was never presented with both forms of error in the same run. One would expect the system's performance to degrade gracefully as more noise was introduced, but this does not appear to be the case. Instead, the program's behavior was largely unaffected by noise in the 0 to 30 percent range, and then was affected drastically when the noise rose above this level. Whether these results are due to the chance order in which the data were presented in these runs, or whether this is a basic characteristic of the system, can only be determined by further experiments with the program. Still, it is clear that the discrimination learning approach, combined with a strengthening mechanism to focus attention on promising variants, is capable of discovering concepts despite significant amounts of noise.

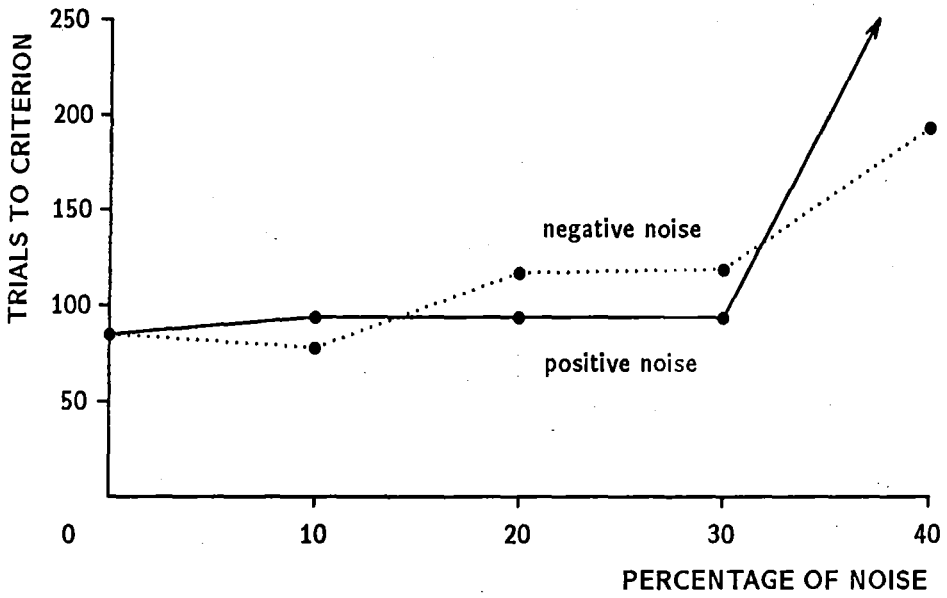

Figure 3. Effect of noise on learning a two-condition concept.

Adapting to a Change in the Concept

The strengthening mechanism has not only proved useful for learning concepts in the presence of noise, but also in the process of recovering from a change to a well-learned concept. Any intelligent system must be capable of revising its prediction rules as change occurs in the environment. For example, suppose that tomorrow our traffic laws were changed so that the color of stop lights became blue instead of red. Presumably we would learn to stop when the light turned blue. The concept learning program has succeeded on a similar task in which it must alter the conditions under which it predicts an example of a concept. The system is

* A noise level of 1.0 would lead to complete regularity, though the system would learn the negation of the rule it was intended to learn.

presented with instances of one concept until it achieves criterion performance at predicting that concept. Then the concept is changed and the program must revise its prediction rules in order to regain its previous level of success.

Figure 4 describes the program's performance on this task, mapping the percentage of errors of omission and commission made against the number of trials. For the first forty trials, the learning system is presented with positive and negative examples of the concept *blue*. This process is similar to that described earlier in reference to simple conjunctive concept learning. The criterion of perfect performance is reached by trial 30, and the subsequent ten trials are predicted correctly. At this point the system's strongest rule states that if the color of a token is *blue*, then the token is an example of the concept. This rule fires on all positive instances so that no errors of omission are made. The next to strongest rule is the default rule, which makes negative predictions. This production fires on all the negative instances, so no errors of commission are made, either. A number of weaker rules created in the process of learning the concept *blue* are also present, but they are not strong enough to compete with the other two and do not fire.

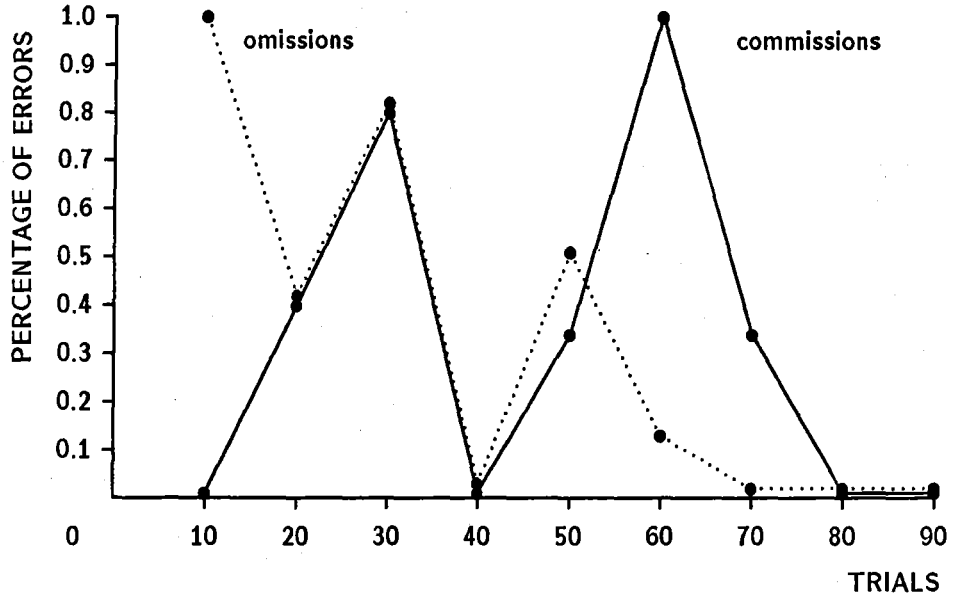

Figure 4. Errors with a change of concept on trial 40.

At trial 41, the concept is changed to *square* and the program's current set of rules becomes insufficient for continued perfect performance. Errors of commission are made in response to instances that are *blue* but not *square*, causing the *blue* rule to be weakened and variants to be created. Errors of omission are made when the program is presented with examples which are *square* but not *blue*, resulting in the strengthening of the most general prediction rule by designation. Note that errors of both types increase rapidly following the change of the concept, with omission errors reaching a peak at trial 50. Errors of commission rise even higher after the overly general rule becomes strong enough to fire. This rule consistently makes commission errors, and variants, including the *square* rule,

are created (or strengthened, if they were created while learning the concept *blue*). Errors decrease gradually as the system revises its rules, until by trial 80 both types of errors have diminished to nothing, and perfect predictive power has been achieved again.

Summary

In conclusion, the discrimination-based approach to condition-finding provides a viable alternative to the more traditional generalization-based paradigm. Discrimination can be used to learn conjunctive concepts, including those composed of complex relational terms. And because it compares a single good instantiation to a single bad instantiation, the technique can discover disjunctive concepts as well. Finally, when combined with a strengthening process, discrimination can successfully learn the desired conditions in the presence of noise, and can recover from changes in the environment. One limitation of the program revolved around its inability to intelligently select instances that might lead to errors of commission, and this is an obvious direction in which the concept learning system should be extended. Although we do not wish to detract from the excellent work on generalization, we believe that the discrimination learning approach offers some distinct advantages, which we will explore further in the following sections.

Learning Search Strategies through Discrimination

Upon first encountering a problem-solving situation, humans engage in search; however, once they have gained experience in an area, their search is much more directed, and in some cases may disappear entirely. The domain of strategy learning is another field in which discrimination learning techniques can be applied. This results from the fact that search can be cast as the successive application of operators that transform one state into another, in the hopes of achieving some goal. The novice may possess the *legal* conditions for applying the operators in some task, but may lack the heuristically useful conditions under which those operators should be applied. The discovery of such useful conditions can be cast in a form analogous to that for finding the conditions on concepts. However, while the assignment of blame and credit is trivial in the concept learning paradigm, it becomes considerably more complex when search is involved.

Sleeman, Langley, and Mitchell (1982) have discussed an apparently general solution to the credit assignment in the context of strategy learning. If one is willing to find the solution to some problem by trial and error, or if a sample solution is provided by a tutor, then one can use the known solution path to determine the appropriateness of an operator as soon as it has been applied. Thus, one can easily distinguish between good and bad instances of an operator, and the task of learning the heuristically useful conditions under which these operators should be applied is reduced to a task very like that of learning concepts. Brazdil (1978) has employed this approach, along with a discrimination-like learning mechanism, to discover strategies from sample solutions, while Mitchell, Utgoff, and Banerji (1981) have explored strategy learning from self-generated solution paths, using Mitchell's (1977) version space technique. Langley (1982, 1983) has described these systems and their approach to strategy learning in greater detail.

Below we describe SAGE, an adaptive production system that operates in a similar fashion. The system that starts with legal operators for some task (stated as productions), finds a solution path by trial and error, and then attempts to learn from this solution path. Upon trying the same problem a second time, it can tell immediately when it has applied the incorrect operator. Such an error implies that the responsible rule is overly general, and discrimination is used to generate more conservative variants. This process continues (with relearned variants being strengthened) until the problem can be solved without errors. SAGE has many similarities to Brazdil's ELM and Mitchell, Utgoff, and Banerji's LEX, but there are some important differences as well. Like the concept learning program described in the previous section, SAGE is not intended as a detailed model of human strategy acquisition. Let us consider the system's behavior on a simple puzzle known as slide-jump.*

Table 1
Solution path for the four-coin slide-jump puzzle

Q Q _ N N	initial state
Q _ Q N N	slide a quarter from 2 to 3
Q N Q _ N	jump a nickel from 4 to 2
Q N Q N _	slide a nickel from 5 to 4
Q N _ N Q	jump a quarter from 3 to 5
_ N Q N Q	jump a quarter from 1 to 3
N _ Q N Q	slide a nickel from 2 to 1
N N Q _ Q	jump a nickel from 4 to 2
N N _ Q Q	slide a quarter from 3 to 4

The Slide-Jump Puzzle

Like many puzzles, the slide-jump task appears deceptively simple on the surface, but is fairly difficult for humans to solve. In this puzzle, one starts with equal numbers of two types of coins (e.g., quarters and nickels) set in a row. All quarters are on the left, all nickels are on the right, and the two sets are separated by a blank space. The goal is to interchange the positions of the quarters and nickels. However, quarters can move only to the right, while nickels can move only to the left. Two basic moves are allowed: a coin can *slide* from its current position into an adjacent blank space, or it can *jump* over a coin of the opposite type into a blank space.† Table 1 presents one solution path for the four-coin problem.

