Machine Learning 35, 57–90 (1999)
© 1999 Kluwer Academic Publishers. Manufactured in The Netherlands.

Learning to Take Actions*

RONI KHARDON[†] roni@dcs.ed.ac.uk
Division of Informatics, University of Edinburgh, JCMB, King's Buildings, Edinburgh EH9 3JZ, Scotland

Editor: J. Shavlik

Abstract. We formalize a model for supervised learning of action strategies in dynamic stochastic domains and show that PAC-learning results on Occam algorithms hold in this model as well. We then identify a class of rule-based action strategies for which polynomial time learning is possible. The representation of strategies is a generalization of decision lists; strategies include rules with existentially quantified conditions, simple recursive predicates, and small internal state, but are syntactically restricted. We also study the learnability of hierarchically composed strategies where a subroutine already acquired can be used as a basic action in a higher level strategy. We prove some positive results in this setting, but also show that in some cases the hierarchical learning problem is computationally hard.

Keywords: learning to act, stochastic domains, supervised learning, rule based systems, hierarchical learning, NP-complete

1. Introduction

We formalize a model for supervised learning of action strategies in dynamic stochastic domains, and study the learnability of strategies represented by rule-based systems. In this model, the learner is given access to traces of behavior of another agent and using these traces it tries to reconstruct a strategy for behaving successfully in the same world. Following previous work on learning to reason (Khardon & Roth, 1995, 1997) the formalization utilizes two general ideas. First, one can gain insights by focusing on learning that is done for the purpose of performing well in a particular task. Second, when coupling learning with the task, the competence required of the agent can be defined relative to its learning interface. This allows for relaxed definitions describing plausible scenarios that admit efficient solutions.

Technically, our framework is based on the PAC model of learning from examples (Valiant, 1984) but applied to problems where the agent has to act in the world and achieve goals, similar to what is done in the study of planning (Allen, Hendler, & Tate, 1990). The formalization considers stochastic partially observable worlds as in reinforcement learning (Littman, Cassandra, & Kaelbling, 1995), where the state is described using relational

*An earlier version of this paper appears in the proceedings of the National Conference on Artificial Intelligence AAAI-1996.

[†]Most of this work was done while the author was at Harvard University and supported by ARO grant DAAL03-92-G-0115 and ONR grant N00014-95-1-0550.

information. We describe the dynamics in terms of "runs", where in each run a random initial state and goals are chosen and the agent has to act so as to achieve the goals. The examples are provided by using a fixed strategy to choose the actions on such random problems. After seeing some examples the learner has to find a strategy that performs as well as the strategy providing the examples. This generalizes previous work by Tadepalli and Natarajan (Tadepalli, 1991; Tadepalli & Natarajan, 1996) who studied similar problems of acting in deterministic worlds. Indeed our basic result shows that if a learning algorithm finds a strategy that can be described concisely, and such that it suggests the same actions that have been observed in the example traces, then it is guaranteed to be successful. Thus the well known convergence results for Occam algorithms in the PAC model (Blumer et al., 1987), that are known to hold in deterministic worlds (Tadepalli & Natarajan, 1996), hold also for stochastic partially observable worlds.

A large part of the paper is devoted to the study of rule-based action strategies and their learnability. The rules in the representation are of the form $C \rightarrow A$, where the condition C is an existentially quantified first order expression, and the right hand side A may be either a name of an action or a predicate. In particular, three collections of such rules are used to describe a strategy, that is, a program prescribing how an agent (either learner or example provider) chooses its actions. The main part of a strategy includes a priority list of rules whose right hand sides are real actions in the world. Whenever the agent needs to take an action it consults this list and chooses the action recommended by the first rule on this list whose condition holds at that moment. This is a generalization of propositional decision lists studied by Rivest (1987). The right-hand side of rules in the other two collections includes internal predicates private to the agent. One collection includes propositional state information, while the other includes recursively defined first order predicates. These internal predicates are used in the conditions of the priority list in the main part of the strategy. A concrete example describing such a rule-based system for the blocks world is given in Table 2 and discussed in more detail in Section 4.

We describe restrictions on such rule-based strategies so that efficient learnability can be achieved. In particular the left hand side of each rule is restricted to have only a constant number of predicates and existentially quantified variables, the internal state is restricted to a constant size, and the syntactic definition of the internal predicates is restricted so that they can be enumerated efficiently. We describe a learning algorithm that is efficient under these conditions, generalizing Rivest's (1987) algorithm for propositional decision lists, in a manner similar to Valiant's (1985) relational DNF expressions. The time required of the algorithm is polynomial in the number of predicates in the vocabulary of the agent, and in the number of objects that it encounters in the example traces. It grows exponentially, however, with the number of variables in the rules, and the size of the internal state machine; thus when these parameters are fixed to small constants we may expect efficient learnability.

We also study more complex action strategies that are composed hierarchically. Denote the class of strategies described above by simple strategies. Then a simple strategy for a particular task can be used as a subroutine in a hierarchically composed strategy for another task. This is in particular enabled by naming the subroutine and using this name as a basic action in the main part (the priority list) in the new strategy. Two types of control structures are considered. The first stipulates that control is given to a lower level subroutine for a single time step. In the next time step the conditions in the higher level are tested and if needed the subroutine is applied again for one time step. In this case hierarchical strategies are learnable under the same restrictions as above. This result is applicable to hierarchical teleo-reactive programs (Nilsson, 1994). In particular the main part of the strategy is equivalent to such a program and therefore such programs are learnable if the number of quantified variables is bounded. In the second control structure, once a subroutine is started it continues its execution until its "local goals" are achieved; only then control is returned to the strategy in the higher level. For this case we show that information on the hierarchy cannot be implicit. In particular we show that if the example traces are annotated so that it is known which subroutine is responsible for taking each action then efficient learning is possible. However, if this information is not given then the task becomes computationally hard (even if the subroutines are already known). Thus if learning of such strategies is to be performed then annotation must be provided.

