Selecting Appropriate Representations for Learning from Examples

Nicholas S. Flann and Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

Abstract

The task of inductive learning from examples places constraints on the representation of training instances and concepts. These constraints are different from, and often incompatible with, the constraints placed on the representation by the performance task. This incompatibility explains why previous researchers have found it so difficult to construct good representations for inductive learning—they were trying to achieve a compromise between these two sets of constraints. To address this problem, we have developed a learning system that employs two different representations: one for learning and one for performance. The learning system accepts training instances in the "performance representation," converts them into a "learning representation" where they are inductively generalized, and then maps the learned concept back into the "performance representation." The advantages of this approach are (a) many fewer training instances are required to learn the concept, (b) the biases of the learning program are very simple, and (c) the learning system requires virtually no "vocabulary engineering" to learn concepts in a new domain.

1 Introduction

In the idea paper entitled "Learning Meaning," Minsky (1985) stresses the importance of maintaining different representations of knowledge, each suited to different tasks. For example, a system designed to recognize examples of cups on a table would do well to represent its knowledge as descriptions of observable features and structures. In contrast, a planning system employing cups to achieve goals would require a representation describing the purpose and function of cups.

When we turn from the issue of *performance* to the issue of *learning*, it is not clear what representation to choose. The most direct approach is to choose the same representation for learning as for performance, thus gaining the advantage that any knowledge learned will be immediately available to support performance. Early machine learning work, such as Winston's ARCH (Winston 1975) and Michalski's AQ11 system (Michalsk & Chilausky, 1980), employed this approach, and it worked quite well. The design of a structural language capable of capturing the concepts of interest was straightforward, and concepts were learned quickly with (relatively) few training instances.

However, when Quinlan (1982) attempted to pursue this approach in his work on learning chess end-game concepts, he encountered difficulties. His representation for high-level chess features was effective for the task of recognizing end-game positions, but it introduced many problems for the learning task. First, the concept language was very difficult to design. Quinlan spent two man-months iteratively designing and testing the language until it was satisfactory. The second problem was that it took a large number of training instances (a minimum of 334) to learn the concept of *lost-in-3-ply* completely. These problems illustrate that the approach of employing the same representation for learning and for performance was inappropriate for this domain.

In this paper, we show that inductive learning places constraints on the representation for training instances and concepts and that these constraints often conflict with the requirements of the performance task. Hence, the difficulty that Quinlan encountered can be traced to the fact that the concept *lost-in-3-ply* is an inherently functional concept that is most easily learned in a functional representation. However, the performance task (recognition) requires a structural concept representation. The vocabulary that Quinlan painstakingly constructed was a compromise between these functional and structural representations.

The remainder of this paper is organized as follows. First, we discuss the constraints

that the task of inductive learning places on the representation for training instances and concepts. Second, we describe a strategy for identifying the most appropriate representation given these constraints. Third, we consider the problems that arise when the representation for learning is different from the representation in which the training instances are supplied and from the representation that is needed by the performance task. Finally, we describe an implemented system, Wyl, that learns structural descriptions of checkers and chess concepts by first mapping the training instances into a functional representation, generalizing them there, and converting the learned concept back into a structural representation for efficient recognition.

2 Representational Constraints of Inductive Learning

The goal of an inductive learning program is to produce a correct definition of a concept after observing a relatively small number of positive (and negative) training instances. Gold (1967) cast this problem in terms of search. The learning program is searching some space of concept definitions under guidance from the training instances. He showed that (for most interesting cases) this search cannot produce a unique answer, even with denumerably many training instances, unless some other criterion, or bias, is applied. Horning (1969), and many others since, have formulated this task as an optimization problem. The learning program is given a preference function that states which concept definitions are a priori more likely to be correct. The task of the learning program is to maximize this likelihood subject to consistency with the training instances.

This highly abstract view of learning tells us that inductive learning will be easiest when (a) the search space of possible concept definitions is small, (b) it is easy to check whether a concept definition is consistent with a training instance, and (c) the preference function or bias is easy to implement. In practice, researchers in machine learning have achieved these three properties by (a) restricting the concept description language to contain few (or no) disjunctions, (b) employing a representation for concepts that permits consistency checking by direct matching to the training instances, and (c) implementing the bias in terms of constraints on the syntactic form of the concept description.

Let us explore each of these decisions in detail, since they place strong constraints on the choice of good representations for inductive learning.

