A Knowledge Level Analysis of Learning Programs

Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, OR 97331

Abstract:

This chapter develops a taxonomy of learning methods using techniques based on Newell's *knowledge level*. Two properties of each system are defined: knowledge level predictability and knowledge level learning. A system is predictable at the knowledge level if the principle of rationality can be applied to predict its behavior. A system learns at the knowledge level if its knowledge level description changes over time. These two definitions can be used to generate the three-class taxonomy. The taxonomy formalizes the intuition that there are two kinds of learning systems: systems that simply improve their efficiency (symbol-level learning; SLL) and systems that acquire new knowledge (knowledge-level learning; KLL). The implications of the taxonomy for learning research are explored. Automatic programming research can provide ideas for SLL. Development of methods for KLL must rely either on the development of a principle of *plausible* rationality or on the construction of learning methods that work well only for certain kinds of environments. Explanation-based generalization and chunking methods address only SLL and do not provide solutions to the problems of KLL.

1 Introduction

In his AAAI President's Address, Allen Newell (1981) introduced an interesting method for describing the "knowledge" contained in a program. The method is based on viewing computer programs as rational agents at a "knowledge level." An intriguing application of Newell's method is to use it to define "learning" to be any process that increases the knowledge contained in a program. The purpose of this chapter is to investigate this definition and explore its consequences for machine learning research.

The chapter is organized as follows. In section 2, we review Newell's definition of the knowledge level and describe a modification of that definition appropriate for analyzing learning programs. Two important properties—knowledge level predictability and knowledge level learning—are defined. In section 3, we perform a knowledge level analysis of three learning systems and determine which systems possess these two properties. Based on this analysis, section 4 describes a three-class taxonomy of learning systems and attempts to characterize each of the classes precisely. In section 5, three general issues raised by the taxonomy are discussed. Finally, section 6 summarizes the main points of the chapter.

2 The Knowledge Level

2.1 Newell's description of the knowledge level

The purpose of the knowledge level is to provide a tool for describing systems and predicting their behavior. While many levels of description are adequate for predicting behavior—for example, the subatomic level, the register-transfer level, or the symbol level—the goal of the knowledge level is to predict behavior without examining the internal structure of the system. Our goal is to predict the future behavior of the system by considering its past (externally observable) behavior and any new information it has received from the environment.

Let us begin with some definitions. A *system* is defined to be everything contained within a given boundary. Everything outside the boundary is called the *environment*. The *behavior* of the system is the sequence of output actions (or information) that cross the boundary from the system to the environment.

At the knowledge level, a system is simply described by a body of knowledge. Included in this knowledge is a set of goals that the system is trying to achieve. Also included in this knowledge is knowledge of the possible actions that are available to the system.

To predict the behavior of the system, we treat the system as if it were a rational agent. A rational agent is any system that obeys the *principle of rationality*, which states that

> If the agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action as one of the possible actions to perform next.

Hence, given the knowledge, goals, and available actions of the agent, we can arrive at a prediction of its future behavior.

For example, if we know that an agent named Garfield is hungry (i.e., has the goal of eating food) and that Garfield knows there is a pan of lasagna in the refrigerator and that Garfield knows that he is capable of opening the refrigerator, obtaining the lasagna, and eating it, then we can predict that Garfield will do exactly that. Because we are assuming that Garfield is rational, we assume that he will realize the connection between his goals and his knowledge and act accordingly.

Unfortunately, the rationality principle is not always able to predict uniquely the future behavior of the system. Multiple goals, multiple action sequences, and inconsistent or incomplete knowledge can each prevent unique prediction. For example, suppose that Garfield, in addition to having the goal of eating food, also has the goal of getting as much sleep as possible. These two goals conflict, and the principle of rationality can not tell us which goal will take precedence at any given time.

Similarly, suppose that Garfield encounters several refrigerators rather than only one. In this situation, several alternative actions are available, each of which will achieve his goal of eating food. The principle of rationality cannot predict *which* refrigerator he will open first. All it can do is predict that *at least one* of the refrigerators will be opened and its contents consumed.

Finally, suppose Garfield's knowledge is contradictory or incomplete. As an example of contradictory knowledge, consider the following situation. Assume, as above, that Garfield knows all refrigerators contain food (unless he has eaten the food). But assume in this case that Garfield notices that the refrigerator in question is unplugged and he has the knowledge that unplugged refrigerators do not contain food. What will he do? Will he only consider the first rule (that all refrigerators contain food) and open the refrigerator? Will he only consider the second rule (that all unplugged refrigerators do not contain food) and ignore the refrigerator? Will he realize that his own knowledge is contradictory and find some way to resolve the contradiction? The principle of rationality cannot give us the answer. The rationality principle is also unable to provide predictions in the case of incomplete knowledge. Suppose Garfield knows nothing about ice chests and that he encounters one in his kitchen. How will he behave?

All of these three problems—multiple goals, multiple actions, and incomplete or contradictory knowledge—arise frequently in AI systems. Hence, it would appear that the rationality principle (and the knowledge level on which it is based) would not be very useful. As we shall see below, however, there are many systems for which the rationality principle can be applied to derive useful predictions. Let us therefore make the following definition.

Definition 1 *A system is said to be* predictable at the knowledge level *if its behavior can be predicted using the principle of rationality.*

Suppose now that we wish to predict the behavior of a particular system, S. There are three problems that we must solve before we can apply the principle of rationality.

1. We must determine what knowledge S contains. This is difficult because knowledge is not a concrete substance, nor is it always localized in discrete data structures within the system. Hence, we cannot determine the knowledge contained in S by simply inspecting its internal structure.

2. We must have some way of making this knowledge concrete so that we can manipulate it using the rationality principle. Otherwise we have no way of deciding ourselves which actions of S will, according to the knowledge in S, achieve S's goals.

3. We must decide what knowledge is acquired by S when new inputs are received from the environment. This is difficult, because all perception (of environmental inputs) involves interpretation; the problem of perception has resisted philosophical analysis for centuries.

Newell proposes solutions to each of these problems, but as we shall see, the solutions are only approximate. First, to determine the knowledge contained in S, Newell applies the principle of rationality in reverse. We begin by observing the behavior of S. Then we attribute to S a body of knowledge (including goals) that suffices to make S's behavior rational. We then use this attributed knowledge to predict future behavior.

To clarify this solution, let us again consider the Garfield agent. Assume that we do not know what knowledge is contained in Garfield, but we observe Garfield walk to the refrigerator, open it, remove all of the contents (a plate of lasagna, a meat loaf, and a carton of milk), and eat each item. This behavior would be rational if we attribute to Garfield the knowledge that everything in the refrigerator is food, that Garfield has the goal of eating (large quantities of) food, and that Garfield knows how to walk to the refrigerator, open it, and eat the contents. In the future, we can predict that Garfield will repeat this behavior when he is hungry. We can also make other predictions. For example, if Garfield sees a plate of lasagna across the room, we can predict that he will walk over to the plate and eat the lasagna because we know (a) that he has the goal of eating food, (b) that he knows how to walk, (c) that he knows lasagna is food, and (d) that he knows the lasagna is there because he saw it.

