OREGON STATE UNIVERSITY COMPUTER SCIENCE DEPARTMENT

SELECTING APPROPRIATE REPRESENTATIONS FOR LEARNING FROM EXAMPLES

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

86-30-5

Selecting Appropriate Representations for Learning from Examples

Nicholas S. Flann & 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.

Submitted to AAAI-86, Philadelphia, PA

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 Buchanan & Mitchell's Meta-DENDRAL system (Buchanan & Mitchell, 1978), 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 (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.

Now that we have reviewed the constraints that inductive learning places on the representation of training instances and concepts, we can explain why some machine learning systems have been more successful than others. Consider Winston's ARCH program. In his structural representation, the positive examples of arches are all very similar (three objects that are restricted in shape and arrangement). Negative examples—non-arches—are all easily distinguished by some simple observable features such as *touching* or *standing*. This explains why the ARCH system was so successful.

In contrast, consider Quinlan's work on the chess concept *lost-in-3-ply*. When positive and negative examples of *lost-in-3-ply* are represented as simple board positions, there are no obvious distinguishing features. Moving a piece one square often changes the classification of an instance. When we consider that there are 1.8 million distinct training instances, it is clear that inductive learning in this low level representation would require that the concept description include vast numbers of disjuncts. It is not surprising that Quinlan chose to design a higher-level vocabulary for describing his training instances.

In his (1982) article, Quinlan showed how one could evaluate the correctness of an inductive learning program by asking how many training examples are required for the program to discover a concept of a given complexity (i.e., with a given number of disjuncts). His definition of a perfect learning program was one that required only one positive training example for each disjunct in the concept definition. Such a learning program would possess a perfect bias.

We can turn this analysis around and use it to evaluate the combined appropriateness of the bias and the representation language. If a program requires few training instances to discover a concept, then the combination of the bias and representation is working well to select the proper concept definition.

By this measure, the ARCH program has an excellent bias and representation language, since very few training instances are required. On the other hand, even after Quinlan carefully engineered the representation language so that it included high level structural and functional terms, his system required 334 training instances to learn the concept *lost-in-3-ply*. This indicates that the representation language and the bias were still not very appropriate for learning this concept.

We can summarize this section by stating the following constraints on the choice of representation languages for inductive learning. First, the language should be able to represent the desired concept succinctly (i.e., conjunctively or as a short disjunction). Second, the training instances should have the same form as the concept definition. Third, the representation should capture semantic similarities among the positive training instances directly in syntactic forms. Fourth, the representation should capture semantic differences between the positive and negative training instances syntactically.

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 behavior 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.

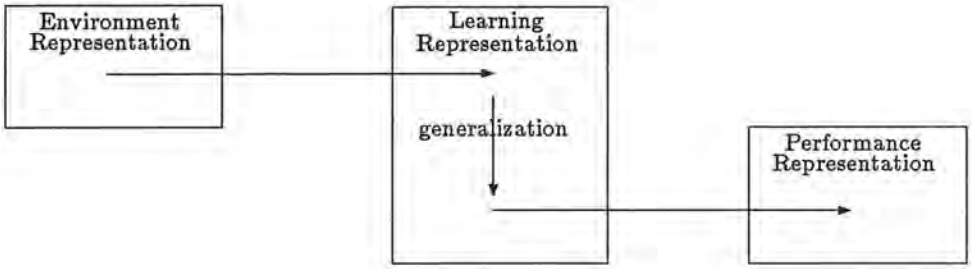

Figure 1: The Multiple Representation Strategy

Winston's *ARCH* program was successful because the natural representation—structural—was also the performance representation. 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*).

Another example of a case where the natural representation is functional and the performance representation is structural is the concept *cup* (Winston, et al., 1983; Kedar-Cabelli, 1985). Winston considers the task of teaching a machine vision system how to recognize cups. For this recognition task, a structural representation is desired. However, the natural representation is functional. Cups are strongly distinguished from many other objects in the universe by their function of being "something you can drink from." In contrast, structural descriptions of cups are quite distinct from one another and yet quite similar to objects that are not cups. There exist many changes to a cup (such as drilling holes in it) that cause only small and subtle changes in appearance and yet convert the cup into a "non cup." If we wish to build a system which could inductively learn *cup*, we would do well to employ a functional representation that included terms such as *liftable* and *stable*.