Consider first the restriction that the concept description language must contain little or

no disjunction. This constraint helps keep the space of possible concept definitions small. It can be summarized as saying "Choose a representation in which the desired concept can be captured succinctly."

The second decision—to use matching to determine whether a concept definition is consistent with a training instance—places constraints on the representation of training instances. Training instances must have the same syntactic form as the concept definition. Furthermore, since the concept definition contains little or no disjunction, the positive training instances must all be very similar syntactically. To see why this is so, consider the situation that would arise if the concept definition were highly disjunctive. Each disjunct could correspond to a separate "cluster" of positive training instances. With disjunction severely limited, however, the positive training instances must form only a small number of clusters.

In addition to grouping the positive instances "near" one another, the representation must also allow them to be easily distinguished from the negative instances. This is again a consequence of the desire to keep the concept definition simple. The concept definition can be viewed as providing the minimum information necessary to determine whether a training instance is a positive or a negative instance. Hence, if the concept definition is to be short and succinct, the syntactic differences between positive and negative instances must be clear and simple.

The third decision—to implement bias in terms of constraints on the syntactic form of the concept description—makes the choice of concept representation even more critical. Recall that the function of bias is to select the correct, or at least the most plausible, concept description from among all of the concept descriptions consistent with the training instances. Typically, the bias is implemented as some fixed policy in the program, such as "prefer conjunctive descriptions" or "prefer descriptions with fewest disjuncts." The bias will only have its intended effect if conjunctive descriptions or descriptions with fewest disjuncts are in fact more plausible. In other words, for syntactic biases to be effective, the concept description language must be chosen to make them true. The net effect of this is to reinforce the first representational constraint: the concept representation language should capture the desired concept as succinctly as possible.

3 Choosing the Most Suitable Representation

Now that we have reviewed the constraints that inductive learning places on the representation, we must consider how to satisfy those constraints in a given learning task. It should be clear that we want to select the representation that captures the concept most "naturally." The "natural" representation is the one that formalizes the underlying reason for treating a collection of entities as a concept in the first place. A concept (in the machine learning sense anyway) is a collection of entities that share something in common. Some entities are grouped together because of the way they appear (e.g., arches, mountains, lakes), the way they behave (e.g., mobs, avalanches, rivers), or the functions that they serve (e.g., vehicles, cups, doors). Occasionally, these categories correspond nicely. Arches have a common appearance and a common function (e.g., as doorways or supports). More often, though, entities similar in one way (e.g., function) are quite different in another (e.g., structure).

The performance task for which a concept definition is to be learned may require a structural representation (e.g., for efficient recognition), a functional representation (e.g., for planning), or a behavioral representation (e.g., for simulation or prediction). When we review the successes and failures of machine learning, we see that difficulties arise when the representation required for the performance task is not the natural representation for the concept.

Winston's *ARCH* program was successful because the natural representation—structural—was also the performance representation. The structural representation captured the important similarities among the positive training instances as a simple conjunction. It also separated the positive instances from the negative ones by simple features such as *touching* and *standing*.

Quinlan's difficulties with *lost-in-3-ply* can be traced to the fact that this concept is naturally defined functionally, yet the performance task required a structural representation. All board positions that are *lost-in-3-ply* are the same, not because they have the same appearance, but because they all result in a loss in exactly 3 moves. This concept can be captured naturally in a representation that includes operators (such as *move*) and goals (such as *loss*). In Quinlan's concept language, which includes both structural and functional terms, this concept required a disjunction of 334 disjuncts.

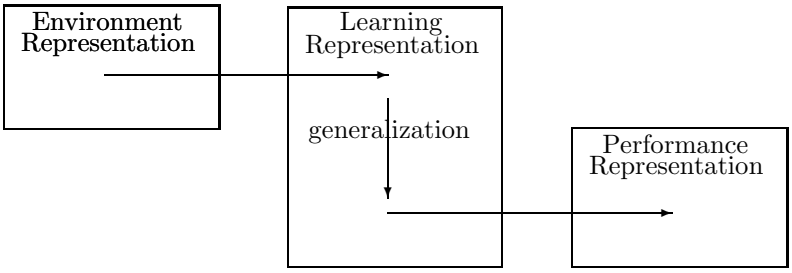

Figure 1: The Multiple Representation Strategy

4 Coordinating Different Representations

For situations in which the representation most appropriate for learning is different from the one required for the performance task, there are two basic approaches that can be pursued. First, we can try, as Quinlan did, to find an intermediate representation that provides some support for both learning and performance. However, the alternative that we have investigated is to employ two separate representations—one for learning and one for performance. This raises the problem of converting from one representation to another.