This approach to using the principle of rationality in reverse has three interesting consequences. First, the knowledge ascribed to S will always be consistent.[1] This is because if S internally has two conflicting items of knowledge, only one of these items will contribute directly to the selected action. The actions of S are always unambiguous even if the underlying knowledge is not. Second, for similar reasons, the goals ascribed to S will not conflict. Any internal goal conflict will have been effectively resolved when S finally acts. Third, if alternative actions were available, there is no way that we, as observers, can know whether S knew about them. Hence, there is no need to ascribe knowledge of alternative actions in order to view S's behavior as rational. In short, because we are using the principle of rationality to ascribe knowledge based on behavior, it is necessarily the case that that behavior can be predicted (ex post facto) at the knowledge level using the principle of rationality.

Newell solves the second problem—of making knowledge concrete—by using a second system, S_2 as a model of S. To construct a knowledge-level model of S, we attribute to S_2 a set of symbol structures (Newell & Simon, 1976) such that S_2 would behave the same way that S was observed to behave. We then use S_2 as a model of the future performance of S. We will call S_2 the *modelling system.*

Consider Garfield once again. To predict Garfield's future behavior, we could use the STRIPS (Fikes, Hart & Nilsson, 1972) planning system. We could give STRIPS operators for walking, opening the refrigerator, and eating, as well as the ability to visually recognize food containers. We could then employ STRIPS to generate our predictions. STRIPS is able, at least in situations of this kind, to determine what action sequence will achieve its goals. Hence, it provides a sufficiently rational model of Garfield.

Newell solves the third problem—of assigning knowledge to environmental inputs—by using the modelling system S_2 again. The knowledge contained in an input from the environment is whatever knowledge S_2 can know by observing that same input (using the knowledge already ascribed to S to aid in interpreting the input).

Consider the Garfield example one last time. When Garfield sees the plate of lasagna sitting across the room, we ascribe to him the knowledge that there is a plate of lasagna sitting across the room (or at the very least, that there is a quantity of food sitting across the room). The basis for this ascription is twofold. First, we have seen previously that Garfield was able to recognize lasagna when he opened the refrigerator door. Second, when we look across the room (using ourselves as S_2), it sure looks like a plate of lasagna to us!

Each of Newell's solutions to these three problems introduces uncertainty into the knowledge level description of the system S. Let us first consider the uncertainties arising from the use of the principle of rationality (in reverse) to ascribe knowledge. The rationality principle does not uniquely determine the knowledge contained in a system. Given a particular sequence of behavior by

[1]It is certainly possible to ascribe inconsistent knowledge to an agent, but such an ascription would then prevent the rationality principle from predicting the observed behavior.

S, there are many alternative bodies of knowledge that can be ascribed to S such that its behavior is rational. In Garfield's case, he might have the goal of maximizing his weight and therefore consume large quantitites of food even when he is not hungry. Or perhaps even more fancifully, he might believe that the refrigerator is going to explode and that the only way to prevent this is to open it and consume the contents. Neither of these knowledge level descriptions is particularly plausible however, because they conflict with our own knowledge (as observers playing the role of the modelling system S_2). In general, an observer who is performing a knowledge level analysis must apply his or her own knowledge to construct the most plausible knowledge level description of the system.[2]

In addition to the uncertainties introduced when we ascribe knowledge to S, further approximations arise when we employ a modelling system, S_2 as a concrete model of S. There is no physical system that behaves as a fully rational agent. This is because no (finite) physical system can explore all of the possible connections between its knowledge and its goals. As a result, knowledge level predictions (based on the symbol level calculations of S_2) must always be approximate, and not exact. STRIPS, for example, is not entirely appropriate as a model of Garfield, because it has many shortcomings: there are plans it cannot construct, it can get caught in infinite loops, and so on. Garfield has shortcomings himself, of course, but these are unlikely to be the same as the shortcomings of STRIPS.

Finally, Newell's solution to the problem of environmental input is the third point at which our knowledge-level description of S may only be approximate. Every act of perception involves interpretation, and consequently, different modelling systems (different S_2's) receiving identical inputs may interpret them in wildly different ways.

When we combine the uncertainties introduced by our choice of S_2 for interpreting inputs and computing predictions with the uncertainties introduced by the use of the principle of rationality to ascribe knowledge, we see that the knowledge level is inherently approximate—indeed, Newell calls it "radically approximate." Nevertheless, it can provide a very useful level of description of complex systems.

2.2 The computational-closure knowledge level

In this chapter, we are interested in determining the *limits* of what a given learning system knows. Consequently, it is useful for us to adopt a slightly modified definition of the knowledge level.

Normally, we attribute knowledge to a system S based on S's behavior over some rather short period of time. In such cases, there is a tendency to underestimate the knowledge contained in S for two reasons. First, we have observed only a small sample of the total behavior of which S is capable. Second, the behavior that we have observed was probably selected based on an incomplete consideration by S of its knowledge. Any finite agent must make decisions without considering all of the consequences and ramifications of those decisions. Hence, S may have missed an opportunity that, given an indefinite amount of computational resources, it would have discovered.

To circumvent this problem, let us define the computational-closure knowledge level (KL_{cc}) of a system as follows. To determine the knowledge contained in a system S, we will allow S infinite computational resources before it selects the actions to perform. We will then construct a knowledge level description of S following the approach described above.

[2]A programmer analyzing (or designing) a program can learn something about the knowledge it contains by inspecting the program's data structures. If the program is well-designed, these data structures will have a declarative reading that aligns well with a knowledge level description. However, the programmer must always be wary of the "pretend it's English" problem (McDermott, 1976).

In practice, of course, we cannot permit any program to consume infinite time and space resources. However, by inspecting the implementation of the program, we can often determine whether additional computational resources would change the actions selected by the system.

There are two important points to note about this definition of KL_{cc}. First, because the knowledge level (and particularly KL_{cc}) is based only on the *behavior* of a system, it ignores all issues concerning how the system is implemented. This is particularly true for KL_{cc}, where even the computational limitations of the implementation are ignored. Hence, if two programs have the same behavior but one program is much less efficient than the other, then the two systems contain the same knowledge (from the point of view of KL_{cc}).

Second, because KL_{cc} ignores questions of implementation, it provides a very nice specification level for describing the desired properties of systems that are not yet implemented. This specification defines the meaning of correct and incorrect behavior for the system.

The idea of using the computational closure can be traced to a paragraph in Newell's paper in which he discusses one of the many informal uses of the term "knowledge" in describing AI systems:

> [W]hen we say, as we often do, that a program "can't do action A, because it doesn't have knowledge K," we mean that no amount of processing by the processes now in the program on the data structures now in the program can yield the selection of A. (E.g., "This chess program can't avoid the tie, because it doesn't know about repetition of positions.") Such a statement presupposes that the principle of rationality would lead to A given K, and no way to get A selected other than having K satisfies the principle of rationality. If in fact some rearrangment of the processing did lead to selecting A, then additional knowledge can be expected to have been imported, e.g., from the mind of the programmer who did the rearranging (though accident can confound expectation on occasion, of course).

In other words, when we want to claim that a program does not know some knowledge K, we must show that no matter what computational resources are given to the program, it will never act as if it knows K.