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 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 this "multiple representation strat-

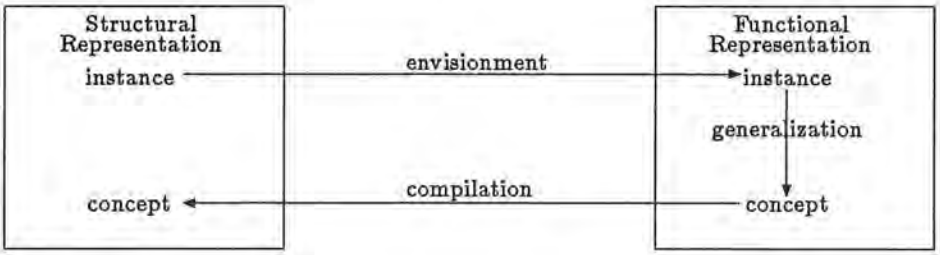

Figure 2: Representations in Wyl

egy." One instructive example is the ANALOGY system of Winston, et al., (1983). Here the training instances are functional descriptions written in English. The ER is therefore a subset of the English language. The LR captures functional descriptions and is implemented as a semantic net employing causal relationships between functional terms. The knowledge transformation method from ER to LR is a two-phase process of (a) parsing the English description and (b) generating the appropriate semantic net form. The performance task is one of recognition, so the PR is mostly structural. The translation in this case uses knowledge about how functional characteristics map to structural features.

5 Overview of Wyl

We have developed a learning system named Wyl (after James Wyllie, checker champion of the world from 1847 to 1878) that applies the multiple representation strategy to learn concepts in board games such as checkers and chess. We have chosen this domain because there are simple and complete "domain theories" available and 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 *1-move-to-trap* 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. Wyl learns from positive instances only. These training instances are board positions, represented in an environment representation of simple structural features—namely, the kinds and locations of the playing pieces.

Wyl converts these training instances into a functional representation through a process of envisionment. The purpose of envisionment is to determine how a given board position relates to the known goals of the players (e.g., *loss* or *win*). Wyl knows the rules of checkers (and chess), so it is able to conduct a forward minimax search to see what outcomes the given board position might lead

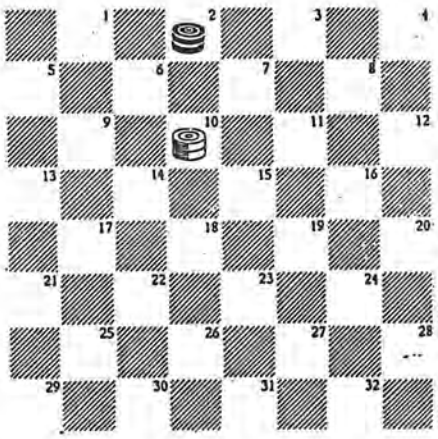

Figure 3: Checkers *Trap* training instance **State1**

to. Once it has related the board position to some known goal, it constructs an AND/OR graph that explains the relationship. We call this AND/OR graph a *functional training instance*. Wyl employs functional training instances to conduct inductive inference in the functional representation. This results in a functional concept definition that captures the desired concept.

The final task is to convert this functional definition into an equivalent structural description that can support efficient recognition. This is accomplished through a compilation process that generates, as a side-effect, a generalized structural vocabulary for representing the concept.

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 king at square 10. No matter what move the red king makes, the white king 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.