Figure 1 shows the general structure of a learning system that employs this "multiple representation strategy." Training instances are presented to the system in a representation called the "Environment Representation" (ER). To support induction, the instances are translated into training instances written in the "Learning Representation" (LR). Within this representation, the instances are generalized inductively to produce a concept description. For this concept to be employed in some performance task, it must be translated into the "Performance Representation (PR)."

Many existing learning systems can be viewed as pursuing variants of this "multiple representation strategy." For example, consider the operation of Meta-DENDRAL (Buchanan & Mitchell, 1978). Training instances are presented in a structural ER consisting of molecular structures and associated mass spectra. The program INTSUM converts these structural training examples into a LR of behavioral descriptions called cleavage processes, which are sequences of cleavage steps. These individual cleavage steps are then inductively generalized by programs RULEGEN and RULEMOD to obtain general cleavage rules. In this domain, the PR is the same as the LR. These cleavage rules are produced as the output of Meta-DENDRAL for use in predicting how other molecules will behave in the mass spectrometer.

One can imagine an implementation of Meta-DENDRAL that attempted to inductively generalize the training instances in the given structural ER of molecules and mass spectra. However, this representation does not capture the important similarities between different molecules. The similarities are properly captured at the level of individual cleavage steps that produce single spectral lines, rather than entire molecules and spectra.

In addition to Meta-DENDRAL, most of the explanation-based learning systems (e.g., Mitchell, Keller, & Kedar-Cabelli, 1986; DeJong & Mooney, 1986) can also be viewed as employing a version of this multiple representation strategy. In LEX2 (Mitchell, et al., 1982), for example, training instances are presented in an ER consisting of structural descriptions of symbolic integration problems. By applying its integration problem solver, LEX2 converts each training instance into a functional representation (the LR) consisting of a particular sequence of integration operators leading to a solution. In this LR, the specific sequence of integration operators is generalized by permitting the final state to be *any* solved problem state. In some sense, LEX2 is assuming that the teacher is trying to teach it the concept of "all integration problems solvable by this particular sequence of operators." Once it has developed this generalized concept description in the LR, LEX2 must convert it into the PR, which is the same representation as the ER. This translation is accomplished by back-propagating the description of a solved problem through the operator sequence to compute the weakest preconditions of this particular operator sequence.

This view of LEX2 explains why the original LEX system was not as successful as LEX2. In LEX, inductive inference was applied to positive training examples represented in the ER. The goal of inductive learning was to find general structural descriptions of integration problems for which particular operators, such as OP3, should be applied. This knowledge of the learning goal was not explicit in the structural representation, but only in the teacher's mind. Hence, LEX could not take advantage of it. However, by mapping the training examples into the functional representation, the learning goal could be made explicit and used to guide the generalization process. The functional representation concisely captures the desired similarity between the different training examples.

5 Overview of Wyl

Although previous learning systems can be viewed as applying the multiple representation strategy, none of these systems fully exploits this approach. In particular, the explanation-

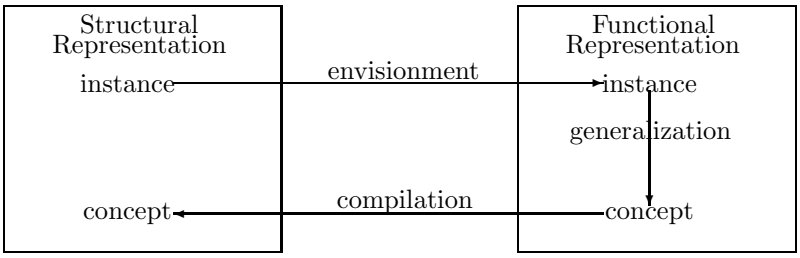

Figure 2: Representations in Wyl

based learning systems do not perform any significant inductive inference in the LR aside from generalizing the final state of the operator sequence. In order to explore the multiple-representation strategy, we have developed a learning system named Wyl (after James Wyllie, checker champion of the world from 1847 to 1878) that applies the strategy to learn concepts in board games such as checkers and chess. We have chosen this domain because there are many interesting concepts that are naturally functional (e.g., *trap*, *skewer*, *fork*, *lost-in-2-ply*) and yet have complex structural definitions. Wyl has been applied to learn definitions for *trap* and *trap-in-2-ply* in checkers and *skewer* and *knight-fork* in chess.