* SAGE has also learned useful heuristics for solving algebra problems in one variable, and for a seriation task in which blocks must be ordered according to their lengths. The system's behavior in these domains is discussed in Langley (1982, 1983); however, we will be using slide-jump as our main example in this discussion.

† A more complex version of the task allows one to jump over coins of the same type. Although this results in a larger search space, such moves always lead to dead ends, and we will consider only the simpler version of the task.

Figure 5 shows one half of the state space for the four-coin puzzle (the other half is simply a mirror image of that shown). Dead end states are represented by squares, and the solution path is shown in bold lines. We will refer to the numbers on the states later in the section. SAGE starts this task with one condition-action rule for sliding and one for jumping, as well as some additional rules which support the search for a solution. The initial rules are correct in that they propose only *legal* moves, but they lack conditions for distinguishing *good* moves from *bad* moves. As a result, the program makes many poor moves on its first pass, and is forced to back up whenever it reaches a dead end.

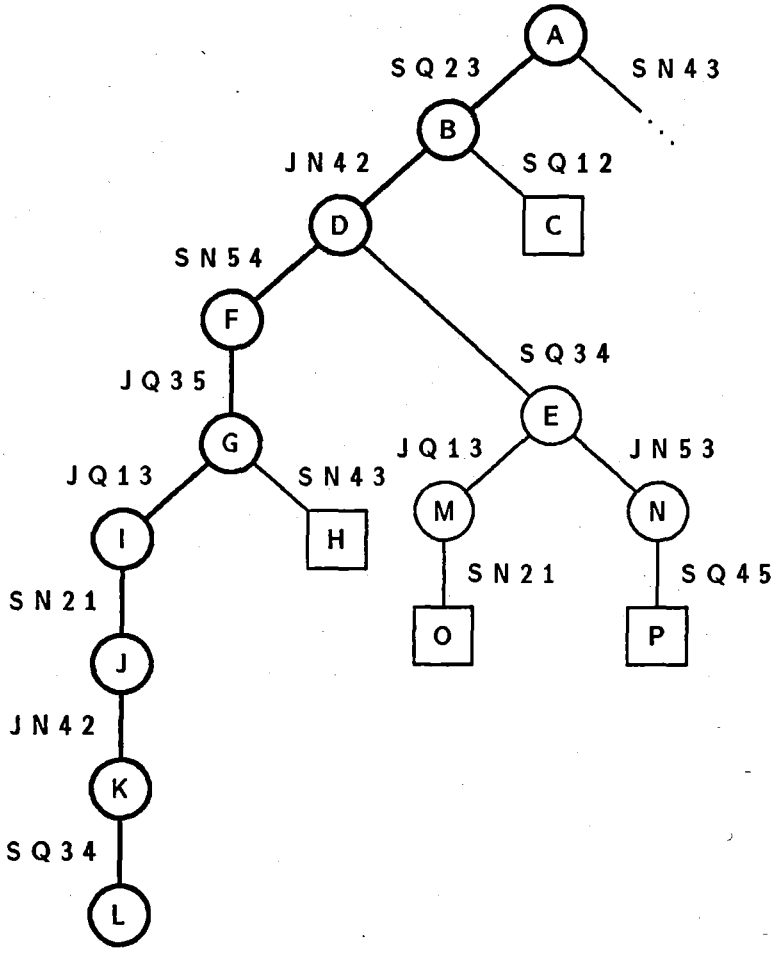

Figure 5. State space for the four-coin slide-jump puzzle.

However, SAGE does eventually solve the problem in this trial-and-error way. At this point, it attempts a second solution, but this time it has the initial solution to guide its search. When one of the move-proposing rules is incorrectly applied, the system knows its mistake immediately. If such an error occurs, SAGE calls on the discrimination process to compare the last correct application of the offending production to the current incorrect application. The resulting variants contain additional conditions that prevent them from firing in the undesired case. The program continues to learn in this fashion, constructing more conservative rules when errors are made and strengthening rules when they are relearned,

until it traverses the entire solution path with no mistakes.

The reader may wonder why the program should bother to improve its rules once it knows the solution to the puzzle. The reason is simple: there is a chance that these revised productions will also be useful in related problems for which the solution path is *not* available. As we shall see later, the heuristics SAGE learns on the four-coin puzzle let it solve the six-coin puzzle immediately, without backtracking.

Representing Problem States and Operators

Any problem solver must have some *representation* to work upon, as well as operators for transforming the *initial state* into the *goal state*. Although SAGE necessarily uses different representations for the different tasks it attempts, they all have one major feature in common: each problem state is represented as a number of *distinct* elements in working memory. SAGE's operators are implemented in such a way that they affect only some of these elements, so the entire problem state need not be respecified each time a move is made. Instead, those elements that have become true are added to memory, while those that are no longer true are removed. Also, this composite representation allows partial information about the problem state to be included as conditions on a move-proposing rule to the exclusion of less relevant information.

Table 2
Initial state for the four-coin slide-jump task

(quarter one)	(one is left of two)	(five is right of four)
(quarter two)	(two is left of three)	(four is right of three)
(blank three)	(three is left of four)	(three is right of two)
(nickel four)	(four is left of five)	(two is right of one)
(nickel five)	(nickel moves left)	(quarter moves right)

For example, consider the representation for the four-coin slide-jump puzzle. Table 2 presents the initial state for this task. The first column states that a quarter is in position one, another quarter is in position two, a blank is in position three, and so forth. Note that each of these facts is stored as a separate element. Thus, if SAGE slides a quarter from two to three, it simply removes the elements (quarter two) and (blank three), and adds the elements (blank two) and (quarter three); nothing else must be altered. The remaining columns state the spatial relations between the positions, along with the directions in which each coin may move; these facts remain unchanged throughout the problem.

The composite representation SAGE uses for problem states leads naturally to the rules for proposing moves between those states. These rules incorporate the legal conditions for applying an operator in their left hand sides, and include a proposal to apply that operator in their action sides. When one of these productions is selected, it inserts a goal to apply the associated operator; when this operator is applied, it removes some elements from working memory and adds others. For example, the basic *slide* production can be stated:

SLIDE-1
If a *coin* is in *position1*,
  and *position2* is to the *direction* of *position1*,
  and *position2* is blank,
  and that *coin* can move to the *direction*,
then consider sliding that *coin* from *position1* to *position2*.

This production will match* whenever a coin is next to the blank space and the coin is allowed to move in that direction; it will work for either coin and for any pair of adjacent positions. Once a goal has been set, another production is responsible for updating the problem state by actually adding and deleting elements from memory. This division of labor simplifies matters in later runs when the solution path is known, for an incorrect goal may be caught before it is implemented. Also, the goals provide a trace of what has been done, which can prove useful in stating more heuristics for directing the search process.

Controlling the Search

Before SAGE can learn from its mistakes, it must be able to identify those mistakes. The most obvious way to distinguish good moves from bad moves is by examining the correct solution path to a given problem.† And to find a solution using the very weak move-proposing rules it has at the outset, the system must search. In addition to rules for suggesting moves, effective search requires the ability to eventually recognize a fruitless path, and the ability to backtrack once such a path is recognized. SAGE carries out a form of *depth-first* search in each of the tasks it attempts, and learning consists of constructing more specific move-generating rules that direct the search down fruitful paths.

When SAGE decides it has reached a bad state or dead end, it backtracks to the previous state. This is possible only because the system keeps a trace of all previous moves made along the current path. This trace serves a second purpose in that it provides a record of the solution path once the goal has been reached. During the initial search, SAGE also remembers bad moves it has made along the current path, so that it can avoid making these moves a second time. However, this negative information is removed once the solution is reached.

* Note that although this rule proposes *legal* moves, there is no guarantee that these will be *good* moves. As stated, the slide-1 rule will generate many bad moves, and learning must occur before only useful actions are suggested. Also note that the variable *coin* will match against the symbol *quarter* or *nickel*, rather than against a particular coin.

† One can imagine other means of determining good instances of a rule from bad instances, such as Anzai's (1978a, 1978b) loop move and shorter path heuristics, and these certainly have a role to play in domains where search alone is prohibitive. However, in the current version of SAGE we have chosen to focus on techniques that rely on knowledge of the complete solution path.

SAGE employs a single production for recognizing the solution state. The rule for slide-jump may be paraphrased:

SOLVED
If a *coin1* moves to the *direction1*,
  and a *coin2* moves to the *direction2*,
  and *position* is blank,
  and no *coin1* is to the *direction2* of *position*,
  and no *coin2* is to the *direction1* of *position*,
then you are finished.

The goal-recognizing rules for other tasks are necessarily different, but they are always stated as single productions in much the same spirit as the above rule. Below we trace the details of SAGE's improvement on the slide-jump puzzle.

Learning Search Heuristics

SAGE was initially presented with the four-coin slide-jump task. After finding the solution path by depth-first search, a second attempt was made to solve the same problem. SAGE's first proposal was to slide a quarter from 2 to 3; this move is represented by the line connecting states A and B in Figure 5, where A stands for the initial state and B for the resulting state. Since this move lay upon the known solution path, the system implemented its plan and stored the latest instantiation of slide-1 as a good instance of that rule.* However, this was immediately followed by a proposal to slide a quarter from 1 to 2, which would have led to state C in the figure. Since this move lay off the known solution path, it was deemed an error and the discrimination routine was called to compare the instantiation of slide-1 that proposed this action to the good instantiation that had just been stored. In this case, the discrimination process discovered two differences, and so two variants of the slide-1 production were created. One of these was based on the fact that the correct move was the first that was made, while the incorrect one occurred in the context of an earlier move. The result was a production that would apply only on the first move of every task (the discriminant condition is shown in bold italics):

SLIDE-2
If a *coin* is in *position1*,
  and *position2* is to the *direction* of *position1*,
  and *position2* is blank,
  and that *coin* can move to the *direction*,
  and you have not made any previous moves,
then consider sliding that *coin* from *position1* to *position2*.