To summarize, the contributions of the paper are twofold: First we introduce the supervised learning model and show that general convergence results can be achieved in this model. The restrictions imposed in the model are both its strength and its limitation. Indeed, in order to use the results, examples of behavior by a teacher[1] are needed and the expressiveness of the rule-based strategies is somewhat restricted. On the other hand, these restrictions make for problems that can be solved efficiently, and in addition we can prove that some algorithms will be both efficient and successful. Our results can also be seen as a partial theoretical explanation to similar empirical studies that have been done. For example, Sammut et al. (1992) study learning of action strategies for flying a plane. Their algorithm takes example traces, and produces situation-action pairs for a standard decision tree learning algorithm. Our general learning result suggests that the success of this approach can be quantified; if a small decision tree can be found then it will be useful for selecting actions in the future. The second contribution concerns the study of rule-based systems. The effort here is to find as expressive as possible classes of strategies that are learnable and that might be useful in other settings as well. We identify a subset of rule-based strategies that can be learned, and some limits to this learnability when strategies are hierarchically composed.

An empirical evaluation of these ideas has been performed (Khardon, 1997) where our learning algorithm is shown to be useful for learning action strategies in small planning domains that have been studied before. These experiments and their practical implications are briefly discussed in Section 7.

This work draws on several previous lines of research on learning, planning, rule-based systems, and relational representations, of which only some were mentioned above. An extensive discussion of the similarities and differences between our model and previous ones is given in Section 8.

The rest of the paper is organized as follows. The next two sections present the model and the result on Occam algorithms. The three sections that follow study rule-based strategies and their learnability. Discussions of the experimental system and related work appear in the next two sections. The final section concludes with a summary and directions for future work.

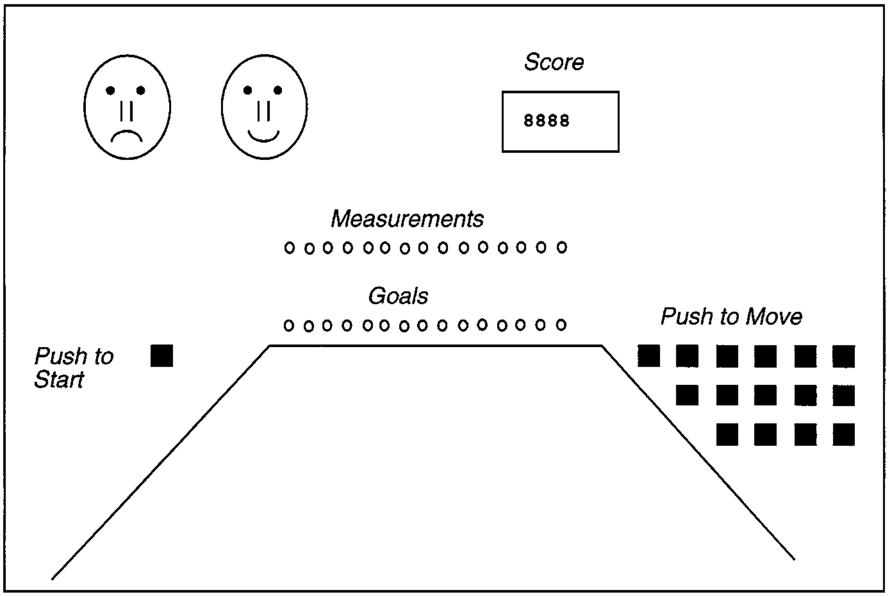

Figure 1. The game.

2. The model

We start with an example that illustrates the world model and the learning model. Suppose you go to visit a friend who knows you are interested in computer games. Your friend is very excited, and takes you immediately to the game room where you are shown a new game (see figure 1). On the right there are 50 or so buttons, with the heading "push to move". In the middle, you see two aligned sets of 50 or so light bulbs. One set with the heading "Measurements" and the other with the heading "Goals". The bulbs are flashing with red and green colors, and are sometimes turned off. On the left there is a button saying "push to start". Above these buttons and light bulbs there is a display with digits with the header "score" and two faces drawn, one smiling and the other frowning.

You are encouraged to try the game, which you do by pushing the start button, then you try to win the game by pressing the move buttons. Soon though, the frowning face is flashing, and the game stops; you have failed. You try a few more times but fail again.

Your friend, eager to show off, sits down, plays for a few rounds, and wins in all these rounds. You observe your friend carefully, recording which buttons are pushed in each situation. You then get a second chance with the game. Can you use the information recorded on your friend's moves so as to learn which buttons to push in order to win in the game?

Of course, we do not have enough information to answer the question. It may be that you are being fooled, and in fact some third person is controlling the light bulbs to make it appear that you are losing the game while your friend is winning. The question is whether some learning strategy is guaranteed to succeed under some reasonable assumptions; the paper formalizes this question and identifies such sufficient conditions.

Notice, that the scenario presented in the example corresponds to supervised learning. Namely, the learner gets labelled examples by observing a teacher who is playing the game. Similar questions can be raised for less supervised scenarios. For example, can one learn without the labelled examples, that is, just by trial and error? This corresponds to the reinforcement learning model. We only pursue the supervised setting here.

Intuitively, the world is modeled as a randomized state machine, in each step the agent takes an action, and the world changes its state where the transition probabilities depend on the current state and the action taken. The agent is trying to get to a state in which certain "goal" conditions hold. The basic scenario is similar to the example given above.

2.1. Acting

The interface of the agent to the world is composed of three components:

- The measurements of the learner are represented by a set of n literals, $X = x_1, x_2, \ldots, x_n$, each taking a value in $\{0, 1, *\}$. The value $*$ is intended to denote that the value of some variable is not known or has not been observed. No special semantics is given to partial assignments; instead we follow Valiant (1995) and Roth (1995) and simply consider the value $*$ as a third value an attribute can take.[2] The set $\{0, 1, *\}^n$ is the domain of these measurements.

  For structural domains, as in work by Haussler (1989), the input is composed of a list of objects, and values of predicates instantiated with these objects. Namely, there are n_1 objects, and n_2 predicate symbols, each of arity a or less, and the input is described by the $n = n_2 n_1^a$ instantiated predicates.
- The agent can be assigned a goal, from a previously fixed set of goals $\mathcal{G} = \{f \mid f : \{0, 1, *\}^n \to \{0, 1\}\}$. The intention here is that each f designates a certain goal, and that the learner can determine whether the goal f is satisfied in a certain state $x \in \{0, 1, *\}^n$. Namely, $f(x)$ can be easily evaluated.