The performance task of Wyl is recognition. Given a board position, represented simply in terms of the kinds and locations of the playing pieces, Wyl must decide whether that position is, for example, a *trap*. To perform this task, the *trap* concept must be represented in a structural vocabulary that permits efficient matching against the board positions. However, as we have noted above, concepts such as *trap* are most easily learned in a functional representation.

In addition to requiring a structural representation for performance, a structural representation is also needed for the training instances. To teach Wyl checkers and chess concepts, we want to simply present board positions that are examples of those concepts. Hence, in the terminology of the previous section, the ER and the PR are structural representations, but the LR is a functional representation.

The organization of Wyl is shown in Figure 2. The three main processes in Wyl are envisionment, generalization, and compilation. The envisionment process translates each supplied structural training instance into the functional representation to obtain the corre-

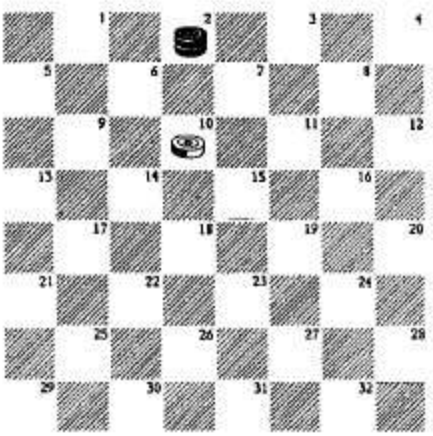

Figure 3: Checkers *Trap* training instance, red to play

sponding *functional training instance*. The generalization process performs inductive inference on these functional training instances resulting in a functional definition that captures the desired concept. Finally the compilation stage converts this functional definition into an equivalent structural description that can support efficient recognition.

The initial knowledge given to Wyl takes four forms. First, there is the environment representation for board positions. Second, there is a representation for each of the legal operators in the game (e.g., *normal-move* and *take-move*). Third, Wyl is given the rules of the game, represented as a recursive schema that describes what moves are legal at what points in the game. Finally, Wyl is given definitions of the important goals of the game, such as *loss*, *win*, and *draw*. For chess, Wyl is also told that *lose-queen* is an important goal.

These given goals are the key to Wyl's learning ability. Wyl learns new functional concepts as *specializations* of these known concepts. For example, the checkers concept *trap* is a specialization of *loss*. To see this, consider the particular *trap* position shown in Figure 3. In this position, the red king in square 2 is trapped by the white man at square 10. No matter what move the red king makes, the white man can take him. Hence, *trap* is a particular way to lose a checkers game. Once Wyl learns a recognition predicate for *trap*, it is added to the pool of known concepts, where it may be specialized further to form some future concept (such as *trap-in-2-ply*).

The goals are provided to Wyl in a predicate calculus notation. The checkers concept of *loss* is represented below (*win* is the dual case):

$$
\begin{array}{l}
\forall\, state1\ side1\ LOSS(state1\ side1) \Leftrightarrow \\
\quad recognizedLOSS(state1\ side1) \\
\quad \vee \forall\ \ state2\ side2\ type\ from\ over\ to \\
\qquad oppositeplayer(side1\ side2) \\
\qquad \wedge [\ \ [\ takemove(state1\ state2\ from\ over\ to\ side1\ type) \\
\qquad\quad \wedge WIN(state2\ side2)] \\
\qquad\ \ \vee\ \ [normalmove(state1\ state2\ from\ to\ side1\ type) \\
\qquad\quad \wedge WIN(state2\ side2)]].
\end{array}
$$

This formula is interpreted as follows. A board is an instance of *loss* if, for all legal moves available to *side*1, the outcome is a *win* for the other player (*side*2). In checkers, there are two kinds of moves: *takemove*s, in which one piece captures another by jumping over it, and *normalmove*s, in which a piece simply moves one square.

This completes our overview of the Wyl system and the information that it is initially given. The following three sections describe each of the main phases of the program: envisionment, generalization, and compilation. We illustrate the operation of Wyl as it learns the checkers concept of *trap*.