* The slide-jump task has two optimal solution paths which are "mirror images" of each other; the particular path is determined entirely by the initial move. To ensure that SAGE made the same first move on every run, a special production was included that increased the activation of part of the problem description. This was sufficient to focus the system's attention on the relevant coin and to avoid the effort of trying to distinguish between two equally good moves.

Unfortunately, this rule contains no information in its conditions to direct the search down useful paths, since only slide moves are possible from the initial state. The second variant was based on information about the particular type of move that had occurred before the bad slide move; the resulting production included more specific information in its new negated condition:

SLIDE-3
If a *coin* is in *position1*,
and *position2* is to the *direction* of *position1*,
and *position2* is blank,
and that *coin* can move to the *direction*,
and you did not just slide a *coin* **from** *position2* **to** *position3*,
then consider sliding that *coin* from *position1* to *position2*.

This second variant is more selective than slide-2; it would not have proposed moving a coin from position 1 to 2, provided that coin had just been moved from 2 to 3, and so it would have avoided the error produced by slide-1. In general, it would not slide a coin *into* a position which had just been vacated by another slide.

Still at state B, SAGE next considered jumping a nickel from 4 to 2; this agreed with its previous experience, so the suggestion was carried out, leading to the state labeled D in Figure 5. On the following move, slide-1 (which was still stronger than the two variants) proposed sliding a quarter from 3 to 4, which would have led off the path to state E. Again, the program decided it had made an error, and the responsible instantiation of slide-1 was compared to the same good instantiation that was used earlier (since no new good instances of the rule had occurred in the meantime). This time discrimination reproduced the variants slide-2 and slide-3, causing them to be strengthened; in addition, a new production was constructed, based on the jump that had just been made:

SLIDE-4
If a *coin* is in *position1*,
and *position2* is to the *direction* of *position1*,
and *position2* is blank,
and that *coin* can move to the *direction*,
and you did not just jump the *other* **coin from** *position2* **to** *position3*,
then consider sliding that *coin* from *position1* to *position2*.

This rule states that one should not slide a coin *into* a position from which one has just jumped the other brand of coin. Note that this production may still propose those moves avoided by slide-3 variant, while the earlier version may still propose this type of sliding move. In their current forms, both rules are overly general.

At this point SAGE considered sliding a nickel from 5 to 4 (still via the original rule slide-1), which would have led to state F in the figure. Since this agreed with the known path, the proposal was implemented, giving the system a new good instantiation of the responsible production to consider in subsequent discriminations. Next the system correctly jumped a quarter from 3 to 5 (leading to state G), but this was followed by a proposal to slide a nickel

from 4 to 3. This would have led off the solution path to state H, and so discrimination again searched for differences, this time generating a fourth variant:

SLIDE-5
If a *coin* is in *position1*,
and *position2* is to the *direction* of *position1*,
and *position2* is blank,
and that *coin* can move to the *direction*,
and you have just jumped a *coin* **from** *position2* **to** *position3*,
then consider sliding that *coin* from *position1* to *position2*.

Here we have a positive condition included in the variant, suggesting that slides should occur after a jumping spree has been completed, and in the same direction. In addition, discrimination produced four other less useful variants that made reference to moves earlier in the problem. By now SAGE had reached the halfway point for the problem. Since only one move was possible* at each step from this point onward, the program finished with no more mistakes. However, earlier errors had been made, so the system tried the problem a third time. In this run, identical mistakes occurred, and each of the variants was strengthened.

The program continued along these lines, until during the fifth run the second variant, slide-3, came to surpass the original rule in strength. After this, the more conservative production was applied whenever possible. When slide-3 proposed sliding a quarter from 3 to 4 while the system was at state D (recall that slide-1 made this same mistake earlier, leading slide-4 to be learned), discrimination resulted in a variant of slide-3 that contained yet another condition (shown in bold italics) for directing search down fruitful paths:

SLIDE-6
If a *coin* is in *position1*,
and *position2* is to the *direction* of *position1*,
and *position2* is blank,
and that *coin* can move to the *direction*,
and you did not just slide a *coin* from *position2* to *position3*,
and you did not just jump the *other* **coin from** *position1* **to** *position3*,
then consider sliding that *coin* from *position1* to *position2*.

This rule includes the negated conditions of both slide-3 and slide-4, stating that one should not propose a slide if *either* condition is met. Other variants† were created as well, but never gained sufficient strength to have any effect on the system's performance. After five more runs, the slide-6 rule acquired more strength than its precursor, and on its eleventh run, SAGE reached the goal state (marked L in the figure) without error. In addition, when

* Note that the variants slide-4 and slide-5 are *not* true on the slides required for the last half of the problem. For this reason, it is essential that the original slide rule remain available, and that the variants simply come to be preferred when competition occurs.

† Altogether, SAGE generated some 18 variants on the initial slide rule, but only 4 of these can be considered useful; fortunately, the strategy of giving variants low initial strengths and strengthening upon recreation was sufficient to focus attention on the more promising rules.

the system was presented with the six-coin task, the system successfully applied its heuristics to direct search, so that it solved the problem on its first attempt with no backtracking.

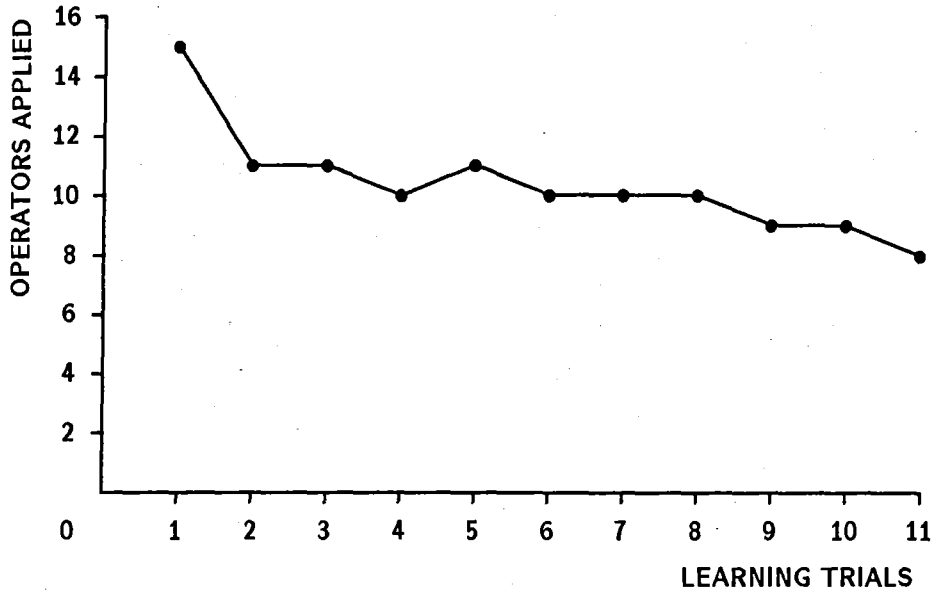

Figure 6. Learning curve for the slide-jump task.

Figure 6 presents SAGE's learning curve on the four-coin slide-jump task. The graph plots the number of moves that were considered before the solution was reached, against the order in which each run occurred. The drop between the first and second trials results from knowledge of the solution path, and so does not reflect the system's progress is developing useful heuristics. However, the slow improvement from the second trial onward shows that the system was gradually narrowing its search. And since the solution path was not needed in the eleventh and final run, it is appropriate to compare the number of moves on this trial to that on the first run. This comparison reveals that the variants on the slide-1 production, with their heuristically useful conditions, reduced the number of moves from the initial 15 to the optimum number 8.

Summary

In summary, SAGE is an adaptive production system that improves its search behavior by adding heuristically useful conditions to rules that initially contain only the legal conditions for applying operators. Presented with the slide-jump task, it solves the problem first by trial and error, and then attempts to learn from the solution sequence using the discrimination process. The system learns a number of useful variants, including one that allowed it to solve the four-coin and six-coin tasks with no errors. SAGE has also learned heuristics for solving algebra problems in one variable, and for ordering blocks according to their lengths. Although the performance rules for these cases differ, all versions of the system share a common core of adaptive productions that direct the learning process. Although SAGE is not intended as a model of human learning, it does shed light on one important way in which

strategies can be improved.

Future research with SAGE should concentrate on testing the system in additional domains. These tasks should include other puzzles for which the problem spaces are well understood, such as Tower of Hanoi or Missionaries and Cannibals. However, it is also important to test the learning system in domains involving many operators and large search spaces, such as symbolic integration or geometry theorem proving. The first of these is especially attractive, since it would allow direct comparison of SAGE's behavior to Mitchell, Utgoff, and Banerji's LEX system. In addition to further testing the program's generality, other methods should be explored for solving the credit assignment problem without resorting to complete solution paths. For example, one should be able to modify Anzai's loop move and shorter path heuristics so as to determine good and bad applications of an operator, and so allow discrimination learning to occur before a complete solution is found. Such an approach would be particularly useful for learning in domains with large problem spaces, where the search involved in finding a solution by trial and error would be prohibitive.

A Model of First Language Acquisition

Children do not learn language in an all-or-none fashion. They begin their linguistic careers uttering one word at a time, and slowly evolve through a number of stages, each containing more adult-like speech than the one before. In this section, we present a model that attempts to explain the regularities in children's early syntactic development. The model is called AMBER, an acronym for Acquisition Model Based on Error Recovery. AMBER learns a grammar by comparing its own utterances to those of adults and attempting to correct any errors, and discrimination plays an important role in the system's learning scheme. The model focuses on the omission of content words, the occurrence of telegraphic speech, and the order in which function words are mastered. Before considering AMBER in detail, let us first review some major features of child language, along with some earlier models of these phenomena.