  We have to fix a representation of goals to be presented to the agent. For simplicity we assume that $\mathcal{G}$ is the class of conjunctions over the literals $g_1, g_2, \ldots, g_n$, and its negations, where g_i represents the desired state of x_i. This is similar to conjunctive goals in STRIPS style planning problems. Every such goal f can be represented using an assignment $y \in \{0, 1, *\}^n$, where $f = \bigwedge g_i^{y_i}$, and $g_i^1 = g_i$, $g_i^0 = \bar{g}_i$, and $g_i^* = 1$. For example, if $y = (0 * 1)$ then $f = \bar{g}_1 \wedge g_3$.

  For relational problems we achieve the same effect by introducing a goal modality G that can take any relational predicate in the input as its argument. For example, if $p()$ is a relation of arity 2, and a, b are object names then $G(p(a, b))$ would denote the appropriate goal, in similarity with the literals g_i above.

  In view to representation of strategies to be discussed later we note that conditions involving both the required goal and the current inputs can be constructed. For example, in the propositional setting, the condition $\bar{g}_1 \wedge x_1$ expresses the fact that it is desired that x_1 be off and that it does not hold in the current input. Note that since $*$ is considered a third distinct value neither g_i not $\bar{g}_i$ are satisfied if $x_i = *$.
- The agent has at its disposal a set of actions $O = \{o_1, \ldots, o_n\}$. The symbol o_i denotes the name of the action. In the learning model, the agent is not given any information on

the effects of the actions, or the preconditions for their application. In particular, there is no hidden assumption that the effects of the actions are deterministic, or that they can be exactly specified. (The choice of n, the number of actions, to be the same as the number of literals, is simply intended to reduce the number of parameters used. One can think of n as a bound of this number.)

The protocol of acting in the world is modeled as an infinitely repeated game. At each round, an `instance`, (x, g), such that $x \in \{0, 1, *\}^n$ and $g \in \mathcal{G}$ is first chosen. Then the agent is given some time, say N steps (where N is some fixed polynomial in the complexity parameters), to achieve the goal g starting with state x. In order to do this the learner has to apply its actions, one at a time, until its measurements have a value y which satisfies g (i.e., $g(y) = 1$).

Intuitively, each action that is taken changes the state of the world, and at each time point the agent can take an action and then read the measurements after it. However, some of the actions may not be applicable in certain situations, so the state does not have to change when an action is taken. Furthermore, we would allow the state to change even when no action is taken. In order to simplify notation, we assume that one of the actions is a no-op action, and when the agent chooses not to take an action it simply chooses this action.

Definition 2.1 (*stationary strategy*). A *stationary strategy* $s : \{0, 1, *\}^n \times \mathcal{G} \to O$ is a mapping from instances into actions.

An agent is following a stationary strategy s, if on input x, and with goal g, the agent chooses the action $s(x, g)$. In general, however, a strategy may have an internal state:

Definition 2.2 (*strategy*). A *strategy* s is composed of a state machine (I, i_0, δ_s), and a mapping $s : \{0, 1, *\}^n \times \mathcal{G} \times I \to O$ from instances and states into actions.

In the state machine, I is the set of states, i_0 is the initial state, and $\delta_s : \{0, 1, *\}^n \times \mathcal{G} \times I \to I$ is the transition function.

An agent is following a strategy s if on a new instance it is initialized to state i_0, and if whenever it is in state $i \in I$, and on input (x, g), the agent chooses the action $s(x, g, i)$, and changes its state to $\delta_s(x, g, i)$. Note that the strategies we have defined are deterministic; this fact is used later in our arguments.

Definition 2.3 (*run*). A *run* of a strategy s on instance (x, g), is a sequence resulting from repeated applications of the strategy s,

$$R(s, x, g) = x, s(x, g, i^0), x^1, s(x^1, g, i^1), x^2, s(x^2, g, i^2), \ldots,$$

where $i^0 = i_0$, and for each $j \geq 1$, $i^j = \delta_s(x^{j-1}, g, i^{j-1})$. The run is continued until g has been achieved or N steps have passed.

Definition 2.4 (*successful run*). A run is *successful* if for some $i \leq N$, $g(x^i) = 1$.

Notice that, depending on the characteristics of the world, a run might be a fixed value or a random variable. In order to ensure that R behaves as a random variable we assume that the world behaves as a partially observable Markov decision process.[3] Namely, the world is composed of set of states, and a set of matrices describing the transition probabilities of moving from one state to another depending on the actions of the agent. The agent does not observe the actual state of the system but only some partial measurement over the state. That is, many states may be mapped to the same measurement.

Definition 2.5 (*world*). The world W is modeled as a partially observable Markov decision process whose transitions are effected by the actions of the agent.

It should be noted that we do not make any assumptions on the size or structure of W. Furthermore, we do not expect an agent to have complete knowledge of W. Instead, we would want an agent to have a strategy that copes with its task when interacting with W.

We next model the start button in the scenario described above. We assume that at the beginning of a random run, a state of the Markov chain is randomly chosen according to some fixed probability distribution D. This distribution induces a probability distribution D over the measurements $\{0, 1, *\}^n \times \mathcal{G}$ that the learner observes at a start of a run.

Definition 2.6 (*random run*). A *random run* of a strategy s with respect to a world W, and probability distribution D, denoted $R(s, D)$, is a run $R(s, x, g)$ where (x, g) are induced by a random draw of D, the actions are chosen according to s, and the successor states are chosen according to the transition matrix of W.

Since W is thought of as fixed, we suppress the parameter W in the notation for R. The above definition ensures that a random run is indeed a random variable. Notice that if the strategy is deterministic then the distribution of R is determined by D and W.

The start button in our model assumes that some form of a reset operation is given to the agent. This may limit the application of the results in some situations. The assumption is however not too strong, and may be thought of as saying that the choice of new problems generated for the agent is invariant of other changes in the world; that is, in some sense the source of problems is stationary. Similar assumptions have been made in several works in reinforcement learning (Sutton, 1990; Fiechter, 1994; Littman, Cassandra, & Kaelbling, 1995).