5.1 Envisionment

Wyl starts with a given structural training instance (i.e., board position), which it is told is an instance of *trap*. In Figure 3, we illustrate the first training instance for *trap*, with red to play. The structural representation of the instance State1 is

$$
\begin{array}{l}
occupied(\mathsf{State1\ s2\ rk1}) \wedge \\
occupied(\mathsf{State1\ s10\ wm1}) \supset TRAP(\mathsf{State1\ red}).
\end{array}
$$

Where rk1 and wm1 are playing pieces, described as
$type(\mathsf{wm1\ man}) \wedge side(\mathsf{wm1\ white}) \wedge type(\mathsf{rk1\ king}) \wedge side(\mathsf{rk1\ red})$.

To convert this into a functional instance, Wyl applies a special proof procedure to State1. This proof procedure has the effect of conducting a minimax search to look for known goals. When a known goal is discovered, the proof procedure returns a minimax search tree in which each state is marked with its outcome.

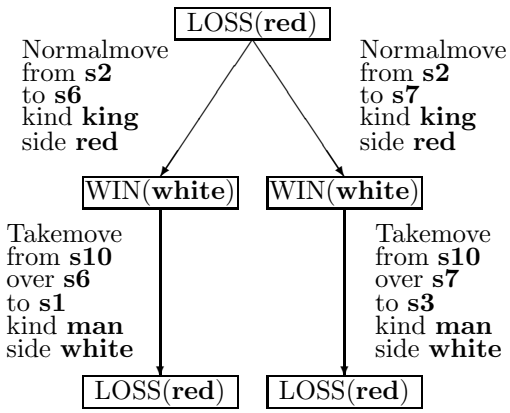

Figure 4: Functional training instance for *trap*

In our *trap* example, the proof procedure discovers that the board position is an instantiation of the concept *loss*, with each node representing a state in the search and each branch representing the particular operators in the search. The first operators instantiated are the *normalmoves* from square s2. These are followed by *takemoves* that lead to an instantiation of the predicate *recognizedLOSS* and termination of the search.

The next step is to convert this minimax tree into an explanation tree (along the lines of Mitchell, et al., 1986). An explanation tree is a proof tree that explains the computed outcome (i.e., *loss*) of the training instance. The minimax tree contains all of the information needed to construct this proof, but usually it also contains extra information that is irrelevant to the proof. Hence, Wyl traverses the minimax tree to extract the minimum (i.e., necessary and sufficient) conditions for the proof of the outcome. Figure 4 shows the functional training instance that is produced by this process.

5.2 Generalization

This functional instance describes a particular way to lose a checkers game. It is a conjunction of two fully instantiated (i.e., ground) sequences of operators, each resulting in a *loss*. If Wyl were to follow the standard paradigm of explanation-based learning, it would now attempt to find the weakest precondition of this particular operator graph that would result in a *loss*. However, this is not the concept of *trap* that the teacher is trying to get Wyl to learn, because it only describes traps in which the trapped piece has two alternative moves. There are other traps, against the sides of the board, in which the trapped piece has only one

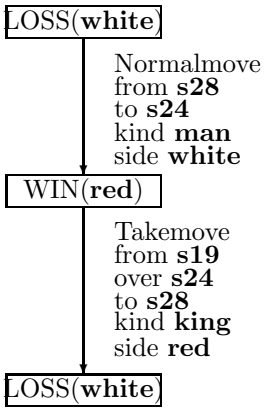

Figure 5: Second functional training instance of *trap*

possible move. Hence, rather than having Wyl generalize based on one training instance, we provide it with several training instances and allow it to perform inductive inference on the functional representations of these instances.

To demonstrate this generalization process, let us present Wyl with a second (very well-chosen) training instance. This structural training instance can be expressed in logic as

$$occupied(\mathsf{State8\ s28\ wm1}) \wedge$$
$$occupied(\mathsf{State8\ s19\ rk1}) \supset TRAP(\mathsf{State8\ white}).$$

In this instance, a red king has trapped a white man against the east side of the board (see Figure 3). Wyl performs the envisionment process and discovers that this situation again leads to a *loss*—this time for white. The minimax tree is a simple sequence of moves, because white has only one possible move. Figure 5 shows the resulting functional training instance.