Around the age of one year, the child begins to produce words in isolation, and continues this strategy for some months. At approximately 18 months, he starts to combine words into meaningful sequences. In order-based languages such as English, the child usually follows the adult order. Initially only pairs of words are produced, but these are followed by three-word and later by four-word utterances. The simple sentences occurring in this stage consist almost entirely of content words, while grammatical morphemes such as tense endings and prepositions are largely absent. During the period from about 24 to 40 months, the child masters the grammatical morphemes that were absent during the previous stage. These "function words" are learned gradually; the time between the initial production of a morpheme and its mastery may be as long as 16 months. Brown (1973) has examined the order in which 14 English morphemes are acquired, finding the order of acquisition to be remarkably consistent across children. In addition, those morphemes with simpler meanings and involved in fewer transformations were learned earlier than more complex ones. These findings place some strong constraints on the learning mechanisms one postulates for morpheme acquisition.

Although language learning has been a popular topic in artificial intelligence, only a few researchers have attempted to construct plausible models of the child's learning process. For example, Kelley (1967) has reported on a system that modeled the omission of content words and the gradual lengthening of utterances. However, in order to move between stages, the program had be told about new syntactic classes by the programmer, making it less than satisfactory as a model of first language learning. Selfridge's CHILD (1981) was much more robust than Kelley's program, and is unique in modeling children's use of nonlinguistic cues for understanding. However, CHILD's explanation of the omission of content words – that those words are not yet known – was implausible, since children often omit words that they have used in previous utterances. Reeker's PST (1976) explained this phenomenon through a limited memory hypothesis, which is consistent with our knowledge of children's memory skills. Still, PST included no model of the process through which memory improved; in order to let the system master more complex constructions, Reeker would have had to increase the system's memory size himself. Both CHILD and PST learned relatively slowly, and made mistakes of the general type observed with children. Both systems learned through a process of error recovery, starting off as abominable language users, but getting progressively better with time. This is a promising approach, and in this section we develop it in its extreme form.

An Overview of AMBER

Although Reeker's PST and Selfridge's CHILD address the transition from one-word to multi-word utterances, problems exist with both accounts. Neither of these programs focus on the acquisition of function words, and their explanations of content word omissions leave something to be desired. In response to these limitations, the goals of the current research are:

- Account for the omission of content words, and the eventual recovery from such omissions.
- Account for the omission of function words, and the order in which these morphemes are mastered.
- Account for the gradual nature of both these linguistic developments.

In this section we provide an overview of AMBER, a model that provides one set of answers to these questions. Since more is known about children's utterances than their ability to understand the utterances of others, AMBER models the learning of generation strategies, rather than strategies for understanding language.

Like both Selfridge's and Reeker's models, AMBER learns through a process of error recovery.* The model is presented with three pieces of information: a legal sentence, an event to be expressed, and a main goal or topic of the sentence. An event is represented

* In spirit, AMBER is very similar to Reeker's model, though they differ in many details. Historically, PST had no impact on the development of AMBER. The initial plans for AMBER arose from discussions with John R. Anderson in the fall of 1979, while we did not become aware of Reeker's work until the fall of 1980.

as a semantic network, using relations like agent, action, object, size, color, and type. The specification of one of the nodes as the main topic allows the system to restate the network as a tree structure, and it is from this tree that AMBER generates a sentence. If this sentence is identical to the sample sentence, no learning is required. If a disagreement between the two sentences is found, AMBER modifies its set of rules in an attempt to avoid similar errors in the future, and the system moves on to the next example.

AMBER's performance system is stated as a set of productions that operates on the goal tree to produce utterances. Although the model starts with the *potential* for producing (unordered) telegraphic sentences, it can initially generate only one word at a time. To see why this occurs, we must consider the three productions that make up AMBER's initial performance system. The first of these (the *start* rule) is responsible for establishing subgoals. Matching first against the main goal node, it selects one of the nodes below it in the tree and creates a subgoal to describe that node. This rule continues to establish lower level goals until a terminal node is reached. At this point, a second production (the *speaking* rule) retrieves the word for the concept to be described, and actually says the word. Once this has been done, a third rule (the *stop* rule) marks the terminal goal as satisfied. Moreover, since the stop rule is stronger than the start rule (which would like to create another subgoal), it marks each of the active goals as satisfied, and AMBER halts after uttering a single word. Thus, although the model starts with the potential for producing multi-word utterances, it must learn additional rules (and make them stronger than the stop rule) before it can generate multiple content words in the correct order.

In general, AMBER learns by comparing adult sentences to the sentences it would produce in the same situations. These predictions reveal two types of mistakes – errors of *omission* and errors of *commission*. Below we discuss AMBER's response to errors of omission, since these are the first to occur and thus lead to the system's first steps beyond the one-word stage. We consider the omission of content words first, and then the omission of grammatical morphemes. Finally, we discuss the importance of errors of commission in discovering conditions on the production of morphemes, and it is here that our theory of discrimination will be required.

Learning Preferences and Orders

AMBER's initial self-modifications result from the failure to predict content words. Given its initial ability to say one word at a time, the system can make two types of content word omissions – it can fail to predict a word *before* a correctly predicted one, or it can omit a word *after* a correctly predicted one. Rather different rules are created in each case. For example, imagine that Daddy is bouncing a ball, and suppose that AMBER predicted only the word "ball", while hearing the sentence "Daddy is bounce ing the ball". In this case, the system notes that it omitted the content word "Daddy" before the content word "ball", and an agent production is created:

AGENT
If you want to describe *event1*,
  and *agent1* is the agent of *event1*,
then describe *agent1*.

A similar rule for describing actions results from the omitted "bounce". Note that the above production does *not* give AMBER the ability to say more than one word at a time. It merely increases the likelihood that the program will describe the agent of an event instead of the action or the object.

However, as AMBER begins to prefer agents to actions and actions to objects, the probability of the second type of error (omitting a word after a correctly predicted one) increases. For example, suppose that Daddy is again bouncing a ball, and the system says "Daddy" while it hears "Daddy is bounce ing the ball". In this case, a slightly different production is created that is responsible for *ordering* the creation of goals. Since the agent relation was described but the object was omitted, an agent-object rule is constructed:

AGENT-OBJECT
If you want to describe *event1*,
  and *agent1* is the agent of *event1*,
  and you have described *agent1*,
  and *object1* is the object of *event1*,
then describe *object1*.

Together with the agent rule shown above, this production lets AMBER produce utterances such as "Daddy ball". Thus, the model provides a simple explanation of why children omit some content word in their early multi-word utterances. Such rules must be constructed many times before they become strong enough to have an effect, but eventually they let the system produce sentences containing all relevant content words in the standard order and lacking only grammatical morphemes.

Learning Suffixes and Prefixes

Once AMBER begins to correctly predict content words, it can learn rules for saying grammatical morphemes as well. As with content words, such rules are created when the system hears a morpheme but fails to predict it in that position. For example, suppose the program hears "Daddy * is bounce ing * the ball",* but predicts only "Daddy bounce ball". In this case, the following rule is generated:

* Asterisks represent pauses in the adult sentence. These cues are necessary for AMBER to decide that a morpheme like "is" is a prefix for "bounce" instead of a suffix for "Daddy". Although adults tend to speak slowly when addressing children, it is probably too much to assume that pause information is actually available to children learning their first language. Thus, AMBER's reliance on pauses must be viewed as a limitation of the current system that should be overcome in future versions.

ING-1
If you have described *action1*,
  and *action1* is the action of *event1*,
then say ING.

As stated, this production is overly general and will lead to errors of commission. We consider AMBER's response to such errors in the following section.

The omission of prefixes leads to very similar rules. In the above example, the morpheme "is" was omitted before "bounce", leading to the creation of a prefix rule:

IS-1
If you want to describe *action1*,
  and *action1* is the action of *event1*,
then say IS.

Note that this rule will fire *before* the action has been described, while the rule ing-1 can apply only *after* the goal to describe the action has been satisfied. AMBER uses such conditions to control the order in which morphemes are produced.

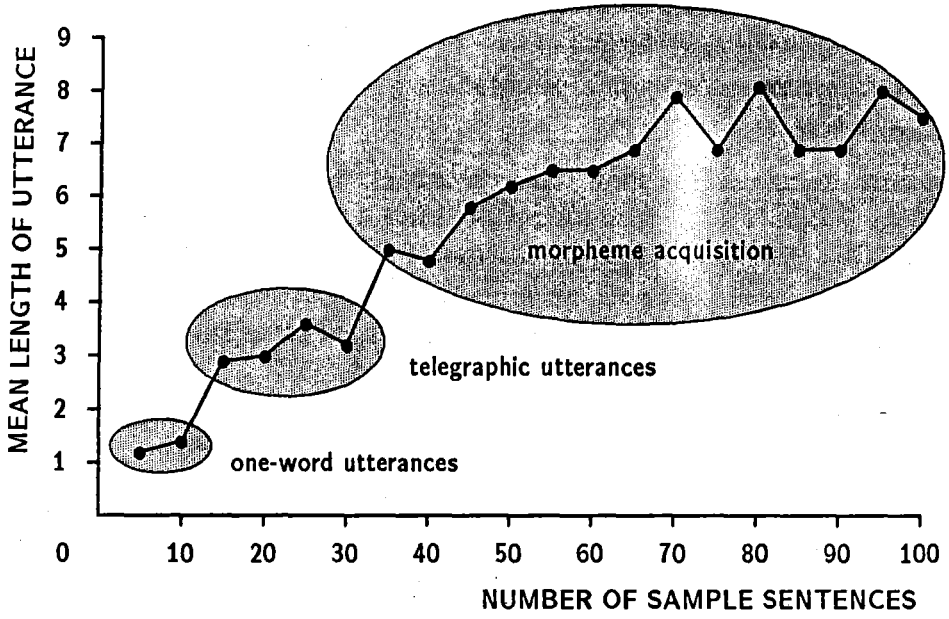

Figure 7. Mean length of AMBER's utterances.

Figure 7 shows AMBER's mean length of utterance as a function of the number of sample sentences (taken in groups of five) seen by the program. As one would expect, the system starts with an average of around one word per utterance, and the length slowly increases with time. AMBER moves through a two-word and then a three-word stage, until it eventually produces sentences lacking only grammatical morphemes. Finally, the morphemes are included, and adult-like sentences are produced. The temporal positions of these three stages are shown by the successive regions in the figure; although the stage boundaries are

not so distinct as indicated, the regions should give some of idea of the relative time the model spends in mastering various aspects of syntax. The incremental nature of the learning curve results from the piecemeal way in which AMBER learns rules for producing sentences, and from the system's reliance on strengthening.