The quality of a strategy is the probability that a random run is successful. Formally,

Definition 2.7 (*quality of a strategy*). The quality $Q(s, D)$ of a strategy s, with respect to a world W, and probability distribution D, is

$$Q(s, D) = \texttt{Prob}[R(s, D) \text{ is successful}]$$

where the probability is taken over D, and the random variable R.

2.2. Learning

We assume that a teacher has some strategy t, according to which it chooses its actions. The example oracle returns a random run of the teacher's strategy.

Definition 2.8. The oracle `example`(t) when accessed, returns a random sample of $R(t, D)$.

A learning algorithm will get access to the oracle `example` and will try to find a strategy which is almost as good as the teacher's strategy. Let S be a class of strategies; assume some standard representation for strategies in S, and for $s \in S$ let $|s|$ be the size of the representation of s.

Definition 2.9 (*learning*). An algorithm A is a *Learn to Act* algorithm, with respect to a class of strategies S, class of worlds $\mathcal{W}$, and class of distributions $\mathcal{D}$, if there exists a polynomial $p()$, such that on input $0 < \epsilon, \delta < 1$, for all $t \in S$, for all $W \in \mathcal{W}$, for all $D \in \mathcal{D}$, and when given access to `example`(t), the algorithm A runs in time $p(n, |t|, 1/\epsilon, 1/\delta)$, where n is the number of predicates measured in each example, and with probability at least $1 - \delta$, A outputs a strategy s such that $Q(t, D) - Q(s, D) \leq \epsilon$.

3. Learning action strategies

In this section we present a general learning result. Similar to results in the PAC model (Blumer et al., 1987) we show that an Occam algorithm that finds a concise action strategy which is consistent with all the examples seen, is a learning algorithm. This result is later used to prove the learnability of rule-based systems.

The main idea is that an action strategy that is very different from the teacher's strategy will be detected as different by a large enough random sample. In the PAC model of concept learning examples are randomly and independently sampled and do not depend on the learner. Thus, if a hypothesis is consistent with a large sample of examples, it is expected to behave well on the same distribution when tested. In contrast, when learning to act, the distribution of states visited after taking the first action depends on this action. Namely, the distribution of runs on which a strategy s is measured depends on s. Therefore, the above argument is not sufficient here, and one has to show that the quality of the strategy, measured by this new distribution, is also good. As the following theorem shows, Occam algorithms are successful since most of the good runs of the teacher are also covered by a consistent strategy.

Recall that the strategies we defined are deterministic. Therefore, the randomness in a run depends only on the world as expressed through D and the Markovian process. As a result we can talk about a strategy being consistent with a run. We say that a strategy is consistent with a run $R = x, o_{i_1}, x^1, o_{i_2}, x^2, o_{i_3}, \ldots, o_{i_l}x^l$ if for all j, the action chosen by the strategy in step j, given the history on the first $j-1$ steps (which determine the internal state of the strategy), is equal to o_{i_j}.

Theorem 3.1. *Let H be a class of strategies, and let L be an algorithm such that for any $t \in H$, and on any set of runs $\{R(t, D)\}$, L finds a strategy $h \in H$ which is consistent with all the runs. Then L is a* learn to act *algorithm for H when given $m = \frac{1}{\epsilon}\ln(\frac{|H|}{\delta})$ independent example runs.*

Proof: The examples presented to the learner are independent samples of the random variable $R(t, D)$. We would next consider the set of runs produced by a strategy; if s is consistent with a run R then we say that R is in s, and otherwise R is not in s.

Denote by D^t (D^s, respectively) the distribution on runs induced by t (s, respectively). Then, since s, t are deterministic, we get that for a run R which can be produced by both s and t, $D^t(R) = D^s(R)$.

First observe that a strategy s satisfying $D^t(R \text{ in } s) > 1 - \epsilon$ has good quality, since

$$\begin{aligned} Q(s, D) &= D^s(R \text{ in } s \text{ and } R \text{ successful}) \\ &\geq D^s(R \text{ in } s \text{ and } R \text{ in } t \text{ and } R \text{ successful}) \\ &= D^t(R \text{ in } s \text{ and } R \text{ successful}) \\ &> 1 - \epsilon - (1 - Q(t, D)) \\ &= Q(t, D) - \epsilon. \end{aligned}$$

Now, the theorem follows since the probability that any strategy s not satisfying $D^t(R \text{ in } s) > 1 - \epsilon$ is consistent with m examples is very small. Since the runs are independent, the probability that s agrees with all of them is at most $(1 - \epsilon)^m < e^{-\epsilon m} = \delta/|H|$. The probability that this happens for any strategy in H is at most δ. □

The theorem assumes a fixed size hypothesis class H. It is straightforward to generalize this theorem to cases where the hypothesis size depends on the size of the strategy being learned, in line with previous results on Occam algorithms (Blumer et al., 1987; Kearns & Vazirani, 1994). The above proof also ensures that the learner is almost as bad as the teacher. A similar result without this unwanted guarantee can be derived by using only successful runs in the sample. Namely, the learning algorithm will take a sample of $m = \frac{1}{\epsilon} \log(\frac{|H|}{\delta})$ independent successful example runs. (The expected number of calls to `example`(t) is $O(1/Q(t, D))$ times the above sample size.) Generalizations of this result will be interesting. In particular the restriction to deterministic strategies enabled the above proof; it remains to be seen whether some version of the result holds for randomized strategies, or when there is "noise" in the examples.

Using the above theorem we can immediately conclude that strategies representable as macro tables (Korf, 1985) and intersection-closed strategies, for which Occam algorithms exist (Tadepalli, 1991; Tadepalli & Natarajan, 1996), are learnable in our model.

4. Representation of strategies

We use a rule-based representation of strategies motivated by work on production systems. The representation exemplifies that a symbolic relational representation including declarative information can be an inherent part of a reactive agent.[4] We start by discussing some general features of production systems and motivate the types of restrictions that are employed. Then, in order to facilitate the presentation of the results, we describe a Production Rule System (PRS) for the blocks world which is in the class of strategies that can be learned. Finally, formal definitions are given.