Now that two training examples have been presented, Wyl is able to perform some inductive generalization. Two simple and strong biases are employed to guide this induction.

The first is the familiar bias toward maximally-specific generalizations. The two functional instances are generalized as little as possible. The second bias can be stated as "There are no coincidences." More concretely, if the same constant appears at two different points within a single training instance, it is asserted that those two different points are *necessarily* equal.

The result of applying these two inductive biases to the training instances is shown in Figure 6. In order to make the two separate branches of the first training instance match the

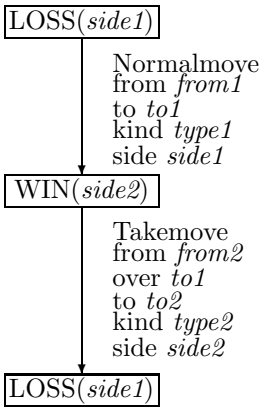

Figure 6: Generalized functional definition of *trap*

single branch in the second instance, Wyl must generalize to a single universally-quantified line of play that allows *any number* of branches. Similarly, in order to make the kinds, types, and locations of the pieces match, Wyl must generalize all of these specific constants to variables. However, these variables are not completely independent of one another. First, the two sides are known to be opposing. Second, the no-coincidences bias is applied to ensure that the square that the first piece moves to ($to1$) is the same as the square that the second piece jumps over in the *takemove*.

Because we chose these two training examples carefully, this generalized functional description is the correct definition of *trap*. This functional definition of *trap* can be expressed in logic as

$$\begin{aligned}
&\forall\ state1\ side1\ from1\ from2\ TRAP(state1\ side1) \Leftrightarrow \\
&\quad \forall\ state2\ type1\ to1 \quad oppositeplayer(side1\ side2) \\
&\qquad \wedge normalmove(state1\ state2\ from1\ to1\ side1\ type1) \\
&\qquad \wedge\ \exists\ state3\ type2\ to2 \\
&\qquad\quad takemove(state2\ state3\ from2\ to1\ to2\ side2\ type2) \\
&\qquad\quad \wedge recognizedLOSS(state3\ side1).
\end{aligned}$$

5.3 Compilation

The third stage of the learning process is to translate the functional knowledge into a form suitable for recognition—that is, to re-describe the acquired functional concept in the PR.

This is a difficult task, because, unlike LEX2, Wyl is not given a good vocabulary for the performance language. The only structural representation that Wyl receives from the teacher is the representation used to describe individual board positions. This language *could* be used to represent the structural concept, but for *trap* this would require a large disjunction with 146 disjuncts. For other functional concepts, this approach is clearly infeasible.

Instead of employing the same low level structural language in which the training instances were presented, Wyl must construct its own structural concept language for expressing the functional concept.

Currently, there are no methods capable of designing such a structural language automatically. The only method that provides even a partial solution to this problem is the method of constraint back-propagation or goal regression (Mitchell, et al., 1986). Utgoff (1986) shows that this method can create new structural terms to extend a structural language. We are experimenting with extensions to his method to construct terms in chess and checkers, but to date we do not have a fully satisfactory method.

Instead, we have explored an alternative (and extremely inefficient) approach in Wyl based on generation and clustering. First we apply the functional definition to generate all possible structural examples of the concept (i.e., all possible board positions that are *trap*s according to the functional definition). This can be viewed as a highly disjunctive description of the concept in the supplied environment language. Next the large number of disjunctions in the description is reduced by a compaction process that creates simple new terms.

The generator works by employing the functional concept as a *constructive* proof, generating all possible board positions consistent with the concept. (We employ an extension of the Residue inference procedure of the MRS system; see Russell, 1985.) Each *trap* position generated is a conjunction of the two single *observable* facts like the structural trap examples given above and illustrated in Figure 3. In the *trap* case, a disjunction of 146 possible positions is generated. The compaction stage then applies two algorithms to compress this set of 146 positions into a disjunction of 11 (more general) descriptions.