Recovering from Errors of Commission

Errors of commission occur when AMBER predicts a morpheme that does not occur in the adult sentence. These errors result from the overly general prefix and suffix rules that we saw in the last section. In response to such errors, AMBER calls on the discrimination routine in an attempt to generate more conservative productions with additional conditions.* Earlier, we considered a rule (is-1) for producing "is" before the action of an event. As stated, this rule would apply in inappropriate situations as well as correct ones. Suppose that AMBER learned this rule in the context of the sentence "Daddy *is* bounce ing the ball". Now suppose the system later uses this rule to predict the same sentence, but that it instead hears the sentence "Daddy *was* bounce ing the ball".

At this point, AMBER retrieves the rule responsible for predicting "is" and lowers its strength; it also retrieves the situation that led to the faulty application, passing this information to the discrimination routine. Comparing the earlier good case to the current bad case, the discrimination mechanism finds only one difference – in the good example, the action node was marked *present*, while no such marker occurred during the faulty application. The result is a new production that is identical to the original rule, except that a present condition has been included:

IS-2
If you want to describe *action1*,
  and *action1* is the action of *event1*,
  and *action1* is in the present,
then say IS.

This new condition will let the variant rule fire only when the action is marked as occurring in the present. When first created, the is-2 production is too weak to be seriously considered. However, as it is learned again and again, it will eventually come to mask its predecessor. This transition is aided by the weakening of the faulty is-1 rule each time it leads to an error.

Once the variant production has gained enough strength to apply, it will produce its own errors of commission. For example, suppose AMBER uses the is-2 rule to predict "The boy s *is* bounce ing the ball", while the system hears "The boy s *are* bounce ing the ball". This time the difference is more complicated. The fact that the action had an agent in the good situation is no help, since an agent was present during the faulty firing as well. However, the agent was *singular* in the first case but not during the second. Accordingly,

* Anderson's ALAS (1981) system uses a very similar process to recover from overly general morpheme rules. AMBER and ALAS have much in common, both having grown out of discussions between Anderson and the author. Although there is considerable overlap, ALAS generally accounts for later developments in children's speech than does AMBER.

the discrimination mechanism creates a second variant:

IS-3
If you want to describe *action1*,
  and *action1* is the action of *event1*,
  and *action1* is in the present,
  and *agent1* is the agent of *event1*,
  and *agent1* is singular,
then say IS.

The resulting rule contains *two* additional conditions, since the learning process was forced to chain through two elements to find a difference. Together, these conditions keep the production from saying the morpheme "is" unless the agent of the current action is singular in number.

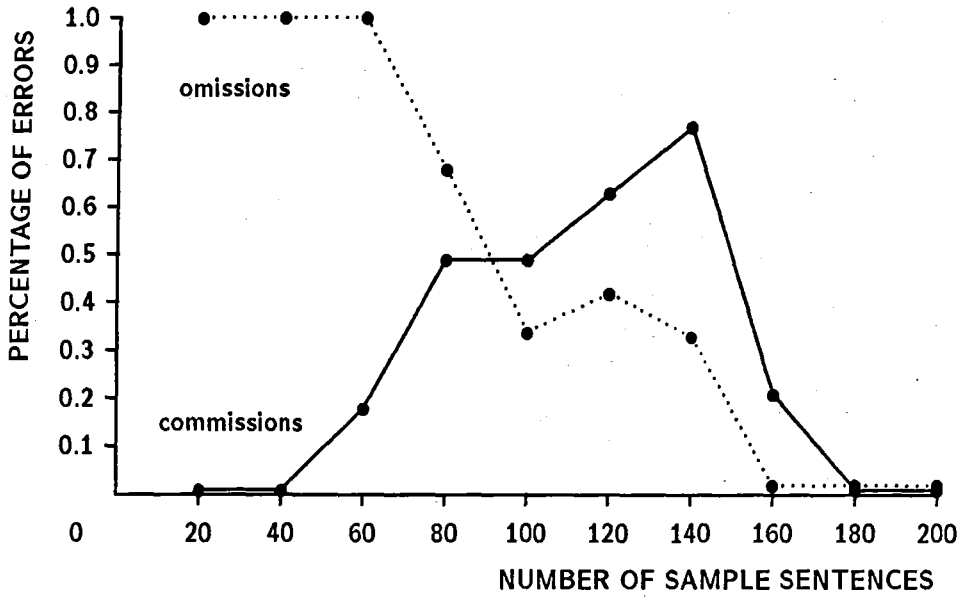

Figure 8. AMBER's learning curve for the morpheme "ing".

Note that since the discrimination process must learn these sets of conditions separately, an important prediction results: *the more complex the conditions on a morpheme's use, the longer it will take to master.* For example, multiple conditions are required for the "is" rule, while only one condition is needed for the "ing" production. As a result, the former is mastered after the latter, just as found in children's speech. Thus, the discrimination process provides an elegant explanation for the observed correlation between a morpheme's semantic complexity and its order of acquisition. As we noted earlier, a generalization-based learning system would master more complex rules before learning simpler ones, and so would predict exactly the opposite of the observed correlation.

Unfortunately, parents seldom present sentences in an ideal order, so that the relevant conditions are obvious to the child. In order to more closely model the environment in which

children learn language, AMBER was presented with randomly generated sentence/meaning pairs. Thus, it was usually impossible to determine *the* correct discrimination that should be made from a single pair of good and bad situations. As in other domains, the strategy of strengthening relearned rules and weakening faulty versions was sufficient to focus attention on useful variants. Figure 8 presents the learning curves for the "ing" morpheme. Since AMBER initially lacks an "ing" rule, errors of omission abound at the outset, but as the "ing" rule and its variants are strengthened, these errors decrease. In contrast, errors of commission are absent at the beginning, since AMBER lacks an "ing" rule to make false predictions. As the morpheme rule becomes stronger, errors of commission grow to a peak, but they disappear as discrimination takes effect.

Summary

In conclusion, AMBER provides explanations for several important phenomena observed in children's early speech. The system accounts for the one-word stage and the child's transition to the telegraphic stage. Although AMBER and children eventually learn to produce all relevant content words, both pass through a stage where some are omitted. Because it learns sets of conditions one at a time, the discrimination process explains the order in which grammatical morphemes are mastered. Finally, AMBER learns *gradually* enough to provide a plausible explanation of the incremental nature of first language acquisition. Thus the system constitutes a significant addition to our knowledge of syntactic development.

Of course, AMBER has a number of limitations that should be addressed in future research. For example, the current model cannot master irregular constructions like "ate" and "feet", and this is another area where discrimination would seem to have an important role to play in recovering from overgeneralizations. Another application of this learning mechanism lies in the acquisition of word order, since there is no reason in principle why the conditions on rules such as the agent-object production could not be learned through a discrimination-like process. The solutions to other limitations are not so apparent. For instance, the existing program models only the earliest stages of language acquisition, and major extensions may be necessary to replicate later developments. As it stands, AMBER says nothing about the relation between generation and comprehension, and the model's strategies for error recovery have only been used to learn rules for generation. Finally, the system has been tested only on English, and it is not clear how relevant the model's learning mechanisms will be for languages in which word order plays a less important role. But despite these limitations, AMBER has helped to clarify the incremental nature of language acquisition, and future versions should further our understanding of this complex process.

Modeling Cognitive Development

A final area of application lies in the domain of cognitive development. Most researchers in this field have been strongly influenced by the early work of Piaget (1969). Although the majority of research in cognitive development has been experimentally oriented, recent efforts have been made to explain some developmental trends in information processing terms. Production systems have been a popular framework for modeling behavior at different

stages, partially because their modularity allows the statement of successive models that differ by only one or two productions. For example, Baylor, Gascon, Lemoyne, and Pother (1973) have constructed production system models of children at various stages on Piaget's weight seriation task, and Young (1976) has devised similar models for the related length seriation task. The most comprehensive work along these lines has been carried out by Klahr and Wallace (1976), who have proposed production system models of children's behavior at various stages on tasks ranging from quantification to class inclusion. The next step is to construct models of the transition process that account for the movement between these stages. These models can be stated as adaptive production systems, and it is here that our theory of discrimination learning has an important role to play.

Table 3

Klahr and Siegler's model of behavior on the balance scale task

STAGE 1

P1
If you have a balance with *side*,
and the weights are equal,
then predict the sides will balance.

P2
If you have a balance with *side*,
and *side* has the greater weight,
then predict *side* will go down.

STAGE 2

P3
If you have a balance with *side*,
and the weights are equal,
and *side* has the greater distance,
then predict *side* will go down.

STAGE 3

P4
If you have a balance with *side*,
and *side* has the greater weight,
and *side* has the lesser distance,
then muddle through.

P5
If you have a balance with *side*,
and *side* has the greater weight,
and *side* has the greater distance,
then predict *side* will go down.

The Balance Scale Task

Klahr and Siegler (1978) have studied children's development on a variant of Piaget's balance scale task. In this task, the child is presented with a two-arm balance, with several pegs spaced evenly along each arm. Small disks of equal weight are placed on the pegs (only one peg on each side has weights), and the child is asked to predict the direction in which

the scale will move when released. The standard method for correctly making this prediction involves the notion of torque. The number of weights on a peg is multiplied by that peg's distance from the center. If one side has the greater product, that side will go down; if the products are equal, the scale will remain balanced.

Despite the apparent simplicity of this rule, Klahr and Siegler found their subjects using quite different prediction schemes. Also, these schemes differed systematically with the age of the child. Four basic stages appeared to exist, and the researchers successfully modeled each stage as a simple production system. Table 3 presents the productions used to model the first three stages, paraphrased in English. Since each successive stage differs from the previous model by the inclusion of only one or two new productions, we have grouped the rules according to the stage at which they were introduced. We have omitted the productions required for the final stage, which include rules for computing torque and responding appropriately.