4.1. Production rule systems

Production rule systems (Anderson, 1983; Klahr, Langley, & Neches, 1986; Laird, Rosenbloom, & Newell, 1986; Newell, 1990) are composed of a collection of condition-action rules $C \to A$, where C is usually a conjunction (over some relevant predicates), and A is used to denote an action. Actions in PRS denote either a real actuator of the agent, or a predicate which is "made true" if the rule is fired. PRS are simply a way to describe programs with a special kind of control mechanism. An important part of this mechanism is played by the *working memory* and *goal structures*.[5] The working memory captures the "current state" view of the system, and similarly the goal structures capture the current goals, and may also be thought of as a part of the working memory. Initially, the input is put into the working memory, and the PRS then works in iterations. In each iteration, the condition C of every rule is evaluated, to get a list of rules which "match" the current state. Out of these rules, one is selected, by the "resolution mechanism", and its action A is executed. That is, either the actuator is operated, or the predicate mentioned as A is added to the working memory. The above cycle is repeated forever or until the goal is achieved. Note that, while the production rules look similar to logical formulas, a production is not a logical statement but rather a procedural description of the system.

Various resolution mechanisms have been discussed in the literature. Some choose the rule according its its current "strength" which is dynamically updated (Anderson, 1983). Others use other rules to resolve between competing rules (Newell, 1990; Rosenbloom, Laird, & Newell, 1993) encoding some sort of priority between the rules or their instantiations.

Notice that PRS can provide a substrate both for procedural representations, that are the natural interpretation of the architecture, and for a symbolic declarative representation which can be used as in planning. In fact, PRS have been mostly used as a symbolic representation, and learning mechanisms similar to explanation based learning (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986) have been studied in this framework (Anderson, 1983; Newell, 1990).

A similar view on knowledge representation and dynamics of reasoning evolves from Valiant's (1994) study of neural circuits. There, a procedural description of one item in terms of others is enforced by the structure of the system. Furthermore, computational considerations suggest the use of working memory, called an imagery device, which works together with the neural circuit in a manner similar to that of productions. This view was further expanded into a framework for studying cognitive systems (Valiant, 1995, 1996). An integral part of this view, however, is that most rules are inductively acquired, rather than hand-crafted or compiled as in PRS. The claim is that this fact is a crucial one, and is the reason for the competence of the set of rules.

4.2. Restricted PRS

We next motivate some of the restrictions that are taken in our representation. First recall the operation cycle of PRS. In every iteration all rules are compared to the current situation, and out of the rules that match the situation one is chosen to be executed. This choice is made by a resolution mechanism, and several possibilities for such mechanisms have been studied. For our action strategies, we use a simple priority encoding as a resolution mechanism. Namely, the rules can be ordered in a *priority list* and the first rule on this list that matches the current state is the one chosen. Priorities have been previously used for resolution (Newell & Simon, 1972), and they also resemble the end product of the situation in Soar (Newell, 1990) where chunking adds control rules for selection between competing rules, and therefore some priority between rules is enforced.

Notice that enforcing priority between rules is still not sufficient for an exact semantics. Suppose we have an ordered list of rules with object variables; if the first rule applies we do not need to test the other rules. The question is what to choose if more than one binding for the same rule applies. There are several possibilities here; we will assume that *the lexicographic ordering is used to resolve between bindings*. Namely, the lexicographically first binding that matches the condition is the one to choose the action.

Secondly, we must ensure that once we have a PRS, it can compute its output efficiently. PRS have mainly been used in structural domains. In this case, literals are predicates with free variables (e.g., $move(x, y)$). When the condition is specified using such predicates, and the "current state" includes a list of objects and some relations that hold between them, a new computational problem arises. Namely, one has to *bind* the object variables in the condition to the actual objects in the current state. This problem is computationally hard in general and typically either the number of objects or the number of variables is restricted (Haussler, 1989). In order to avoid this problem we *restrict the number of variables that appear in the condition to be bounded by some fixed constant α*. In this case one can test all possible bindings for n objects in time $O(n^{\alpha})$. While each rule is restricted in this way, the fact that rules are used in a priority ordering makes for conditions that are effectively more complex (since the condition of a rule in the PRS is effectively conjoined with the negation of the conditions of previous rules in the priority list).

The third issue that has to be discussed is the use of working memory. Notice that when using PRS, allowing for extra internal "working memory" variables can reduce the size of the strategy considerably. For example, consider a propositional domain with propositions $x_1, \ldots, x_n$, and consider the strategy expressed by $(x_1 \vee x_2 \vee x_3) \wedge (x_5 \vee x_6 \vee x_7) \rightarrow o_1$. If we are not allowed to use intermediate variables, then we must multiply out the expression to get in the general case an exponential number of condition-action rules. On the other hand, using the internal predicates x_4, x_8 such that the rules $x_1 \rightarrow x_4, x_2 \rightarrow x_4, x_3 \rightarrow x_4$, $x_5 \rightarrow x_8, x_6 \rightarrow x_8, x_7 \rightarrow x_8$ hold, we can use the PRS: $x_1 \rightarrow x_4; x_2 \rightarrow x_4; x_3 \rightarrow x_4$; $x_5 \rightarrow x_8; x_6 \rightarrow x_8; x_7 \rightarrow x_8; x_4 x_8 \rightarrow o_1$, which has linear size. As we demonstrate below, in structural domains the effect of internal predicates is even stronger, allowing for strategies which would otherwise not be expressible in the language.

Furthermore, notice that working memory is internal to the strategy and will therefore be hidden from a learner observing the actions of the strategy. As a result the larger the internal memory the harder learning will be. We therefore *restrict the amount and type of internal working memory* of the strategies considered. Similar restrictions are motivated on cognitive grounds in VanLehn's (1987) "show-work" mode of learning. We will use two types of internal memory. One type includes recursive predicates which we call *support predicates*. The other type includes propositional variables which constitute a small state machine. We call these internal *state variables*.

4.3. A PRS for blocks world

In order to illustrate the style of PRS considered, we present a PRS strategy for the Blocks World. In this problem, there is a set of cubic blocks $a, b, c, \ldots,$ placed on a table. The table can fit all the blocks, which are all of the same size and can fit exactly one on top of another to form stacks of blocks. The task involves moving the blocks from an arbitrary initial configuration into a goal state which satisfies some arbitrary (though legal) conditions.