2.3 Knowledge Level Learning

Now that we have reviewed the definition of the knowledge level and the techniques for applying that definition, we can turn to the main concern of this chapter—the definition of "learning." Let us define *knowledge level learning* (KLL) as follows:

Definition 2 *A program is said to exhibit* knowledge level learning *if its computational-closure knowledge level description changes over time.*

In other words, suppose that at time t_1 we analyze system S using KL_{cc} and determine that it contains knowledge K_1. At some later time t_2, after S has changed in some way, we repeat the analysis and conclude that S now contains knowledge K_2. If K_2 is different from (and preferably larger than) K_1, then we say that S has learned at the knowledge level.

3 Knowledge Level Analysis of Three Learning Systems

In this section, we will apply the tools described above to analyze three learning systems. For each system, we will be interested primarily in two questions: (a) can the system's behavior be predicted at the knowledge level and (b) does the system exhibit knowledge level learning?

3.1 Explanation-based Generalization

First, let us consider explanation-based generalization (EBG) systems such as the `safe-to-stack` system described in Mitchell, Keller, and Kedar-Cabelli (1986). The task of this system is to determine, by examining descriptions of two objects, whether it is safe to stack one of the objects on top of the other. The system starts with a "domain theory" that provides an inefficient definition of `safe-to-stack`. As it learns, it develops a much more efficient definition using only the "operational" features of the theory.

If, prior to any learning, we analyze the knowledge in the system using KL_{cc}, we find that it already knows how to perform its task for any two objects.[3] Internally, it determines whether the objects are `safe-to-stack` by constructing a proof in its domain theory.

During the learning process, training examples of `safe-to-stack` objects are presented to the system. Interestingly, from the KL_{cc} point of view, these training examples do not provide any new knowledge to the system—it already knows that they are `safe-to-stack`. However, as Mitchell et al. point out, the training examples provide guidance to the "concept operationalization" process. The EBG program constructs a more operational definition of `safe-to-stack` by constructing an explanation of *why* the example objects are `safe-to-stack` and then using the proof to construct a generalized description of all pairs of objects to which the *same proof* could apply. Hence, from a training example in which object a has a volume of 3 and a density of 2 and object b has a volume of 5 and a density of 4, the `safe-to-stack` program might derive the general statement that object x can be safely stacked on object y if $volume(x) \times density(x) < volume(y) \times density(y)$.

After learning, a KL_{cc} analysis of the knowledge in the system shows that the system is still capable of performing its task for any two objects. Of course now the task is performed by simply applying the operational definition of `safe-to-stack` rather than by applying the domain theory. Hence, the speed at which the system performs is much faster. However, at the knowledge level, this change in implementations is invisible, because given infinite computational resources, the system produces the same behavior.

Let us turn to our two main questions: knowledge level predictability and knowledge level learning. Our analysis shows that the `safe-to-stack` program is predictable at the knowledge level—even while the program is "learning." Furthermore, according to the KL_{cc} analysis, explanation-based generalization programs such as the `safe-to-stack` program do not exhibit knowledge level learning. Hence, our proposed definition of learning does not capture the kind of learning performed by these systems.

3.2 Database Systems

The second kind of learning system we wish to consider is a simple deductive database system, such as Prolog (Clocksin & Mellish, 1984), in which the task is to accept facts and rules and to provide answers to queries. While such systems are not usually considered learning systems, we will see that our definition of knowledge level learning labels them as such.

To make things concrete, consider an ordinary Prolog interactive session. Suppose we type

```
asserta((mortal(X) :- man(X))).
asserta(man(socrates)).
```

This can be interpreted as asserting that all men are mortal and that Socrates is a man. We now ask Prolog whether Socrates is mortal:

```
?- mortal(socrates).
```

[3]The objects are assumed to be represented using only operational features.

```
yes
```

Prolog returns yes. Now suppose we ask it whether Homer is mortal:

```
?- mortal(homer).
no
```

It returns no, which in this case, we will interpret as meaning that it doesn't know.[4] But suppose that we tell it that Homer is a man:

```
asserta(man(homer)).
```

Now, Prolog will answer that Homer is mortal as well:

```
?- mortal(homer).
yes
```

It has learned something new.

A knowledge level analysis of programs like Prolog is very straightforward, because the internal structures of these programs can be viewed as logical sentences. We begin by attributing to Prolog the goal of printing correct answers. After the first two assertions have been entered, we also ascribe to Prolog the knowledge that all men are mortal and that Socrates is a man. Given even modest computational resources, we see from the first query that Prolog also knows that Socrates is mortal. However, Prolog does not know whether Homer is mortal (or whether he is a man), and no amount of computation will allow it to derive that knowledge.

When we tell Prolog that Homer is a man, we are giving Prolog some new knowledge. It combines this new knowledge with its previous knowledge and therefore knows that Homer is mortal as well.

In summary, let us consider the answers to our two questions of knowledge level prediction and knowledge level learning. First, the full behavior of Prolog, as it processes assertions and queries, is predictable at the knowledge level. Second, the KL_{cc} analysis of Prolog shows that it does perform knowledge level learning because its knowledge level description changes over time. At one time (after the first two assertions), it did not know that Homer was mortal and at a later time (after the third assertion), it did.

3.3 Inductive learning programs

The third class of learning programs that we wish to consider is the class of programs that acquire general rules by inductive inference from training examples. For concreteness, we will consider the ID3 program developed by Ross Quinlan (1986).[5]

The performance task of ID3 is to classify objects (or events) into one of a finite number of classes. Each object is described to ID3 as a simple vector of features, and ID3 produces as output the name of the class to which that object belongs. The possible features, feature values, and class names are defined to ID3 during program initialization.

ID3 also accepts input in the form of training examples—examples of objects with the correct classifications given as well. By analyzing these examples, ID3 constructs and maintains definitions for each of the classes in terms of the object features.

[4]We are ignoring the negation-as-failure feature of Prolog. Because SLD resolution is complete, failure to find a proof (which is what no means), indicates that no proof exists. Consequently, Prolog does not know whether Homer is mortal.

[5]Our example treats ID3 as if it were an incremental learning program. This could be accomplished by re-running ID3 after each new example is presented. Alternatively, the incremental version of the algorithm, ID4 (Schlimmer & Fisher, 1986), could be used instead.

Let us consider a particular session with ID3. Suppose there are two (mutually exclusive) classes of objects, `Pretty` and `Ugly`. Each object is described by three features: `Color` (`red` or `black`), `Size` (`small` or `large`), and `Shape` (`circle` or `square`). Suppose we begin by presenting ID3 with a single training example: a small, red, square:

```
[red, small, square] --> Pretty
```

Next, suppose that we ask ID3 to classify a new object, a large, black, circle. In response, ID3 will answer "Pretty." In fact, ID3 will classify any object as pretty after seeing only one example. The definition it has constructed for `Pretty` states that all objects are pretty.

Now suppose we present a second training example, a large, black square that is ugly.

```
[black, large, square] --> Ugly
```

Now if we ask ID3 to classify a large, black, circle, it will answer "Ugly." The definition that it has learned is that all red things are pretty and all black things are ugly. It selected this definition because the color is sufficient to distinguish between the two training examples. It could have just as well selected size, because, based on the two examples, size is also a discriminating feature. The choice of color rather than size was arbitrary.