The first algorithm discovers simple relational terms that describe relationships between squares. For example, in the first training example of *trap* (State1), the white king is directly two squares south of the red king. As part of Wyl's initial environment language, primitive facts are given that describe the relationship between any square on the board and its immediate neighbors. The neighbors of s2 are $sw(\mathsf{s2}, \mathsf{s6})$ and $se(\mathsf{s2}, \mathsf{s7})$. The algorithm identifies new relational terms by a simple breadth-first search from one of the squares in an

instance to discover paths to the others. From State1, a disjunction of two terms is found:

$$\forall\ square1\ square2\ square3\ South2squares(square1\ square2) \Leftrightarrow$$
$$[se(square1\ square3) \wedge sw(square3\ square2)]$$
$$\vee[sw(square1\ square3) \wedge se(square3\ square2)]$$

The second term-creation algorithm is similar to GLAUBER (Langley et al., 1986) and identifies common internal disjunctions over the primitive structural features. The structural instances created by the generator are translated into a feature-vector representation based on the primitive attributes. For example, State1 is translated to the following vector:

$TRAPvector$(red king s2 white man s10).

The first three items describe the red king, the following three, the white man. Next, one of the squares is replaced by its relationship with the other. The new relational term $South2squares$ is used, and it yields the new instance:

$TRAPvector$(red king s2 white man South2squares).

Common disjunctions are found by locating sets of instance vectors that share all but one feature in common. For example, consider two trap positions, the initial training instance and a red king on s3, white man on s11 given below:

$TRAPvector$(red king s2 white man South2squares)
$TRAPvector$(red king s3 white man South2squares)

The algorithm identifies the set of squares {s2, s3}, which is named $NorthCenterSide$. All of the features can be used to form the new terms. Using the *trap* instances, this algorithm creates terms defining regions such as $Center$ {s6, s7, s8, s14, s15, s16, s22, s23, s24}, $NorthSingleSide$ {s4, NorthCenterSide}. Directional relationships between squares produce terms such as $North$, {ne, nw} and $AnyDirection$, {North, South}. In all, Wyl discovers 13 descriptive terms of this kind, along with 6 relational terms like $South2squares$.

While this method is very successful in constructing new terms, it is clearly unsatisfactory, since it does not scale up to domains having large or infinite numbers of structural instances. Even for *trap* this algorithm requires several hours of CPU time on a VAX 11/750. We are optimistic that a more direct algorithm, based on goal regression, can be developed.

6 Relationship to Previous Research

It is informative to compare Wyl to previous work in explanation-based learning. If we were to apply the explanation-based learning paradigm of Mitchell, et al. (1986), we would need to provide Wyl with four things: (a) the goal concept, (b) the domain theory, (c) the operationality criterion, and (d) the training instance. In the checkers domain, the goal concept would be a functional definition of *trap*. The domain theory would be the rules of checkers and the goals of *win*, *loss*, and *draw*. The operationality criterion would state that the final definition should be given in a structural vocabulary. The training instance would, of course, be a structural example of a *trap*. When we consider Wyl, we see that all of these things have been provided *except* for the goal concept. Wyl can be said to acquire the goal concept by inductive inference from the training examples.

An example from LEX2 will clarify this point. In LEX2, one goal concept is *Useful-OP3*, that is, the set of all integration problems that can be solved by applying a sequence of operators beginning with operator OP3. The domain theory consists of the definition of solvable and solved problems. Imagine a new version of LEX2 constructed along the lines of Wyl (call it WYLLEX). WYLLEX would be given the domain theory, the operationality criterion and several training instances, but no goal concept. For each (structural) training instance, it would apply its domain theory to convert it into a functional instance. Suppose we are trying to teach WYLLEX the concept of *Useful-OP3*. We would present positive examples of problems for which applying OP3 leads to a solution. When WYLLEX converted these to functional instances, they would each consist of a sequence of operators beginning with OP3 and ending in a solved problem. Hence, WYLLEX could perform inductive inference on these functional instances and *derive* the concept of *Useful-OP3*.

Notice that in order to convert the structural instances into functional instances, WYLLEX must already have a concept more general than the goal concept, namely, the concept of *solvable problem*. Similarly, Wyl starts out with knowledge about the very general goals of *win*, *loss*, and *draw* and learns more specific goals such as *trap* and *trap-in-2-ply*. (The same is true in Meta-DENDRAL, where all concepts learned are specializations of the initial "half-order theory.") This is not a serious limitation, because it is reasonable to assume that all intelligent agents have available a hierarchy of goals rooted in goals like *survival* and *minimize-resource-consumption*.