A situation is described by listing the names of blocks, and the relations that hold for them. The input relations we consider are: $\mathit{clear}(x)$ which denotes that nothing is placed above block x, and $\mathit{on}(x, y)$ which denotes that block x is on block y. We assume that the goal situation is described in a similar manner using the modality $G()$. For example $G(\mathit{on}(a, b)) \wedge G(\mathit{on}(b, c))$ could be our goal. The only action available is $\mathit{move}(x, y)$ which moves object x to be on y given that both were *clear* beforehand.

The problem of finding the shortest solution for blocks world instances has been shown to be NP-complete by Gupta and Nau (1991). However, they have also shown that there is a simple algorithm that produces at most twice the number of steps that is needed. Essentially the idea is that if a block is above another block, which is part of the goal but is not yet in its goal place, then it has to be moved. If we move such blocks to the table, and otherwise make constructive moves towards the goal, then we will make at most twice the number of steps that are needed. This heuristic has been recently shown to be very close to optimal (Slaney & Thiebaux, 1996). We present a PRS which implements this algorithm (which assumes for simplicity that the target stacks of blocks start on the table).

Our production rule systems have three parts. The first part computes the *support predicates* of the system. The second part consists of a priority list of condition action rules which we will refer to as the *main part of the PRS*. The third part includes rules for updating the *internal state*. The control structure of the PRS, described in Table 1, is accordingly composed of three steps. The PRS for the blocks world is described in Table 2. We explain the representation and computation using this example.

- First, the support predicates are computed by repeated forward application of the appropriate condition action rules, until no more changes occur. The PRS for blocks world computes the predicates $\mathit{inplace}(y)$, and $\mathit{above}(x, y)$. These have the intuitive meaning; namely $\mathit{inplace}(x)$ if x is already in its goal situation, and $\mathit{above}(x, y)$ if x is in the stack of blocks which is above y. The restrictions described later guarantee that the support predicates are monotone in the new predicates. Hence each rule application may add (but

Table 1. The control structure for PRS.

```
Repeat:

    Compute the support predicates for the current input.
    Choose an action using the main part of the PRS.
    Update the internal state.

Until the goal is achieved.
```

Table 2. A PRS for blocks world.

The support predicates

1. $\mathit{inplace}(\text{Table})$
2. $\mathit{on}(x, y) \wedge G(\mathit{on}(x, y)) \wedge \mathit{inplace}(y) \rightarrow \mathit{inplace}(x)$
3. $\mathit{on}(x, y) \rightarrow \mathit{above}(x, y)$
4. $\mathit{on}(x, y) \wedge \mathit{above}(y, z) \rightarrow \mathit{above}(x, z)$

The main part of the PRS

1. $\mathit{clear}(x) \wedge \mathit{clear}(y) \wedge G(\mathit{on}(x, y)) \wedge \mathit{inplace}(y) \rightarrow \mathit{move}(x, y)$
2. $\mathit{inplace}(y) \wedge G(\mathit{on}(x, y)) \wedge \overline{\mathit{on}(x, y)} \wedge \mathit{above}(z, x) \wedge \mathit{clear}(z) \wedge \overline{\mathit{sad}} \rightarrow \mathit{move}(z, \text{Table})$
3. $\mathit{inplace}(y) \wedge G(\mathit{on}(x, y)) \wedge \overline{\mathit{on}(x, y)} \wedge \mathit{above}(z, y) \wedge \mathit{clear}(z) \rightarrow \mathit{move}(z, T)$
4. $\mathit{inplace}(y) \wedge G(\mathit{on}(x, y)) \wedge \overline{\mathit{on}(x, y)} \wedge \mathit{above}(z, x) \wedge \mathit{clear}(z) \rightarrow \mathit{move}(z, T)$

Internal state

1. $\mathit{inplace}(y) \wedge G(\mathit{on}(x, y)) \wedge \overline{\mathit{on}(x, y)} \wedge \mathit{above}(z, x) \wedge \mathit{clear}(z) \oplus \mathit{sad} \rightarrow \mathit{sad}$

not remove) elements from the extension of such a predicate, and "bottom up" forward chaining is sufficient.

- Then, the main part of the PRS chooses the action. The main part of the PRS is considered as a priority list. Namely, the first rule that matches the situation is the one that chooses the action. It is assumed that if the condition holds for more than one binding of the rule to the situation, then the lexicographic ordering is used to choose between them. For the blocks world the main part of the list contains four rules, and they are used to choose which block is moved, implementing the algorithm described above.
- Finally, the internal state is updated after choosing the action. The form of the transition rules is defined syntactically, and we use $\oplus$ to describe the XOR operation. For example we can have the rule $\mathit{on}(x, y)\mathit{on}(y, z) \oplus s_1 \rightarrow s_1$, which indicates that the value of s_1 is flipped if there is a stack of three blocks in the current state. In our example the internal state *sad* is used but is not needed in order to solve the problem.[6] However, it may be useful if a special situation is to be identified.

Notice that recursive predicates enhance the computing power of PRS considerably. Namely, it is not simply enhancing the length of the rules by compressing the value of a k-conjunction into a single literal. Rather, this gives the PRS certain computing power which it did not have otherwise. For example the predicate *above* cannot be described by a simple PRS. Evaluating this predicate may require an arbitrary number of steps, depending on the height of the stack of blocks.

4.4. Definitions

While the PRS we consider have all the properties described above, we restrict these representations syntactically. Let k, c, a be fixed constants. In the following we will assume that all predicates and action names are of arity at most a. We use k to bound the number of predicates in a condition, and c to bound the number of internal states. We refer to the predicates that appear in the interface as *base predicates*. In addition to the base predicates, and the set of actions O, the language includes a goal modality G that can take any base predicate as its argument. A PRS strategy is composed of three parts.