A knowledge level analysis of ID3 must be done carefully. Therefore, let us employ first-order predicate logic as our second system, S_2, for performing the analysis. Let us also postpone for a moment the question of what knowledge ID3 contains initially and begin by considering the knowledge in ID3 after the first training example is presented. Clearly, based on the performance of ID3, the system contains the knowledge that all objects are pretty:

$$\forall x \; \mathit{Pretty}(x)$$

However, after the second training example, ID3 now contains the knowledge that all red objects are pretty and all black objects are ugly:

$$\forall x \; \mathit{Color}(x) = \mathit{red} \;\supset\; \mathit{Pretty}(x)$$

$$\forall x \; \mathit{Color}(x) = \mathit{black} \;\supset\; \mathit{Ugly}(x)$$

Therefore, the knowledge level description of ID3 is changing over time—it is exhibiting knowledge level learning.

In the two previous kinds of learning systems, we were able to predict the behavior of the system from the knowledge level. Can this be done for ID3? Let us focus on the change in knowledge that occurs as a result of the second training example. What knowledge is provided to ID3 by the second example itself? According to S_2, it simply states that a large, black square is ugly:

$$\mathit{Size}(e2) = \mathit{large} \,\wedge\, \mathit{Color}(e2) = \mathit{black} \,\wedge\, \mathit{Shape}(e2) = \mathit{square} \,\wedge\, \mathit{Ugly}(e2)$$

This contradicts the knowledge contained in ID3 at this point (namely, $\forall x \; \mathit{Pretty}(x)$). Because of this contradiction, the principle of rationality cannot provide us with a unique prediction for the future behavior of ID3. Therefore, ID3 is *not* predictable at the knowledge level.

This contradiction between the knowledge in the system and the knowledge provided by inputs from the environment did not arise in either of the two previous learning systems. The reason for this is that neither of those systems "went beyond" the knowledge provided by the environment to make "inductive leaps." Contradictions cannot arise in EBG systems, because the training instances are already derivable from the domain theory and hence do not tell the learning system anything new. Contradictions cannot arise in the Prolog examples because negated assertions cannot be entered.[6]

To summarize, let us consider our two questions of knowledge level prediction and learning once again. The analysis shows that ID3 is not predictable at the knowledge level. The analysis also shows that ID3 exhibits knowledge level learning.

Before leaving ID3 altogether, let us consider what happens when the first training example is processed. Before the example is presented, ID3 will not classify any objects as being either pretty or ugly. Therefore it is difficult to characterize its starting knowledge. Russell (1986) has proposed that the starting knowledge be described by logical sentences called *determinations*. In this case, the determination would state that some combination of color, size, and shape determines the classification:

$$\begin{aligned} \forall\, x, y \qquad & Color(x) = Color(y) \wedge \\ & Size(x) = Size(y) \wedge \\ & Shape(x) = Shape(y) \;\supset\; [Pretty(x) \;\equiv\; Pretty(y)]. \end{aligned}$$

This seems to capture fairly well the intent of the user in specifying these features to ID3.

However, from this weak determination and the knowledge provided by the first example (that a small, red, square is pretty), the principle of rationality cannot predict how ID3 will classify other objects, such as large, black circles. The problem is that our knowledge level description of ID3 is incomplete. All it could predict is that ID3 will classify all other small, red, squares as pretty also. Hence, the same problems that arose with the second training example—the failure of the knowledge level to predict future performance—arise with the first example as well.

If we are willing to attribute more starting knowledge to ID3, we can construct a knowledge level description that predicts ID3's behavior after the first training instance only. For example, suppose we ascribe to ID3 the starting knowledge that either "all objects are pretty" or else "all objects are ugly,"

$$[\forall\, x\; Pretty(x)] \;\vee\; [\forall\, x\; Ugly(x)].$$

Given the first training example (a small, red square that is pretty), ID3 could logically conclude that all objects must be pretty, because the existence of a single pretty object rules out the alternative hypothesis that "All objects are ugly."

This "trick" resolves the problem of the first training example, but no similar trick can be used to predict ID3's behavior after the second training example. The fundamental problem is that inductive learning programs go beyond their starting knowledge (and the knowledge received from the environment) to make "inductive leaps". This shows up as nonmonotonic behavior in this example: ID3 classifies the large, black, circle as pretty after the first training example, but after the second training example, it "changes its mind" and classifies it as ugly.

In summary, ID3 exhibits knowledge level learning, but more significantly, its behavior not predictable at the knowledge level.

4 A Taxonomy of Learning Systems

The three learning systems discussed in the last section were chosen to illustrate the relationship between knowledge level learning and knowledge level prediction. The explanation-based generalization programs do not perform knowledge level learning but they are predictable at the knowledge

[6]Again, we are ignoring the negation-as-failure feature of Prolog. It can easily lead to contradictions and nonmonotonic behavior. Similar problems arise in deductive database systems when they need to perform belief revision (Winslett, 1986).

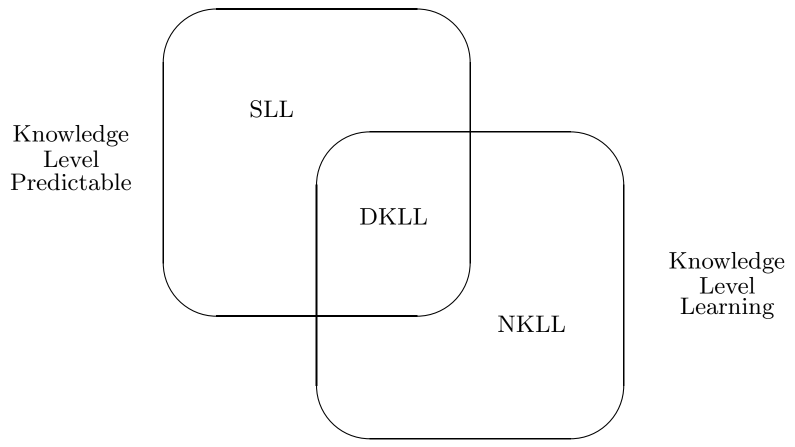

Figure 1: A taxonomy of learning systems

level. The deductive database systems exhibit both knowledge level learning and predictability at the knowledge level. Finally, the inductive learning programs exhibit knowledge level learning but are not predictable at the knowledge level.

Figure 1 summarizes this relationship. The set of systems that perform knowledge level learning partially intersects the set of systems that are predictable at the knowledge level. Based on the definitions of knowledge level learning and knowledge level predictability, we can give reasonable names to these three classes of learning systems. We call the first class (EBG systems) symbol level learning (SLL) systems, because their learning behavior is apparent only at the symbol level.

Definition 3 *Symbol level learning is improvement in computational performance that yields no change in the computational-closure knowledge level description of the system.*

The second class (Prolog-like systems), we call determined knowledge level learning (DKLL) systems, because their behavior can be determined (predicted) at the knowledge level using the principle of rationality. These systems can also be viewed as symbol level learning systems that also receive inputs from some external source of knowledge.