An area of work closely related to explanation-based learning is the work on purpose-

based analogy (Winston, et al., 1983; Kedar-Cabelli, 1985). The constraints imposed on representations by inductive learning are exactly those imposed by analogical reasoning. For two items to be analogous, they must have some commonality. That commonality is often not expressed in surface (observable) features, but in function. A hydrogen atom is like our solar system not because of size or color, but because of the way the respective components interact. So, the best representation for analogical reasoning about different items is one in which their underlying similarity is captured syntactically. The work in Wyl suggests that new analogies may be exploited to identify new functional concepts as specializations of existing goals.

7 Conclusion

In this paper, we have argued that inductive learning is most effective when the concept language captures the "natural" similarities and differences among the training instances. We have also shown that in some domains the representation required for efficient performance is not the "natural" one. To resolve this difficulty, we proposed the "multiple-representation strategy" whereby the learning system translates the training examples into a "natural" representation for inductive learning and then translates the learned concepts into the appropriate performance representation. We have tested this strategy by implementing the Wyl system, which learns functional concepts from structural examples in chess and checkers. Wyl demonstrates three key advantages of this strategy: (a) fewer examples are required to learn the concept, (b) the bias built into the program is very simple (maximally-specific generalization), and (c) the representation language requires little or no domain-specific or concept-specific engineering.

Our analysis of Wyl suggests that previous learning systems can be usefully viewed as pursuing simpler variants of this multiple-representation strategy. This suggests that part of the power of these learning systems derives from the choice of representation (as well as from the use of a domain theory).

8 Acknowledgments

The authors wish to thank Bruce Porter for reading a draft of this paper. The AAAI referees also made many helpful comments. This research was partially supported by a

Tektronix Graduate Fellowship (to Flann) and by the National Science Foundation under grant numbers IST-8519926 and DMC-8514949.

9 References

Buchanan, B. G. and Mitchell, T. M., "Model-Directed Learning of Production Rules," in *Pattern-Directed Inference Systems*, Waterman, D. A. and Hayes-Roth, F. (Eds.), Academic Press, New York, 1978.

DeJong, G., and Mooney, R., "Explanation-Based Learning: An alternative view," *Machine Learning 1*, 1986.

Gold, E. "Language identification in the limit." in *Information and Control,* Vol 16, 447–474, 1967.

Horning, J. J. "A Study of grammatical inference." *Rep. No. CS-139,* Computer Science Department, Stanford University. 1969.

Kedar-Cabelli, S.T., "Purpose-Directed Analogy," in *Proceedings of the Cognitive Science Society,* Irvine, Calif., 1985.

Langley, P. W., Zytkow, J., Simon, H. A., and Bradshaw, G. L., "The search for regularity: Four Aspects of Scientific Discovery," in *Machine Learning: An Artificial Intelligence Approach, Vol II.* Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tioga Press, Palo Alto, 1986.

Michalski, R. S. and Chilausky, R. L., "Learning by Being Told and Learning from Examples: An Experimental Comparison of Two Methods of Knowledge Acquisition," *Policy Analysis and Information Systems,* Vol. 4, No. 2, June 1980.

Minsky, M. "Society of mind," Technical Report, Massachusetts Institute of Technology, (1985).

Mitchell, T.M., Utgoff, P. E. and Banerji, R., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics", in *Machine Learning: An Artificial Intelligence Approach, Vol I.,* Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tioga Press, Palo Alto, 1982.

Mitchell, T., Keller, R., and Kedar-Cabelli, S. "Explanation-Based Generalization: A Unifying View," in *Machine Learning 1, 1*, 1986.

Quinlan, J. R., "Learning Efficient Classification Procedures and their Application to Chess End Games" in *Machine Learning: An Artificial Intelligence Approach, Vol I.* Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tioga Press, Palo Alto, 1982.

Russell, S., "The Compleat Guide to MRS," Rep. No. KSL-85-12, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 1985.

Utgoff, P. E., "Shift of Bias for Inductive Concept Learning," in *Machine Learning: An Artificial Intelligence Approach, Vol II.* Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tioga Press, Palo Alto, 1986.

Winston, P., Binford, T., Katz, B. and Lowry, M. "Learning Physical Descriptions from Functional Definitions, Examples and Precedents," *Proceedings of AAAI–83*, Washington, D.C., 1983.

Winston, P. H., "Learning Structural Descriptions from Examples," in *The Psychology of Computer Vision*, Winston, P. H. (Ed.), McGraw Hill, New York, Ch. 5, 1975.