The goals are provided to Wyl in a predicate calculus notation. The checkers concepts of *loss* and *win* are represented as

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

and

$$\forall\, state1\ side1\ WIN(state1\ side1) \Leftrightarrow recognizedWIN(state1\ side1)$$
$$\vee \exists\ state2\ side2\ type\ from\ over\ to$$
$$oppositeplayer(side1\ side2)$$
$$\wedge [\ [\ takemove(state1\ state2\ from\ over\ to\ side1\ type)$$
$$\wedge LOSS(state2\ side2)]$$
$$\vee\ [normalmove(state1\ state2\ from\ to\ side1\ type)$$
$$\wedge LOSS(state2\ side2)]].$$

These formulas are 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). A board is an instance of *win* if there exists a legal move for *side*1 that results in a *loss* for the other player. In checkers, there are two kinds of moves: *takemoves*, in which one piece captures another by jumping over it, and *normalmoves*, in which a piece simply moves one square.

The alternating quantification over states is a product of the "adversary game" domain. In general the quantification at each depth is dependent upon whether the outcome is favorable to the current player. To prove that a position is unfavorable for *side*1, we must prove that *all* the available moves lead to an unfavorable result. To prove that a position is favorable, on the other hand, we need only prove that *there exists* a move that leads to a favorable result.

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 (converting a structural training instance into the functional representation), generalization (inductive inference in the functional representation), and compilation (converting the functional concept description into a structural representation). 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

$$occupied(\text{State1 s2 rk1}) \wedge$$
$$occupied(\text{State1 s10 wk1}) \supset TRAP(\text{State1 red}).$$

Where rk1 and wk1 are playing pieces, described as

$$type(\text{wk1 king}) \wedge side(\text{wk1 white}) \wedge type(\text{rk1 king}) \wedge side(\text{rk1 red}).$$

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 (see Figure 4(A)) in which each state is marked with its outcome.

In our *trap* example, the proof procedure discovered 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

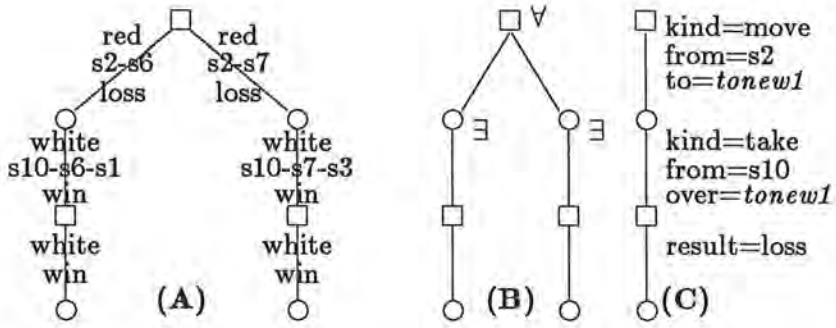

Figure 4: Generalization of Checkers *Trap* instance from Figure 3

(i.e., *loss*) of the training instance. The minimax tree contains all of the information needed to construct this proof, but 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. When the quantification is $\exists$, an OR node is formed, and Wyl extracts sufficient conditions of the proof. Only the one operator and its traversed subtree leading to the computed outcome are needed for the proof. The dual case is when the quantification is $\forall$ and an AND node is formed. Here the current player was at a disadvantage, and the moves were therefore *forced*. In this case, all of the operators (i.e., moves) are needed for the proof. Figure 4(B) shows the outline of 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*. On the other hand, the desired concept *trap* is described as a single generalized sequence of operators ending in a *loss*. To form a more general concept description from this instance, the generalization step first "compresses" the two parallel branches of the instance tree to obtain a single generalized line of play. Two simple and strong biases are applied to drive this generalization.

The first bias is the familiar bias toward maximally-specific conjunctive generalizations. The final functional concept definition must be a conjunction of generalized instantiations of the concept *loss*. To implement this bias, the inverse-enumeration rule (illustrated below) is applied recursively to the predicates present in the explanation tree (i.e., to *normalmove*, *takemove* and *recognizedLOSS* and so on).

$$P(A_1\ B_1 \ldots Z_1) \wedge P(A_2\ B_2 \ldots Z_2) \wedge \ldots \wedge P(A_n\ B_n \ldots Z_n) \Rightarrow \forall a\ b \ldots z P(a\ b \ldots z)$$