Definition 4 *Determined knowledge level learning is knowledge level learning that can be described (and predicted) at the knowledge level.*

Finally, we call the third class (which includes AQ11, ID3 and other inductive learning programs) non-determined knowledge level learning (NKLL) systems.

Definition 5 *Non-determined knowledge level learning is knowledge level learning that cannot be described (or predicted) at the knowledge level.*

Now that we have developed our taxonomy of learning systems, let us attempt to characterize each of the categories in more detail.

4.1 Symbol level learning

As we have seen above, the goal of a symbol level learning system is to improve its performance as much as possible without changing the knowledge level view of the system. Another way of phrasing this is that SLL is *correctness-preserving program improvement.*

This observation raises two important points. First, the knowledge level description of the system serves as a specification for "correct" behavior. This is particularly useful, because it provides a way of evaluating and comparing different SLL techniques. As we shall see below, there is no similar definition of "correctness" for KLL systems, and this has made progress in this area more difficult.

Second, an important source of techniques for SLL is the work that has been accomplished in the areas of automatic programming and algorithm design. Research such as Mostow's (1983) FOO program and Keller's (1983) analysis of LEX2 and Meta-LEX show how automatic programming methods can be adapted and applied to machine learning problems.

4.2 Knowledge level learning

Knowledge level learning (both DKLL and NKLL) can also be characterized precisely. A system performs knowledge level learning if and only if (a) it receives input from the environment and (b) that input causes a change in the behavior of the system (as compared to the behavior without the input or with some other input).

To see that these two conditions are necessary, consider the KL_{cc} analysis of a system that receives no inputs from the environment. Such a system, at the KL_{cc} level, is simply permitted to reach computational closure, at which point, we ascribe knowledge by observing its behavior. Suppose at some future time we again wish to determine the knowledge contained in the system. Again we permit the system to reach computational closure, and the same resulting knowledge will be obtained (because the computation is deterministic[7]). The only difference in the system at the later time may be that it was already at or near computational closure, because it remembered results from the previous computation. Hence, condition (a) is required for knowledge level learning.

Suppose now that we have a system that accepts input from the environment, but this input does not contribute to any change in behavior. Because the knowledge level description of the system is derived from its behavior, this means that there will be no change in the knowledge level. Therefore the system will not exhibit knowledge level learning. Explanation-based generalization programs have precisely this property. The training examples do not cause any behavioral changes.

Given that both conditions are necessary, we must now show that they are sufficient. By condition (b), the behavior of the system changes over time. By condition (a), the system "pauses," and therefore, these changes in behavior will be observable. Hence, the system will exhibit knowledge level learning.

These two conditions therefore provide a complete characterization of knowledge level learning.

4.3 Non-determined knowledge level learning

NKLL is much more difficult to characterize. Although we normally associate it with any kind of inductive or non-monotonic reasoning technique, the use of these techniques does not necessarily produce NKLL.

[7]If the system contains a genuine source of nondeterminism—e.g., the system clock time—then that is considered to be an input to the system.

The LEX system, for example, employs inductive techniques to learn search control heuristics. However, these heuristics merely improve the speed with which the system performs, and hence, the overall system exhibits only SLL (more on this below).

It is also possible for a system to exhibit NKLL when it is applying primarily explanation-based learning techniques. The LEAP system (Mahadevan, 1985), for instance, accepts training examples of VLSI design steps and learns general rules for VLSI design. This process in itself is simple EBG in which the goal concept is "legal VLSI design step" and the learned rules are particular specializations of this concept. However, when LEAP *applies* these rules, it gives them *preference* over other design steps that, while provably correct according to the goal concept, have not been observed by LEAP. In other words, LEAP *assumes* that the training examples it receives are also examples of "desirable VLSI design steps." It expects that applying these rules will lead to *good* designs, not merely *legal* designs. A knowledge level analysis of LEAP shows that prior to learning, it is capable of producing all legal VLSI designs, whereas after learning, it has learned to produce (and recognize) only good designs. It has performed NKLL. Notice however that the use of EBG techniques has nothing to do with the NKLL. The "inductive leap" occurs when LEAP *prefers* the observed design steps to other, legal steps.

Fundamentally, the only characterization of NKLL systems is that they cannot be predicted at the knowledge level. To predict the behavior of the ID3 system, for example, we must leave the knowledge level and descend to the symbol level. Here, we see that ID3 represents its knowledge (of, e.g., pretty and ugly objects) as a decision tree. During learning, it prefers the smallest decision tree consistent with the data. Hence, after seeing one example of a pretty, small, red square, ID3 constructs the degenerate tree (a single leaf node) that simply labels all objects as pretty. After seeing the second example of an ugly, large, black square, this tree is inconsistent, so ID3 constructs a tree in which the root node tests the color of the object. Red objects are labelled pretty; black objects are labelled ugly. This tree is (one of) the smallest tree(s) consistent with the data. It is this preference for syntactic simplicity (smallest trees) that provides a way of predicting ID3's behavior. This preference is usually called a "bias" (Utgoff & Mitchell, 1982).

This lack of predictability at the knowledge level means that the knowledge level cannot provide a specification of correct behavior for NKLL systems. This has important implications for machine learning research, because, without a method for specifying the correct behavior of a system, it is difficult to evaluate alternative implementations of that system.

There are two avenues to explore in response to this problem. One approach is to develop an extension to the knowledge level based on the notion of *rational plausible reasoning.* The other approach is to shift to a "situated action" definition of rational behavior, in which we consider methods that are rational and plausible only in certain environments (Barwise & Perry, 1983). Let us consider each of these in turn.

A theory of rational plausible reasoning would dictate what new beliefs should be adopted when new information is received from the environment. It would provide a general method by which prior beliefs could be revised in the light of new evidence. The best tool we have for developing such a theory is probability theory. Unfortunately, like logics, probability theory is a symbol level mechanism, and it requires something analogous to model theory in order to be applied as a knowledge level analysis tool. This is an important area for future research, and significant progress is being made (Nilsson, 1986).

The alternative approach—the situated action approach—has been explored in much greater depth. The basic theme of this approach is to consider the relationship between the system and its environment rather than focusing on the system alone. Notice, in particular, that the knowledge level defines the knowledge contained in a system without reference to the environment. Consequently, the knowledge can be completely inaccurate. Similarly, efforts to construct theories of plausible rationality also focus on the beliefs of the system. The advantage of such "environment-independent" methods is that they can be applied in *any* environment. Hence, the principle of rationality can be applied to predict the behavior of *any* rational system in *any* environment, no matter how hostile or foreign it may be.

When we shift to the situated-action viewpoint, we give up this environment-independence. Instead of searching for learning methods that work in all possible worlds, we focus instead on learning methods that work in *our* world. Inverting this argument, we stop asking "what methods would be rational in all environments?" and instead ask "in what environments would this method appear to be rational?"

The simplest instance of this approach is the line of research that is exploring the space of learning methods. In this research, new methods are invented and then tested to see how well they work on a variety of test domains. Different groups of researchers are working on different aspects of intelligent systems.