The model of the initial stage consists of two simple productions. The first rule, P1, predicts that the scales will balance if the weights are the same on both sides. Similarly, P2 predicts that if the weights are different, the side with the greater weight will go down. The model of the second stage requires only one new production, P3, which predicts that if the weights are equal but the distances differ, the side with the greater distance will descend. Since Klahr and Siegler were assuming that conflicts between productions would be decided in favor of specific rules over more general ones, the new production would be preferred to P1 whenever both matched. However, P1 would still be selected in cases where both weights and distances were equal (since P3 would not match in this situation), correctly predicting a balance. Modeling third stage behavior required two new rules, labeled P4 and P5 in the table.* The first of these applies only when the weight and distance cues conflict. Klahr and Siegler called these "conflict problems", and they were especially difficult for the subjects. Accordingly, the action associated with this rule is to "muddle through", which seemed to be a way of incorporating random behavior into the model. The final rule, P5, applied whenever the weight and distance cues were greater on the same side, and predicted that side would go down.

Although Klahr and Siegler's models were very simple, they accounted for much of the variance observed in children's behavior on the balance scale task. As we have noted, successive models differed by only one or two productions. Thus, while the authors did not propose a mechanism to account for the transition between these stages, their analysis must be viewed as taking us a major step in that direction. Below we present a slightly revised stage model that brings us even closer to that goal.

* Actually, it is not clear that the second of these was necessary, since P2 would still make the correct prediction if P5 were absent.

Table 4

Revised model of behavior on the balance scale task

RANDOM STAGE	
BALANCE-1 If you have a balance with *side*, then predict the sides will balance.	DOWN-1 If you have a balance with *side*, then predict *side* will go down.
STAGE 1	
BALANCE-2 If you have a balance with *side*, and the weights are equal, then predict the sides will balance.	DOWN-2 If you have a balance with *side*, and *side* has the greater weight, then predict *side* will go down.
STAGE 2	
BALANCE-3 If you have a balance with *side*, and the weights are equal, and the distances are equal, then predict the sides will balance.	DOWN-3 If you have a balance with *side*, and the weights are equal, and *side* has the greater distance, then predict *side* will go down.
STAGE 3	
DOWN-4 If you have a balance with *side*, and *side* has the greater weight, and the distances are equal, then predict *side* will go down.	DOWN-5 If you have a balance with *side*, and *side* has the greater weight, and *side* has the greater distance, then predict *side* will go down.

A Revised Stage Model

Table 4 summarizes our revised model of successive stages on the balance scale task. In addition to the three stages shown in Table 3, we have also included a random stage. Taken together, the first pair of productions (BALANCE-1 and DOWN-1) randomly predict that one of the two sides will go down, or that the sides are balanced.* Klahr and Siegler found no evidence for this stage, presumably because their subjects had moved beyond the random

* Since both productions can match the variable *side* against either side, they will each be selected half the time (on the average) during the random stage. Thus, the system would predict that the left side would go down a quarter of the trials, and would predict that the right side would descend the same fraction of the time.

strategy at an earlier age. When the second pair of rules (BALANCE-2 and DOWN-2) is added to the first pair, the resulting system behaves exactly as Klahr and Siegler's model of Stage 1, *provided* the new rules are stronger than the first pair (and so will always be preferred, since we are assuming PRISM's conflict resolution principles). Upon adding the third pair of productions (BALANCE-3 and DOWN-3), we have a model of Stage 2 behavior, provided that both of these rules are stronger than the production BALANCE-2. Finally, when the productions DOWN-4 and DOWN-5 are inserted (and made stronger than DOWN-2), we have a partial model of Klahr and Siegler's Stage 3.

To complete the model of this stage, the system must have some way of "muddling through" on conflict problems. Klahr and Siegler have made a useful distinction between three types of conflict problems. For conflict situations in which the side with greater weight descended, they used the term "conflict weight" problems. Similarly, conflict problems in which the side with greater distance went down were labeled "conflict distance" problems, and conflicts that resulted in a balance were called "conflict balance" problems. This analysis suggests that we can model "muddling" behavior by the introduction of three additional rules:

DOWN-6
If you have a balance with *side*,
  and *side* has the greater weight,
  and *side* has the lesser distance,
then predict *side* will go down.

DOWN-7
If you have a balance with *side*,
  and *side* has the greater distance,
  and *side* has the lesser weight,
then predict *side* will go down.

BALANCE-4
If you have a balance with *side*,
  and *side* has the greater weight,
  and *side* has the lesser distance,
then predict the sides will balance.

The condition sides of each of these rules are functionally equivalent, since they will each match on all three types of conflict problem. However, DOWN-6 will make correct predictions only on conflict weight problems, and will make the wrong prediction in the other two situations. Similarly, DOWN-7 will be correct only on conflict distance problems, and BALANCE-4 will be correct only on conflict balance tasks. If we assume that these three productions have equal strengths, then none will be preferred over another and one will be selected at random, giving "muddling through" behavior. This explanation provides a more detailed account of the subjects' uncertainty on conflict problems than that given by Klahr and Siegler's single P4 rule with its "muddle through" action.

It is worthwhile to examine the relations between these two sets of models. The productions included in both models of Stage 1 are equivalent except for insignificant representational differences. However, Klahr and Siegler's model of Stage 2 behavior requires the addition of only one production, while our version includes two rules. One of these rules, DOWN-3, is equivalent to P3, and fulfills a similar function in our model to that served by P3 in the other system. However, Klahr and Siegler's model of Stage 3 requires no analog to our BALANCE-3, since P1 handles all situations in which the weights are the same (except for those covered by P3). Although our conflict resolution scheme relies on strength rather than specificity, we could make DOWN-3 stronger than BALANCE-2 to achieve a similar effect, but it will shortly become apparent why we have chosen to include the BALANCE-3 production. A similar relationship holds between Klahr and Siegler's P5 and our rules DOWN-4 and DOWN-5 for Stage 3; as before, they managed to get by with one rule where we have used two, again through use of their specificity principle for conflict resolution. In this case, P5 corresponds to DOWN-5, while situations matched by DOWN-4 are handled by P2. Finally, the "muddle through" production P4 in the earlier model is replaced in our model by three rules with identical conditions but different actions; this is simply a more detailed way to describe the system's uncertainty on conflict problems.

At first glance, one is tempted to judge Klahr and Siegler's approach as more elegant than ours, since their model of Stage 3 contains only five productions and ours contains eleven. Even if we replace our three conflict rules with a single production analogous to P4, our final model would still consist of nine productions. However, our revised models have one characteristic that is lacking in Klahr and Siegler's set: *rules occurring in later stages are always discriminant versions of rules that have occurred in an earlier stage.* This feature lets us employ our theory of discrimination learning to account for the transition process from the random to the final stage shown in Table 4, since this mechanism generates just such discriminant versions in its attempt to recover from errors. Although this feature held for some of Klahr and Siegler's rules (e.g., P5 is a variant on P2), it was by no means true of them all, and this is perhaps one reason why they never proposed a model of the transition process. Note that when we say that one rule is a "discriminant version" of another, we mean more than just the fact that the first rule has conditions that are a special case of the second; we also mean that the action sides of the two rules are the same. Thus, P3 is a special case of P1 in the earlier model, but it is not a discriminant version of P1.*

* The goal of stating all successive rules as discriminants of earlier rules was one of the reasons for including an initial random stage (in addition to its intrinsic plausibility). While the productions BALANCE-3, DOWN-4, and DOWN-5 are all direct variants on the two rules from Stage 1, the DOWN-3 rule is not. In order to account for its origin, and for the origin of the Stage 1 rules, a random stage had to be included.

A Model of the Transition Process

As we have noted, each of the rules in our revised stage model is a variant on one of the rules occurring at some earlier stage, and we have implemented an adaptive production system model of learning on the balance scale task that takes advantage of this insight.* The model begins with the rules BALANCE-1 and DOWN-1, which provide the initial behavior on which the learning is based. In addition, the system contains a production for comparing the sides of the balance scale on dimensions like weight and distance, since the relative weights and distances must be present in memory if discrimination is to discover conditions referring to such relations. Finally, the model includes a production for noting when a correct prediction has been made and storing credit with the responsible rule, and a similar production for evoking the discrimination process when an incorrect prediction is made. Thus, this learning program is nearly identical to the concept learning system we described in an earlier section. One difference is that the current model does not require a designating production to create new rules, since errors of omission never occur. Also, no default rule is required to make negative predictions, since some sort of positive action is always appropriate.

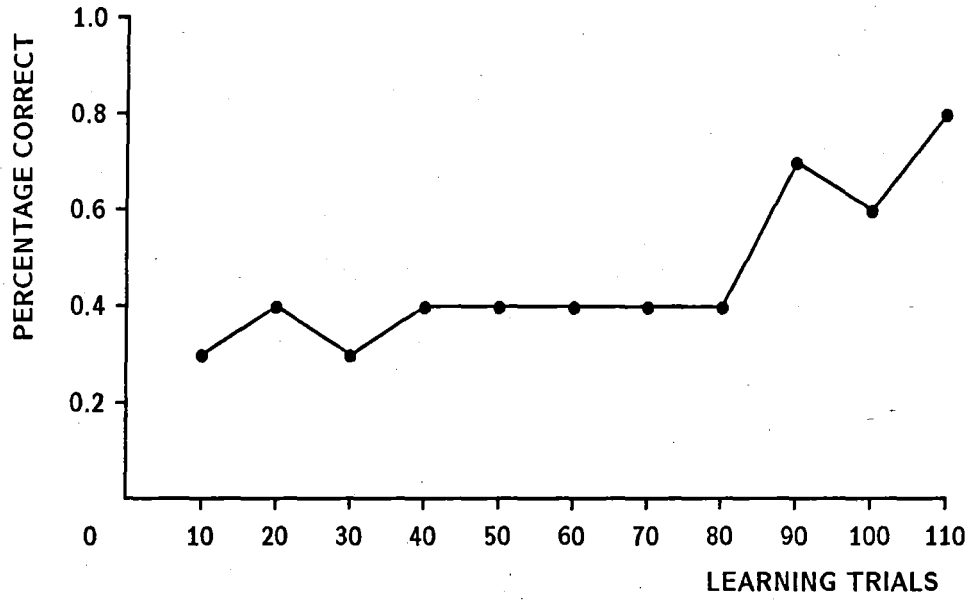

Figure 9. Learning curve for the balance scale task.