The second bias employed by Wyl states that "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. Figure 4 demonstrates this bias. In the left branch of tree (A), the red piece first moves to square s6. Then the white piece in square s10 captures the piece in square s6. The "no coincidences" bias says that these two occurrences of s6 were necessarily identical. Hence, when they are matched against the right branch of tree (A) to s7, they are both generalized to the *same* variable *tonew*1. The resulting functional concept description is the following:

$\forall$ *state*1 $TRAP$(*state*1 red) $\Leftrightarrow$
    $\forall$ *state*2 *tonew*1 *oppositeplayer*(red white)

$\wedge normalmove(state1\ state2\ s2\ tonew1\ \text{red king})$
$\wedge\ \exists\ state3\ tonew2$
$takemove(state2\ state3\ s10\ tonew1\ tonew2\ \text{white king})$
$\wedge recognizedLOSS(state3\ \text{red}).$

This states that a *trap* is a board position in which your opponent has a piece (at s10) for which *there exists* a take move that captures your piece (at s2) *for all* moves available. This is an overly specific version of *trap*, since it only describes traps that occur at one particular place on the board. The correct definition of *trap* can be obtained by presenting further training instances.

Indeed, one more well-chosen training instance suffices to teach *trap*:

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

In this example, a red king has trapped a white man against the east side of the board. The minimax search again discovers that this situation leads to a *loss*—this time for white. The minimax tree is a simple sequence of moves, because white has only one possible move. Hence, the compaction process is trivial. The resulting functional training instance is

$\forall\ state1\ \ TRAP(state1\ \text{white}) \Leftrightarrow$
$\forall\ state2\quad oppositeplayer(\text{white red})$
$\wedge\ normalmove(state1\ state2\ \text{s28 s24 white man})$
$\wedge\ \exists\ state3$
$takemove(state2\ state3\ \text{s19 s24 s28 red king})$
$\wedge recognizedLOSS(state3\ \text{white}).$

Notice that only the specific starting state, State8 has been generalized (to become *state*1). The specific state names are always generalized by Wyl.

When the second training instance is matched to the functional concept description, the following (correct) functional definition of *trap* is obtained.

$\forall\ state1\ side1\ fromnew1\ fromnew2\ TRAP(state1\ side1) \Leftrightarrow$
$\forall\ state2\ type1\ tonew1\quad oppositeplayer(side1\ side2)$
$\wedge normalmove(state1\ state2\ fromnew1\ tonew1\ side1\ type1)$
$\wedge\ \exists\ state3\ type2\ tonew2$
$takemove(state2\ state3\ fromnew2\ tonew1\ tonew2\ side2\ type2)$
$\wedge recognizedLOSS(state3\ side1)$

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. However, 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. Consider trying to represent all possible cups in terms of perceived lines and surface shading.

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.

There are two basic approaches to constructing a structural concept language, and both involve solving the "new term problem." First, the new terms can be created that describe more abstract, but still structural features, by some method of clustering. For example, terms such as circle, cylinder, and handle could be viewed as clusters of primitives in the cup environment language. Second, a more satisfactory solution is to create mixed structural and functional terms that optimize the recognition procedure. The aim of this approach would be to automatically design the kind of language developed by Quinlan (1982).

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 given structural language, but there are several problems with his approach that prevent it from working in the checkers and chess domains (see Porter & Kibler, 1985). The principal problem is that constraint back-propagation can only be applied to OR graphs, so it cannot handle the AND/OR graphs of these game-playing domains.

We have explored an alternative 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 *traps* 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. Each *trap* position generated is a conjunction of the two single *observable* facts like the original trap example given above and illustrated in figure3. 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 12 more general descriptions.

The first algorithm discovers simple relational terms that describe common relationships between squares. For example, in the 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(s2, s6) and se(s2, 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:

$$\begin{aligned} \forall square1\ square2\ square3 \quad & South2squares(square1\ square2) \\ \Leftrightarrow \quad & [se(square1\ square3) \wedge sw(square3\ square2)] \\ \vee & [sw(square1\ square3) \wedge se(square3\ square2)] \end{aligned}$$

The second term-creation algorithm is similar to GLAUBER (Langley et al., 1986) and identifies common internal disjunctions over the primitive structural features. The simple 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 king s10).