One group of researchers is exploring the computational properties of various architectures and exploiting their constraints to provide preferences (biases) for inductive learning (e.g., Genesereth, 1980). This is the approach of the connectionists (e.g., Hinton, Sejnowski & Ackley, 1984) and also of the SOAR group (Steier, et al., 1987).

Another group of researchers—historically, the main stream of AI learning research—is exploring various inductive learning algorithms, all within a conventional symbolic architecture. A recent, healthy development is the use of common data sets to compare the performance of these learning methods. Fisher (in press) and Kibler and Aha (1987) have used a data set originally developed by Quinlan, Compton, Horn, and Lazarus (1986). A data set developed by Michalski and Chilausky (1980) has been employed by Fisher (1987).

A third group is studying the role of vocabulary choice in learning systems, while holding the learning method fixed (see, e.g., Lenat & Brown, 1984; Flann & Dietterich, 1985, 1986). Given a specific learning problem, how can we choose the vocabulary so that the learning is made easy? In current induction systems, a great deal of vocabulary engineering takes place behind the scenes. We need to study this phenomenon and try to extract some principles. Are our vocabulary engineers relying on their own knowledge of the "right answer" to guide their choices?

In all of these groups, the proof that a learning method is valuable is based on empirical testing. Furthermore, the very nature of these methods appears to dictate that methods that perform well in one domain or environment will perform poorly in others. This is the price of the situated-action approach.

The situated-action approach is also being explored via theoretical methods. One of the earliest attempts in this direction was Gold's (1967) definition of identification in the limit. This criterion states that, given enough training instances, the learning system should eventually converge on the correct theory (i.e., concept, language, etc.). To demonstrate that a given learning method converges, Gold (and his successors; see Angluin & Smith, 1983; Oshersson, Stob, & Weinstein, 1986) place certain restrictions on the environments in which the method is to operate. They then prove that for any concept satisfying the given restrictions, the learning method will converge.

Identification in the limit is a very weak criterion, and it certainly does not completely characterize the ideal learning system. Recent work by Valiant (1984) has produced a much more interesting criterion: PAC learning ("probably approximately correct"). The idea is that an NKLL program should, with high probability, adopt knowledge (or beliefs) that are approximately correct (with respect to its environment). There are two opportunities for error under the PAC definition. First, the learning system is occasionally permitted to adopt knowledge that is wildly incorrect. Second, most of the time the knowledge adopted by the learning system will not be exactly correct, but it will be close. Kearns, Li, Pitt & Valiant (1987) give a good review of recent progress in this area.

As with the proofs of identification in the limit, proofs of PAC learning require that we make some assumptions about the nature of the environment (e.g., that the correct theory can be expressed as a formula in disjunctive normal form where each disjunct contains at most k terms). Once we have some limits on the complexity of the environment, we can then show that, with high probability, a given learning algorithm will produce knowledge that has a guaranteed error level. (With low probability, the knowledge will exceed the error level.) Both the error level and the bounded number of exceptions are given as functions of the number of examples observed.

In summary, the use or non-use of "inductive" methods is not sufficient to characterize NKLL. Fundamentally, NKLL signals a failure of the knowledge level to describe and predict an important kind of intelligent behavior. Hence, we must seek alternatives to the knowledge level. One alternative is to develop a principle of rational plausible reasoning and use it to extend the knowledge level notion. The other alternative is to take a situated-action approach in which we consider methods that are rational and plausible only in certain kinds of environments (Barwise & Perry, 1983). The hope is that we can find methods that work over a wide range of environments (but of course, not all).

5 Discussion

Now that we have developed our taxonomy of learning systems and analyzed in detail each kind, we now turn our attention to some general issues raised by the taxonomy.

5.1 The definition of learning

One pleasing aspect of our taxonomy of learning systems is that it formalizes the intuition that there are two very different kinds of learning: learning that improves performance and learning that acquires new knowledge. For many years there has been some controversy about how 'learning' should best be defined. The majority of workers in machine learning subscribed to the following "improved performance" definition (Simon, 1983):

> Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.

This definition was always intended to include both SLL and KLL. The feeling was that performance could be "improved" either by improving the efficiency with which existing knowledge was used or by acquiring new knowledge (see, for example, Dietterich, et al., 1982). However, Simon termed his definition "only partially satisfactory," and other researchers (e.g., Scott, 1983) have criticized it for excluding important kinds of learning. In particular, the improved performance definition requires that there exist some performance task by which the improvement can be measured. Learning in the absence of a specific performance task is not true learning according to this definition. In retrospect, we can see that this was all a maneuver to avoid talking about *knowledge*. By defining knowledge in terms of problem-solving performance, it was possible to convert "acquisition of knowledge" into "improvement of performance." Newell's clarification of the knowledge level eliminates the need for this dodge. We can now come right out and say it: one kind of learning is the acquisition of knowledge.

5.2 Moving the system boundary

A second issue raised by the taxonomy, and particularly by the discussion of NKLL, is how the knowledge level description of a system changes as we move the system boundary. The issue arises most clearly in the LEX system (Mitchell, Utgoff & Banerji, 1983). LEX solves symbolic integration problems in calculus. It contains a simple heuristically-guided uniform-cost problem solver and a body of symbolic integration operators. When it begins, it has no search control heuristics, so it simply conducts a breadth-first search. During the learning process, problems are solved and then analyzed to extract search heuristics so that future problem solving is more efficient. After each integration problem is solved, the sequence of operators on the successful path is analyzed to extract positive and negative examples of cases where a given operator (e.g., integration by parts) should have been applied. These are then generalized inductively to produce a general search heuristic for that operator.

At the KL_{cc} knowledge level, LEX is classified as a symbol level learning system, because before learning begins, it is already capable of solving all (solvable) integration problems (given infinite resources). The learning process simply speeds up this problem solving.

However, if we move the system boundary so that we are only considering the body of search heuristics, then the inputs to this subsystem are the training examples and the outputs are judgments concerning the advisability of applying a given operator in a given state. This subsystem (call it H) is clearly learning at the knowledge level, because, prior to learning, it is incapable of making any operator-selection judgments. After learning, it is capable of making such judgments.

Why could we not detect this KLL subsystem H when we considered the entire LEX system? The reason is that the actions of H are not observable at the boundary of the entire LEX system. The only thing that crosses the LEX boundary is the final solution to the integration problem. The problem solver inside LEX is already capable of producing these outputs correctly before learning even begins. In other words, by moving the system boundary, we change the behavior that is externally observable and we change the performance task of the system.

This example raises two questions. First, if we expand the system boundary far enough, does all learning become SLL? Second, if we shrink the boundary small enough, does all learning become KLL?

The answer to the first question is probably "yes," at least in the most extreme case. If our universe is closed and we expand the boundary so that it encompasses the entire universe, then no information or action flows across the boundary. Without inputs, we have seen that KLL is impossible. Certainly without outputs, there is no externally visible behavior, and the whole exercise becomes pointless. In less extreme cases, as long as some subsystem receives inputs (indirectly) from the environment and produces outputs that influence the KL_{cc} behavior of the overall system, then KLL in the subsystem will be observable at the overall system boundary. In the LEX case, the behavior of H does not change the behavior of LEX as a whole at the computation-closure knowledge level, so it is invisible. Hence, in many cases, enlarging the system boundary will *not* automatically "convert" KLL into SLL until the system is entirely isolated from the environment.