- Let $\mathcal{P}$ be a set of predicates. The main part of the PRS is a list of rules. Each rule is of the form $C \to A$ where $A \in O$, and C is a conjunction of up to k predicates each of which is in $\mathcal{P}$. The parameters of predicates in C and A are instantiated with variables. Each predicate p in C may be either positive or negated. If the predicate is a base predicate, it may be combined with a goal modality in one of the following six forms: $\{p, \bar{p}, G(p), G(\bar{p}), \overline{G(p)}, \overline{G(\bar{p})}\}$. The rules are implicitly taken to be existentially quantified.

  The set $\mathcal{P}$ includes the basic set of predicates; when support predicates, or internal states are allowed it also includes these.

  The semantics for the main part of the PRS is as explained above: in every situation the first rule on the list for which the condition C holds determines the action. If more than one binding matches, the lexicographically first binding that matches is the one chosen.
- The support predicates include two sets of predicates. The first set defines non-recursive predicates. Each non-recursive predicate is defined using a single rule of the form $C \to A$ where C is a k-conjunction of base predicates, and A is a unique predicate name (a positive literal) that does not appear in the base predicates, or on the right hand side of any other rule in the system.

  The second set defines recursive predicates. Each recursive predicate is defined using two rules each of the form $C \to A$. The right hand side in both rules is identical, and includes a unique predicate name (a positive literal) that does not appear in the base predicates, or on the right hand side of any other rule in the system. The first rule is a base-case rule with exactly the same structure as a non-recursive predicate. In the second rule exactly one of the predicates in C is the same one as on the right hand side (but with different variables attached to it). This predicate appears with positive polarity. All other predicates in C in both rules are base predicates.

  In the example for the blocks world, rule number 3 (in the support predicates part) is the base-case rule of the predicate *above*, and rule number 4 is the recursive rule. Since the rules are monotone in the recursive predicate, repeated application of the rules results in a unique extension for the support predicates. This is the semantics attached to these predicates.
- The internal state includes c variables each associated with a single rule of the form $C_1 \wedge C_2 \oplus s_i \to s_i$, where C_1 is a conjunction over $\{s_1, \ldots, s_c, \bar{s}_1, \ldots, \bar{s}_c\}$, and C_2 is a k-conjunction over the base predicates and invented predicates (polarity and goal modality are restricted as above).

  The semantics here postulates that all variables are initialized to 0 in the beginning of a run, and that s_i will change its state wherever the condition $C_1 \wedge C_2$ matches for at least one binding.

Several subsets of PRS that are discussed in the next section are defined below. Hierarchical strategies will be defined in Section 6.

- The class of k-PPRS (for priority PRS) is defined such that the base predicates and actions are propositional, and support predicates and internal state are not used.
- The class of k-SD-PPRS (SD stands for structural domains) is defined such that the base predicates and actions are relational (of arity at most a), and support predicates and internal state are not used.
- The class of k-IP-SD-PPRS (IP stands for internal predicates) is defined such that the base predicates and actions are relational (of arity at most a). Support predicates are allowed but internal state is not used.
- The class of k-IS-IP-SD-PPRS (IS stands for internal state) is defined such that the base predicates and actions are relational (of arity at most a). Both support predicates and internal state are allowed.

5. Learning PRS action strategies

In this section we show that PRS, as described above, are learnable. We start with simple propositional strategies, and gradually increase the expressiveness of the strategies considered.

For simplicity, the definitions given above assumed that all literals have their values in $\{0, 1\}$. As described by Valiant (1995) expressions over $\{0, 1, *\}$ can be built by considering a basic literal for each possible set of values. Namely, instead of having two literals $x_i, \bar{x}_i$ we have seven possible literals $(x_i = 0)$, $(x_i = 1)$, $(x_i = *)$, $(x_i \in \{0, *\})$, $(x_i \in \{0, 1\})$, $(x_i \in \{*, 1\})$, $(x_i \in \{0, 1, *\})$. These literals can then be combined by logical operators in the usual manner. The analysis of learning is presented for the $\{0, 1\}$ case but easily generalizes to the general case of $\{0, 1, *\}$.

5.1. Learning propositional PRS

Recall that by Theorem 3.1 it is sufficient to find a strategy consistent with the example runs in order to learn to act. Since k-PPRS are stationary we can partition each run into situation-action pairs and find a k-PPRS consistent with the collection of these pairs.

The class of k-PPRS strategies is essentially Rivest's (1987) k-decision list generalized from a binary valued concept to a multi-valued concept. Rivest showed that a greedy algorithm succeeds in finding a consistent k-decision list. The same argument (given in the next two lemmas) holds in the multi-valued case. We include it here so as to facilitate the discussion that follows.

Lemma 5.1. *The number of k-PPRS action strategies is bounded by* $M = [n\binom{n}{k}6^k]!$.

Proof: There are $\binom{n}{k}$ subsets of variables and each variable can appear in one of six forms including the goal modality to form a conjunction. Each conjunction can be combined with up to n different actions. Every k-PPRS corresponds to an ordering of the set of all possible rules (where the rules which are never used, since all possible values are matched before, can be omitted). $\square$

Lemma 5.2. *There is a polynomial time algorithm that finds a consistent k-PPRS strategy for any set of runs taken from $R(D, t)$, where t is a k-PPRS.*

Proof: Note that t is a consistent action strategy, and as argued above it is sufficient to consider situation-action pairs. Suppose we found a rule which explains some subset of the situation-action pairs. Namely, it recommends the correct actions for this subset but does not recommend a wrong action for other pairs. Then, if we add t after this rule we get a consistent strategy. Therefore, explaining some examples never hurts. Furthermore, there is always a consistent rule which explains at least one example, since by the construction, at least one of the rules in t does. This implies that a greedy algorithm, arbitrarily adding one consistent rule at a time, succeeds. $\square$

Therefore by Theorem 3.1 we get:

Corollary 5.3. *There is a* learn to act *algorithm with respect to the class of k-PPRS action strategies.*

5.2. Learning PRS in structural domains

We now show that PRS for structural domains, with support predicates, and with internal state are also learnable. To simplify notation, we assume that n bounds the number of objects seen in the examples (in addition to the number of predicates, and the number of action names).

We start by considering k-SD-PPRS. A similar result was obtained by Valiant (1985) where the learnability of relational DNF expressions is shown.