The program was presented with problems selected randomly from seven basic problem types. These included problems in which only the weights differed, in which only the

* The reason we have not attempted to model Stage 4 behavior should now be apparent. The acquisition of a torque rule requires the introduction of an entirely new concept, while the discrimination process can only be used to find conditions on existing concepts. At least in its current form, our discrimination theory cannot account for the manner in which the torque concept is acquired.

distances differed, in which both weights and distances were equal, in which the two cues agreed, and the three types of conflict problems discussed above. Figure 9 summarizes the model's errors as a function of time (in units of 10 trials). Since the system begins with the two random rules BALANCE-1 and DOWN-1, and since there are three basic predictions to choose from (left down, balance, and right down), one would expect about 33 percent of the initial predictions to be correct, and this is approximately what we find.* By trial 100, the system has learned (and sufficiently strengthened) the Stage 2 and Stage 3 rules shown in Table 4, so that it makes correct predictions on all but the three conflict problems, giving a success rate of approximately 60 percent. In the case of conflict problems, the model's representation of the environment (consisting only of information about relative weights and distances) is incapable of even *stating* the torque rule that would correctly predict results in a consistent manner. In other words, the program's representation of the problem is inherently incomplete. However, the discrimination process is sufficiently robust to learn useful rules despite this limitation, and the system arrives at a set of rules that make correct predictions much of the time, just as children do before they are taught the torque rule.

This brings another important feature of of our theory to light: discrimination learning allows one to learn useful rules *even if one's representation is ultimately inadequate.* Since our system has no notion of torque, it can never fully understand the balance scale task, yet it does learn rules that lead to correct predictions in many cases. Since one can never guarantee that a representation is optimal, this is a powerful feature that would be advantageous to any learning system. This capability to learn useful rules despite incomplete representations is as fully important a characteristic of the discrimination process as its ability to learn in the presence of noise, and we are unaware of any other approach to learning that has this capability.

Let us trace the transition model's behavior on the troublesome conflict problems, since it reveals some interesting details about the discrimination process. The program begins by invoking either BALANCE-1 or DOWN-1 on these problems, sometimes predicting the correct outcome and sometimes making an error. When errors occur, the discrimination mechanism is called in an attempt to determine the conditions under which the faulty prediction should be made. If the system bases its learning on a correct prediction made by DOWN-1 that was made on a conflict weight problem (in which the side with greater weight goes down), the discrimination process would construct as a variant the DOWN-2 rule shown in Table 4 (though this is learned in other contexts as well). For correct applications of DOWN-1 on conflict distance problems, the system would create the variant DOWN-2.5 (shown below), and for conflict balance problems, it would construct a variant of BALANCE-1 that contained either a greater weight or a greater distance constraint in its condition side.

* Actually, calculation of the initial probabilities is more complicated than this. One reason for this is that the system initially predicts a balance response 50 percent of the time, while it predicts right down and left down each 25 percent of the time. Another complication is that only two of the problem types lead to balanced scales, while five lead to one side or the other descending. However, the overall probabilities are close enough to 33 percent initially correct responses that we will not consider them further.

However, none of these variants will consistently be correct, and soon after they gain sufficient strength, these rules will lead to errors and generate variants of their own. The result will be the three productions included in our stage model – DOWN-6, DOWN-7, and BALANCE-4 – each containing conditions that match when one side has greater weight and the other has greater distance, but differing as to the predictions they make. One of these will always be correct on conflict weight problems, another on conflict distance problems, and the third on conflict balance problems, but none will be correct on all three. Thus, they will continually be weakened and then strengthened, and the system's preference on conflict problems will oscillate between them. This effect is very similar to behavior that Klahr and Siegler observed in one of their subjects, who seemed to switch back and forth between weight and distance cues whenever the use of one led to an incorrect prediction on a conflict problem. However, their detailed model of the subject included a somewhat ad hoc mechanism for explaining this shift, while our model reacts in a similar fashion (though not in exactly the same manner) purely as a byproduct of learning through discrimination. The long-term result is "muddle through" behavior, in which different cues are used to make the decision at different times.

What is the exact relation between the stage model shown in Table 4 and the transition model's learning path? Although the system creates all the variants given in the table* in the specified order, it also learns other rules, and so does not follow the exact stage sequence observed by the earlier researchers. For example, at approximately the same time that it constructs DOWN-2, the model also creates the following production:

DOWN-2.5
If you have a balance with *side*,
  and *side* has the greater distance,
then predict *side* will go down.

This rule is identical to DOWN-2, but includes a condition about greater distances instead of relying on greater weights. As it is implemented, there is no reason for the discrimination mechanism to prefer conditions about the weight to conditions about the distance. Unfortunately, Klahr and Siegler found evidence for DOWN-2 (or P2, its equivalent), but no evidence for the above rule. Apparently, children tend to focus more on weights than on distances, and our theory of discrimination learning has no way to account for such trends. One can imagine introducing preferences into the model to focus attention on some attributes in favor of others, but unless one can explain where these preferences originated, they would provide no more explanation than labeling one dimension as more "salient" than another. Thus, our transition model does not account for the details of Klahr and Siegler's results, though it does provide a plausible initial explanation of the transition between the observed stages.

---

* The reader may have noticed that the rules DOWN-3, DOWN-4, and DOWN-5 are all correct in the sense that they never lead to errors, and are all variants of the original DOWN-1 production. Thus, the balance scale task provides an example of another domain where the learning of disjunctive rules is required, and which our approach to discrimination learning can handle adequately.

Two additional drawbacks of the model are closely intertwined. One of these is the speed with which the system achieves maximum predictive power. The program arrives at Stage 3 behavior within a hundred trials, and using only 32 CPU seconds on a PDP-10 computer. In contrast, children take years to move from Stage 1 to Stage 3 behavior. A second difficulty is that by presenting the system only with information about weights and distances, we are telling it much about the space of rules it should search during the learning process. However, these two limitations tend to cancel each other out. Clearly, the model learns so quickly precisely because we have presented it with so few alternatives to consider. Other idealizations of the task environment contribute further to the rapid learning. For example, the child is certain to be distracted by other events in his environment, while the model has no analogous attention stealers.

Idealizations of this type have a long and respectable history in science, dating from the time Galileo decided to ignore such annoying factors as air resistance. Thus, the model can be viewed as a useful approximation of the actual situation, which we could complicate to achieve more accurate results if this were deemed worth the effort that would be involved. One might question whether the discrimination theory would still be useful in a more detailed model that had to deal with many irrelevant dimensions. However, we have seen in earlier sections that the discrimination process is very robust in the face of such irrelevant features. By including additional features to the representation of balance scale problems, we would slow down the system's progress, but would not halt it, and this is precisely what we would desire in a more realistic model. In summary, while our model makes some important simplifications, it does provide an initial account of the transition process on the balance scale task, and it suggests that the discrimination learning theory may provide a useful framework for describing other aspects of cognitive development as well.

Evaluating the Theory

Now that we have examined the theory of discrimination learning, along with its applications to a number of domains, it is appropriate to attempt an initial evaluation. Scientific theories can be evaluated along a number of dimensions. One of these is *simplicity*, and the discrimination theory fares well on this criterion. Our implementation of the discrimination method required some 6 pages of LISP code, and the learning components of the systems discussed above ranged from 6 to 19 PRISM productions. Of course, these measures ignore the much more complex implementations of LISP and PRISM, since our models exist within these more basic frameworks. However, it seems reasonable to assume that the usefulness of list processing and production system languages has already been proven in other domains. We are discussing here the simplicity of a learning theory that accounts for phenomena beyond the normal range of these more basic approaches to intelligence.

Another important criterion for any scientific theory concerns its *generality*. We have applied our theory of discrimination learning to the domains of concept attainment, strategy learning, language acquisition, and cognitive development, and we feel that this provides strong evidence for the generality of the theory. However, it is better still if one can predict new areas to which a theory can be applied. In this case, the prediction process is straightforward: *the discrimination theory can be applied to any domain that involves the discovery*

of useful conditions on rules. Thus, one can imagine applications ranging from scientific discovery to motor learning. The current version of the theory is useful only in symbolic domains, but in principle it could be extended to numeric situations as well.

Another aspect of the generality dimension involves the types of rules that can be learned, and the conditions under which this is possible. We have seen that the discrimination theory can discover disjunctive rules, unlike most earlier condition-finding schemes. Moreover, it can learn in noisy situations, recover from changes in its environment, and learn partially useful rules based on inadequate representations. Of course, the method pays for this ability in searching a larger space than earlier approaches, and tends to learn more slowly as a result. However, this is the traditional price one must pay with weak, general methods that make few assumptions about the world in which they work. And from a psychological perspective, this slowness is quite plausible, since humans are generally slow learners as well.

In addition to employing the discrimination technique, the four learning systems described in previous sections share a number of other general characteristics. One such feature is that each of the models contained a critic that was responsible for noting errors of commission, and for invoking the discrimination mechanism. Second, all of the programs included a rule that noted correct predictions and stored the correct instantiation with the responsible production.* Finally, all of the systems represented information in very small chunks, such as *(move-1 before move-2)* and *(event-1 agent agent-1).* This was necessary to let the discrimination process discover and include relevant information to the exclusion of irrelevant knowledge.

Despite the similarities of SAGE, AMBER, and the other two learning systems, differences between the programs do exist. Given this situation, one may legitimately ask why a single, more general system could not be used for all of the domains. Now it is understandable that each domain would require its own performance rules, but why should there not be a common core of adaptive productions responsible for directing the learning process? In fact, this is precisely what we find with SAGE, a model of strategy acquisition that learns in three separate domains. One problem is that there are differences in the nature of the tasks that make some learning techniques inappropriate. For example, in concept attainment, it is reasonable to start with a default production that says "no" whenever no stronger rule exists for saying yes. This strategy leads to errors of omission, and a learning rule for dealing with such cases must be included. One could use an identical strategy for learning balance scale rules, except that a "no" response makes no sense in this context; the child either predicts that one side will go down or that the sides will balance. In this case it is more plausible to assume that down and balance rules already exist, and that errors of omission never occur. Second, in some cases a concern with modeling psychological data prevented us from using completely general approaches. For example, since children never say "ing" after an agent, we felt it appropriate to have the designating production include the relevant semantic role when constructing a morpheme rule. Even though such a condition could be learned through

* The concept learner and AMBER also incorporated a rule that detected errors of omission, and designated entirely new productions that would correct such errors in the future. This was not necessary in the strategy learning and balance scale tasks, though one can imagine variants in which it would be necessary.

discrimination, our concern with the data led us to introduce domain-specific knowledge into the learning process.