The answer to the second question is a "yes", although the reasons are not profound. As we narrow the system boundary, we are likely to cut across lines of communication connecting our selected subsystem with the larger system. Hence, our subsystem will be receiving inputs and producing outputs. If changes in the inputs cause changes in the outputs, then the subsystem will exhibit KLL. Any simple memory unit meets these conditions, since the output produced by the unit depends on the inputs that were stored in it.

5.3 "Justified generalization" is misdirected

A third issue pointed up by our three-part taxonomy of learning systems is that the goal of developing learning systems that produce "justified generalizations" is misguided.

When explanation-based generalization techniques were first developed, they were contrasted with inductive learning techniques (see, e.g., Mitchell, 1983). The latter were criticized for making "unjustified" inductive leaps, whereas the EBG methods are able to determine the correct generalization analytically.

The knowledge level analysis shows that these two kinds of learning systems are attacking two very different goals. The EBG systems are addressing the goal of improving "operationality" or efficiency, while the inductive learning programs are attempting to acquire new knowledge. The reason that EBG generalizations are "justified" is that the EBG systems *already know the right answer.* If a learning system is attempting to acquire knowledge from external sources, it *by definition* does not know the right answer. Therefore, this attempt to compare inductive leaps with explanation-based generalizations is misguided—it compares apples with oranges.

This notion of "justified generalization" turns out to be incorrect for another reason, having nothing to do with knowledge-level learning. Keller (1987) points out that there is no particular reason to assume that concepts expressed using an "operational" vocabulary are actually operational (i.e., actually improve the performance of the system). For example, in LEX2, an analysis of the training example $\int 5x dx$ produces the following rule:

If current state matches $\int r f(x) dx$
Then apply OP3: $\int r f(x) dx \longrightarrow r \int f(x) dx$.

This heuristic will be inefficient for problems like $\int 0x^2 dx$.

5.4 Chunking is not a completely general learning mechanism

The final issue to be discussed in this section concerns the work of the SOAR group in their search for a general learning mechanism. A learning mechanism is a subsystem that, when incorporated into a problem-solving system, confers some learning capabilities on the system.

There are two ways in which a learning mechanism can be "general." One is that it is capable of being incorporated into a wide variety of problem-solving systems. Chunking is a general mechanism in this sense, because it places very few constraints on the problem-solving system.

The second way in which a learning mechanism can be general is if it confers a wide range of learning capabilities. A completely general learning mechanism, in this sense, would result in a system that exhibited all three types of learning discussed in this chapter: SLL, DKLL, and NKLL.

Chunking is clearly not general in this second sense. When chunking is incorporated into a problem-solving system, the only form of learning behavior guaranteed to result is symbol-level learning. This is because chunking is simply a method of generalized caching. It caches the results of previous computations, but it also generalizes those results using a method similar to EBG, so that the memorized "chunk" will apply to situations that are not identical to the situation in which the chunk was developed. The effect is to speed up computations that would have taken place even if the chunk were not present.

Chunking may act in combination with other features of the problem solver to produce DKLL or NKLL. For example, Rosenbloom, Laird, and Newell (1987) have shown that if the problem-solving system accepts inputs from the environment and processes them in a particular way (i.e., as search control on a "reconstructive" problem space), then chunking can be used to store the environmental inputs in long term memory. If these chunks are subseqently processed properly, then the learned information can be applied to aid later problem solving. In short, if the problem-solving system accepts input from the environment, chunking can be used to obtain DKLL.

The larger goal of the SOAR group is to demonstrate how the relatively small number of features of the SOAR architecture can work together to produce a wide range of intelligent behaviors (including, of course, learning behaviors). Hence, while chunking is not itself a general learning mechanism, it appears to be a good building block for providing learning behaviors to SOAR. By itself, it provides SLL. In combination with the simple ability to accept environmental inputs, it provides DKLL. Perhaps some other simple mechanism can be added to SOAR that will provide a way of expressing inductive biases, which could then be combined with chunking to produce NKLL.

6 Summary and Conclusions

Let us review the major points of this chapter.

The knowledge level provides a method for predicting the future behavior of systems based only on their past behavior and the inputs they have received from the environment. The predictions are computed using the rationality principle, which states that rational systems are capable of recognizing the relationships among their knowledge, their goals, and their available actions and taking appropriate actions. Two notions can be defined: knowledge level predictability and knowledge level learning. These notions generate a three-fold taxonomy of learning systems. Symbol-level learning (SLL) systems are predictable at the knowledge level but do not exhibit knowledge level learning. Determined knowledge-level learning (DKLL) systems exhibit both knowledge level predictability and learning. Non-determined knowledge-level learning (NKLL) systems are not predictable at the knowledge level but do exhibit knowledge level learning.

This taxonomy has implications for future research in the development of learning methods. Because SLL is simply correctness-preserving (or goal-preserving) program improvement, new learning methods can be developed by studying the techniques of automatic programming. Because NKLL systems lack a knowledge level description, the knowledge level cannot provide a standard of correct behavior or a way of evaluating alternative NKLL methods. One response to this problem is to extend probability theory to develop a principle of *plausible rationality.* An alternative response, and one that has been explored in much more detail, is to search for learning methods that work well in particular environments. Empirical research, based on comparing learning methods using test data sets, and theoretical research, based on the notion of PAC learning, are both making progress in this direction. However, a major concern is that this research will not lead to any general-purpose NKLL methods.

The taxonomy also formalizes the intuition that there are two distinct kinds of learning: learning that improves existing capabilities of the system and learning that acquires new knowledge (new capabilities) for the system. The definition of knowledge level learning does not require any consideration of the *methods* employed by the learning system, and hence, avoids many of the problems inherent in previous discussions of inductive and deductive reasoning, where it was necessary to assign truth values to internal data structures.

This realization that there are two distinct kinds of learning uncovers some errors in previous discussions of the goals and benefits of explanation-based learning. Explanation-based learning methods are solving a different problem (SLL) than inductive learning methods (NKLL), and therefore, their method of computing "justified generalizations" does not solve the problem of how inductive learning methods should determine which inductive leaps to make.

The distinction between SLL and KLL also clarifies a limitation of the learning method of chunking employed in the SOAR system (and all related methods of explanation-based generalization). Chunking, by itself, is capable only of SLL. The challenge facing the SOAR group is to find other, relatively simple, features of the SOAR architecture that can combine with chunking to produce forms of KLL.

7 Acknowledgements and Historical Note

This chapter addresses the same issues as the paper "Learning at the Knowledge Level" (Dietterich, 1986). It has been completely rewritten in an attempt to correct some of the imprecision and confusion of the earlier paper. In particular, this chapter uses the computational-closure knowledge level to clarify the approach to knowledge level analysis. Additionally, this chapter introduces the notion of knowledge level predictability in place of the notion of deductive describability.

Several people have contributed questions, thoughts, and possible solutions to various problems raised in the first paper. In particular, I thank Paul Rosenbloom and Haym Hirsh for helping me identify the computational-closure knowledge level. Allen Newell answered several questions and attempted to improve my understanding of the knowledge level. Benjamin Grosof, Nicholas Flann, Ritchey Ruff, and James Bennett have participated in many discussions and conversations. The views (and certainly the errors) in this chapter are my own.