The first three items describe the red king, the following three, the white king. 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 king 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 king on s11 given below:

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

The algorithm identifies the set {s2, s3}, which is named *NorthCenterSide*. All of the features can be used to form the new terms. Using the *trap* instances, this algorithm created 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*.

6 Relation to Other Work

Wyl has much in common with other learning systems that pursue the "multiple representation strategy." Explanation-based learning systems (Mitchell, et al., 1986) employ a functional representation to describe the "domain theory." The examples are first presented in a structural language (mathematical expressions (Mitchell, et al., 1982; Shavlik, 1985), circuit primitives (Ellman, 1985; Mitchell, et al., 1985)), then explained via an envisionment stage using the domain theory to form "explanation trees" or "functional instances." These instances are generalized, usually deductively, by extracting sufficient conditions from the proof. Finally, the general concept is compiled into a structural description to make the knowledge "operational" (Mostow, 1982; Keller, 1983). Analogical learning systems (Winston, et al., 1983; Kedar-Cabelli, 1985) employ functional representations to determine when analogy is appropriate.

The work with analogy has particular relevance to the work presented here, because 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 it 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 most applicable representation for analogical reasoning about different items is one in which their underlying similarity is captured syntactically.

Another interesting comparison is to review the role examples play in these systems. In Wyl, examples are crucial to the capability to learn, as new concepts are being acquired. In some explanation-based learning systems (Minton, 1984; LEX2) examples play a different role. Here the systems already know the concepts (e.g., *loss*, *productive-operator*). The task is to translate these concepts into operational form. Examples play the role of focusing the attention of the program. For example, a system knowing the functional definition of a cup may proceed to translate its knowledge into operational form. The resulting structural description will be extremely disjunctive and include all *possible* physical realizations of a cup. A better approach is to use examples to guide this operationalization process.

7 Conclusion

In this paper, we have claimed that inductive learning can only succeed in representations in which the commonality of concept instances can be captured syntactically. We have justified this claim by illustrating the operation of Wyl, a program that learns concepts from examples in chess and

checkers. Wyl employs two representations (structural and functional) and performs induction in the functional space. This approach of using separate representations, each suited to different tasks, has significant advantages:

- Fewer examples are required to learn the concept.
- The bias built into the learning program is very simple (maximally-specific conjunctive generalization).
- The learning system starts with simple languages and requires less domain-specific and concept-specific engineering.

8 Acknowledgments

The authors wish to thank Bruce Porter for reading a draft of this paper. 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.

Ellman, T., "Generalizing Logic Circuit Designs by Analyzing Proofs of Correctness," in *Proceedings of IJCAI–85*, Los Angeles, CA 1985.

Gold, E. "Language identification in the limit." in *Information and Control,* Vol 16, pp447-474, 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.

Keller, R. M., "Learning by Re-expressing Concepts for Efficient Recognition," in *Proceedings of AAAI–83*, Washington, D.C., 1983.

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.

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

Minton, S. "Constraint-Based Generalization: Learning Game-Playing Plans from Single Examples," in *Proceedings of AAAI–84*, 1984.

Mostow, D. J. "Machine Transformation of Advice into a Heuristic Search Procedure," 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.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.M., Mahadevan, S., and Steinberg, L. I., "LEAP: A Learning Apprentice for VLSI Design," in *Proceedings of IJCAI–85*, Los Angeles, CA 1985.

Mitchell, T.M., Keller, R. M., and Kedar-Cabelli, S.T. "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.

Quinlan, J. R., "The Effects of Noise on Concept Learning," in *Machine Learning: An Artificial Intelligence Approach, Vol II* Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), Tiogga Press, Palo Alto, 1986.

Shavlik, J. W., "Learning About Momentum Conservation", in *Proceedings of IJCAI–85*, Los Angeles, CA 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.), Tiogga 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.