A final criterion for scientific theories is *fertility*, or the number of new and fruitful ideas that the theory generates. Our theory also fares well on this dimension, since it leads to two possible extensions that promise to deepen our understanding of learning and intelligence. The first of these concerns the formation of higher level concepts that help to describe information succintly. Recall that in some cases, the discrimination method is capable of discovering *sets* of conditions based on the comparison of a single good and bad instantiation. Suppose that in such cases, rather than including the set of conditions in a new variant, the routine instead created an entirely new production. This rule would contain the new set as its entire condition side, and would have a new predicate in its action side (with arguments taken from the predicates in the condition) which acts as shorthand notation for the set of relations. Once it was sufficiently strong, this rule would fire whenever the combination of conditions was present, *rewriting this set of relations in a more compact form.* This restatement would have two advantages. First, in the future the discrimination process need only look for a single difference rather than a conjunction of differences, so that its search process could be simplified considerably. Second, if the same set of conditions proved useful for other overly general rules, the speedup in discrimination would transfer to these cases as well. In fact, what we are proposing is a way of *changing representations* so that they more compactly describe the world.

A second extension concerns the generality, and thus weakness, of the discrimination process. Proponents of the knowledge engineering approach to Artificial Intelligence would no doubt suggest that we alter the discrimination technique to let it draw on domain-specific knowledge to direct its search for conditions. We have no objection to such an extension, provided it can be accomplished in a general manner. However, we would prefer to address the more challenging problem of devising a system that begins with a very weak but general discrimination learning mechanism, and *modifies* this method so that it is more efficient for the domain at hand. Such learning to learn has been proposed by a few researchers (Langley, Neches, Neves, and Anzai, 1980; Lenat, Sutherland, and Gibbons, 1982), and is clearly an order of magnitude more difficult than the work we have described to date. One clear prerequisite for such an approach, at least within the adaptive production systems framework, is the restatement of the learning theory as productions instead of LISP code. This will be necessary if the learning mechanisms are to change over time according to the same laws that the performance rules obey. This leads to the very tempting notion that the discrimination process can be applied *to* itself to generate more domain-specific and powerful versions *of* itself. Unfortunately, our ideas on this approach remain vague, and we cannot yet construct any examples of such a magical bootstrapping process.

In this chapter we have described a theory of discrimination learning that is capable of discovering appropriate conditions on production rules. We compared this theory to the more traditional generalization-based approaches, and found the discrimination method able to learn in situations where it was awkward or impossible for generalization to succeed. We examined implementations of the theory in four rather different domains, and found that it led to robust learning in each case. The theory appears to be simple, general, and amenable

to at least two promising extensions. Of course, many other processes have an equally important role to play in learning and development, and we have seen examples of some of them in other chapters of this book. However, we hope to have convinced the reader that the discrimination method is an important approach to learning that deserves increased attention in future research efforts.

References

Anderson, J. R., Kline, P. J., & Beasley, C. M. (1978). *A theory of the acquisition of cognitive skills* (Technical Report ONR 77-1). New Haven, CT: Yale University.

Anderson, J. R., & Kline, P. J. (1979). A learning system and its psychological implications. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp.16–21). Tokyo, Japan: Morgan-Kaufmann.

Anderson, J. R., Kline, P. J., & Beasley, C. M. (1980). Complex learning processes. In R. E. Snow, P. A. Federico, & W. E. Montague (Eds.), *Aptitude, learning, and instruction: Cognitive process analyses.* Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R. (1981). A theory of language acquisition based on general learning principles. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 165–170). Vancouver, B.C., Canada: Morgan-Kaufmann.

Anzai, Y. (1978). Learning strategies by computer. *Proceedings of the Canadian Society for Computational Studies of Intelligence* (pp. 181–190). Toronto, Ontario, Canada.

Anzai, Y. (1978). How one learns strategies: Processes and representation of strategy acquisition. *Proceedings of the Third AISB/GI Conference* (pp. 1–14). Hamburg, West Germany.

Baylor, G. W., Gascon, J., Lemoyne, G., & Pother, N. (1973). An information processing model of some seriation tasks. *Canadian Psychologist, 14,* 167–196.

Brazdil, P. (1978). Experimental learning model. *Proceedings of the Third AISB/GI Conference* (pp. 46–50). Hamburg, West Germany.

Brown, R. A. (1973). *A first language: The early stages.* Cambridge, MA: Harvard University Press.

Bruner, J. S., Goodnow, J. J., & Austin, G. A. (1956). *A study of thinking.* New York: Wiley.

Feigenbaum, E. A. (1963). The simulation of verbal learning behavior. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought.* New York: McGraw-Hill.

Feigenbaum, E. A., Buchanan, B. G., & Lederberg, J. (1971). On generality and problem solving: A case study using the DENDRAL program. In *Machine intelligence 6.* Edinburgh: Edinburgh University Press.

Forgy, C. L. (1979). *The OPS4 reference manual* (Technical Report). Pittsburgh, PA: Carnegie-Mellon University, Department of Computer Science.

Hayes-Roth, F., & McDermott, J. (1976). Learning structured patterns from examples. *Proceedings of the Third International Joint Conference on Pattern Recognition* (pp. 419–423).

Hedrick, C. (1976). Learning production systems from examples. *Artificial Intelligence, 7*, 21–49.

Hunt, E. B., Marin, J., & Stone, P. J. (1966). *Experiments in induction.* New York: Academic Press.

Iba, G. A. (1979). *Learning disjunctive concepts from examples.* Master's thesis, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA.

Kelley, K. L. (1967). *Early syntactic acquisition* (Technical Report P-3719.) Santa Monica, CA: The Rand Corporation.

Klahr, D., & Wallace, J. G. (1976). *Cognitive development: An information processing view.* Hillsdale, NJ: Lawrence Erlbaum.

Klahr, D., & Siegler, R. (1978). The representation of children's knowledge. In H. W. Reese & L. P. Lipsett (Eds.), *Advances in child development.* New York: Academic Press.

Langley, P. (1978). BACON.1: A general discovery system. *Proceedings of the Second National Conference of the Canadian Society for Computational Studies of Intelligence* (pp. 173–180). Toronto, Ontario, Canada.

Langley, P. (1979). *Descriptive discovery processes: Experiments in Baconian science.* PhD thesis, Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA.

Langley, P., Neches, R., Neves, D., & Anzai, Y. (1980). A domain-independent framework for learning procedures. *International Journal of Policy Analysis and Information Systems, 4*, 163–197.

Langley, P., & Simon, H. A. (1981). The central role of learning in cognition. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition.* Hillsdale, NJ: Lawrence Erlbaum.

Langley, P. (1982). Strategy acquisition governed by experimentation. *Proceedings of the European Conference on Artificial Intelligence* (pp. 171–176). Orsay, France.

Langley, P. (1983). Learning search strategies through discrimination. *International Journal of Man-Machine Studies, 18*, 513–541.

Lenat, D. B., Sutherland, W. R., & Gibbons, J. (1982). Heuristic search for new microcircuit structures: An application of artificial intelligence. *AI Magazine*, pp. 17–33.

Michalski, R. S. (1977). *Toward computer-aided induction* (Technical Report 874). Urbana, IL: University of Illinois, Department of Computer Science.

Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 305–310). Cambridge, MA: Morgan-Kaufmann.

Mitchell, T. M. (1978). *Version spaces: An approach to concept learning.* PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, CA.

Mitchell, T. M. (1979). An analysis of generalization as a search problem. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 577–582). Tokyo, Japan: Morgan-Kaufmann.

Mitchell, T. M., Utgoff, P., Nudel, B., & Banerji, R. B. (1981). Learning problem solving heuristics through practice. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 127–134). Vancouver, B.C., Canada: Morgan-Kaufmann.

Mitchell, T. M., Utgoff, P., & Banerji, R. B. (1982). Learning problem solving heuristics by experimentation. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto, CA: Tioga Press.

Newell, A., Shaw, J. C., & Simon, H. A. (1960). Report on a general problem-solving program for a computer. *Proceedings of the International Conference on Information Processing* (pp. 256–264).

Piaget, J., & Inhelder, B. (1969). *The psychology of the child.* New York: Basic Books.

Pople, H. E. (1977). The formation of composite hypotheses in diagnostic problem solving: An exercise in synthetic reasoning. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 1030–1037). Cambridge, MA: Morgan-Kaufmann.

Quinlan, J. R. (1982). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Palo Alto, CA: Tioga Press.

Reeker, L. H. (1976). The computational study of language acquisition. In M. Yovits & M. Rubinoff (Eds.), *Advances in Computers.* New York: Academic Press.

Selfridge, M. (1981). A computer model of child language acquisition. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 92–96). Vancouver, B.C., Canada: Morgan-Kaufmann.

Sleeman, D., Langley, P., & Mitchell, T. M. (1982). Learning from solution paths: An approach to the credit assignment problem. *AI Magazine*, pp. 48–52.

Vere, S. A. (1975). Induction of concepts in the predicate calculus. *Proceedings of the Fourth International Joint Conference on Artificial Intelligence* (pp. 281–287). Tbilisi, USSR: Morgan-Kaufmann.

Vere, S. A. (1977). Induction of relational productions in the presence of background information. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 349–355). Cambridge, MA: Morgan-Kaufmann.

Winston, P. H. (1970). *Learning structural descriptions from examples* (Technical Report AI-TR-231). Cambridge, MA: Massachusetts Institute of Technology.

Young, R. M. (1976). *Seriation by children: An artificial intelligence analysis of a Piagetian task.* Basel: Birkhauser.