This research was supported in part by the National Science Foundation under grant number DMC-8514949 and grant number IRI-8657316 (Presidential Young Investigator Award).

8 References

Angluin, D., Smith, C. H. (1983). A survey of inductive inference: theory and methods, *Computing Surveys*, 15 (3) 237–70.

Barwise, J. & Perry, J. (1983). *Situations and Attitudes.* Cambridge, MA: MIT Press.

Clocksin, W. F., and Mellish, C. S. (1984). *Programming in Prolog*, Berlin: Springer-Verlag.

Dietterich, T. G., (1986). Learning at the knowledge level, *Machine Learning*, 1(3) 287–316.

Dietterich, T. G., London, R. L., Clarkson, K., and Dromey, G. (1982). Learning and inductive inference. Rep. No. STAN-CS-82-913. Department of Computer Science, Stanford University. Appears as Chapter XIV in Cohen, P. R., and Feigenbaum, E. A., *The Handbook of Artificial Intelligence*, Vol. III, 323–512, Los Altos, CA: William Kaufmann, 1982.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

Fisher, D. (1987). Improving inference through conceptual clustering. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-87* (461–465). Seattle, WA. Los Altos, CA: Morgan-Kaufmann.

Fisher, D. (In press). Knowledge acquisition via incremental conceptual clustering. *Machine Learning.*

Flann, N. S., and Dietterich, T. G. (1985). Exploiting functional vocabularies to learn structural descriptions. In *Proceedings of the Third International Workshop on Machine Learning*, Rutgers University. 41–43.

Flann, N. and Dietterich, T. G., (1986). Selecting appropriate representations for learning from examples. In *Proceedings of the National Conference on Artificial Intelligence: AAAI86*, Philadelphia, PA. Los Altos, CA: Morgan-Kaufmann, 460–466.

Genesereth, M. R. (1980). Models and metaphors. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-80* (208–211). Stanford University. Los Altos: Morgan-Kaufmann.

Gold, E. (1967). Language identification in the limit. *Information and Control* 16:447–474.

Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984). Boltzmann Machines: Constraint satisfaction networks that learn. Rep. No. CMU-CS-84-119. Department of Computer Science, Carnegie-Mellon University.

Kearns, M., Li, M., Pitt, L., and Valiant, L. G. (1987). Recent results on Boolean concept learning. *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA. Los Altos: Morgan-Kaufmann. 337–352.

Keller, R. M. (1983). Learning by re-expressing concepts for efficient recognition. In *Proceedings of AAAI-83*. Washington, D. C., August, 1983, 182–186.

Keller, R. M. (1987). Defining operationality for explanation-based learning. *Proceedings of the National Conference on Artificial Intelligence, AAAI-87* (482–487). Seattle, WA. Los Altos, CA: Morgan-Kaufmann.

Kibler, D., and Aha, D. W. (1987). Learning representative exemplars of concepts: an initial case study. In *Proceedings of the Fourth International Workshop on Machine Learning* (24–30), Irvine, CA. Los Altos, CA: Morgan-Kaufmann.

Lenat, D. B., and Brown, J. S. (1984). Why AM and EURISKO appear to work, *Artificial Intelligence*, 23(3) 269–294.

Mahadevan, S. (1985). Verification-based learning: a generalization strategy for inferring problem-reduction methods, *Proceedings of IJCAI-85*, Los Altos, CA: Morgan-Kaufmann, 616–623.

McDermott, D. (1976). Artificial intelligence meets natural stupidity. *SIGART Newsletter*. 57. 4–9.

Michalski, R. S., and Chilausky, R. L. (1980). Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis, *Policy Analysis and information Systems*, 4 (2).

Mitchell, T. M. (1983). Learning and problem solving, *Proceedings of IJCAI-83*, Karlsruhe, West Germany, 1139–1151.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: a unifying view. *Machine Learning*, 1 (1) 47–80.

Mitchell, T. M., Utgoff, P. E., and Banerji, R. B. (1983). Learning by experimentation: acquiring and refining problem-solving heuristics, In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., (eds.), *Machine Learning, Vol. I*, Palo Alto: Tioga.

Mostow, D. J. (1983). Machine transformation of advice into a heuristic search procedure, in R. S. Michalski, J. Carbonell, and T. Mitchell (eds.), *Machine Learning: an artificial intelligence perspective*, Palo Alto: Tioga, 367–404.

Newell, A. (1981). The Knowledge Level. *AI Magazine* 2 (2) 1–20.

Newell, A., and Simon, H., Computer science as empirical inquiry: Symbols and search, *Communications of the ACM*, 19:3, 113-126, 1976.

Nilsson, N. J. (1986). Probabilistic logic. *Artificial Intelligence*, 28 (1), 71–87.

Osherson, D., Stob, M., and Weinstein, S. (1986). *Systems that Learn*, Cambridge, MA: MIT Press.

Quinlan, J. R. (1986). Induction of Decision Trees, *Machine Learning,* 1(1), 81–106.

Quinlan, J. R., Compton, P. J., Horn, K. A., and Lazarus, L. (1986). Inductive knowledge acquisition: a case study. *Proceedings of the Second Australian Conference on Applications of Expert Systems*, Sydney. To appear in Quinlan, J. R. (ed), *Applications of expert systems.* Maidenhead: Academic Press.

Rosenbloom, P. S., Laird, J. E., and Newell, A. (1987). Knowledge level learning in SOAR. *Proceedings of the National Conference on Artificial Intelligence, AAAI-87* (499–504), Seattle, WA. Los Altos, CA: Morgan-Kaufmann.

Russell, S. (1986). Preliminary steps toward the automation of induction. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-86* (477–484). Philadelphia, PA. Los Altos, CA: Morgan-Kaufmann.

Schlimmer, J. C., and Fisher, D. (1986). A case study in incremental concept formation. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-86*, Philadelphia, PA. Los Altos, CA: Morgan-Kaufmann. 496–501.

Scott, P. D. (1983). Learning: the construction of a posteriori knowledge structures. In *Proceedings of AAAI-83*, Washington, D.C. 359–363.

Simon, H. A. (1983). Why should machines learn? In R. S. Michalski, T. M. Mitchell, and J. Carbonell (eds.), *Machine Learning*, Palo Alto: Tioga. 25–38.

Steier, D. M., Laird, J. E., Newell, A., Rosenbloom, P. S., et al. (1987). Varieties of learning in SOAR: 1987. In P. Langley (ed.)*Proceedings of the Fourth International Workshop on Machine Learning.* University of California, Irvine. Morgan-Kaufmann: Los Altos, CA. 300–311.

Utgoff, P. E., Mitchell, T. M. (1982). Acquisition of appropriate bias for inductive concept learning, *Proceedings of AAAI-82*, Los Altos: Morgan-Kaufmann.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27,* 1134–1142.

Winslett, M. (1986). Is belief revision harder than you thought? In *Proceedings of the National Conference on Artificial Intelligence (AAAI-86).* 421–427.