Lemma 5.4. *The number of k-SD-PPRS action strategies is bounded by $M = m_0!$, where $m_0 = n(n+1)^k 6^k (a(k+1))^{a(k+1)}$.*

Proof: The quantity m_0 counts the number of rules that can be constructed. There are n possible actions, and k predicates to choose for the condition each having $(n+1)$ possible predicate names (including the no predicate option) and one of 6 forms (including the goal modality). The rest bounds the number of ways we can choose names for the variables (at most $a(k+1)$ names are needed including the variables in the action). To form a PRS we order the set of all rules (the ones in the end do not get used but we do not care). $\square$

We have already discussed our assumption on the existence of a teacher. One important aspect of our strategies is that they do not include object constants. Therefore the class of strategies implicitly induces the assumption that all objects are the same unless the set of predicates indicates otherwise. This forces the use of general rules, and reduces the complexity of the algorithms (since they do not have to consider constants in the construction of rules). In fact if the number of object variables is smaller than the number of objects in the examples, the size of the class of strategies is smaller than the size of the corresponding propositional class of strategies where one fixes the number of objects and instantiates all predicates over these objects. Thus while the structure of strategies is more complex, the sample complexity of the relational learning problem can be smaller than in the corresponding propositional case.

Lemma 5.5. *There is a polynomial time algorithm that finds a consistent k-SD-PPRS strategy for any set of runs taken from $R(D, t)$, where t is a k-SD-PPRS.*

Proof: As before, since the strategies are stationary it is sufficient to consider situation-action pairs, which we refer to as examples. Also, the same high level argument holds: explaining a subset of the examples does not hurt since, as t is consistent, there is at least one consistent rule. It remains to show that given a set of examples we can identify a consistent rule.

Fix a rule $C \rightarrow A$, and a set of examples. All the examples for which at least one binding for C is satisfied must be explained by the rule. That is, if for some example, satisfying the condition C, no binding produces the right action, then we can reject this rule. If for some example more than one binding agrees with C, then if the lexicographically first binding that matches does not produce the right action then we can reject this rule. If the rule is not rejected by any example then it is consistent. The claim follows since we can enumerate the set of rules. □

As before Theorem 3.1 implies:

Corollary 5.6. *There is a* learn to act *algorithm with respect to the class of k-SD-PPRS action strategies.*

5.2.1. Support predicates. Consider the PRS for the blocks world given in Section 4. So far we have seen that one can learn the main part of the strategy, given that a convenient set of predicates is given. In our example, the predicates *inplace* and *above* enabled the usage of a simple PRS for the actions. We next consider the class k-IP-SD-PPRS and show that there is a learning algorithm that can invent such predicates and use them during execution time.

Lemma 5.7. *The number of k-IP-SD-PPRS action strategies is bounded by $M = m_1!$, where $m_0 = n(n+1)^k 6^k (a(k+1))^{a(k+1)}$, and $m_1 = n m_0^{2k} 6^k (a(k+1))^{a(k+1)}$.*

Proof: The same structural restrictions for the PRS itself hold. The only difference is that we have more predicates. The working memory itself does not increase this size, since the predicates in this part are fixed. (In fact we can include all of them; removing the predicates that are not used should only be done for efficiency.)

We next bound the number of predicates. The number of non-recursive predicates that are allowed is bounded by the number of k-conjunctions (combined with the variables for the new predicate) $\leq \beta_1 = (n+1)^k 6^k (a(k+1))^{a(k+1)}$.

The number of recursive predicates is bounded by

$$\beta_2 = \left[(n+1)^k 6^k (a(k+1))^{a(k+1)}\right] \cdot \left[(n+1)^{(k-1)} 6^{k-1} (a(k+1))^{a(k+1)}\right],$$

and the total number of predicates including the original set and the two new sets is bounded by $\beta_1 + \beta_2 + n < m_0^2$. □

Lemma 5.8. *There is a polynomial time algorithm that finds a consistent k-IP-SD-PPRS strategy for any set of runs taken from $R(D, t)$, where t is a k-IP-SD-PPRS.*

Proof: Notice that the strategies considered are still stationary. We use the greedy algorithm as before altering its input by a preprocessing step. In this step, for each x^i we compute the values of *all* possible invented predicates that agree with the syntactic restriction. This computation is clearly possible for the non-recursive predicates. For the recursive predicates, one can use an iterative procedure to compute these values. First apply the base case on all possible bindings. Then in each iteration apply the recursive rule until no more changes occur. This is correct since the recursive rule is monotone in the new predicate. Namely, addition of new positive instances of the predicate cannot negate previously found positive instances.

After this preprocessing step we run the greedy algorithm for PRS using the extended set of literals. We are guaranteed that a consistent PRS exists, and that the greedy algorithm will find one. Once such a PRS has been found we simply include all the invented predicates used in this PRS. □

We therefore conclude:

Corollary 5.9. *There is a* learn to act *algorithm with respect to the class of k-IP-SD-PPRS action strategies.*

While performed for the purpose of acting, the above algorithm solves a simple problem of predicate invention. The problem is simple since our syntactic restrictions bound the total number of possible predicates by a polynomial, and learning is achieved by enumeration. The learnability of these predicates is therefore not interesting in its own right but rather as an add on to the learnability of strategies. What it shows is that one can tolerate a small amount of hidden information. Apart from enhancing the expressiveness of strategies, this scheme is useful if such invented predicates can help in transferring knowledge from one learning scenario to the next.

Recursive predicates and predicate invention have been studied before in Explanation Based Learning (EBL) and Inductive Logic Programming (ILP) (see Section 8). Muggleton (1994) describes a technique where predicates are invented where they are useful for the structure of a proof (for example by adding the predicate N such that $A \rightarrow N$, $B \rightarrow N$, and $N\alpha \rightarrow D$, when a both $A\alpha \rightarrow D$, and $B\alpha \rightarrow D$ exist in the system). Zelle and Mooney (1994) use predicate invention in the context of a covering method for ILP; if the initial covering technique fails, they invent a new predicate so as to explain away the examples on which the clause constructed so far predicts incorrectly. Shavlik (1990) describes an EBL technique of learning recursive predicates that utilizes the tree structure of an explanation in identifying recursive constructs. In contrast with Muggleton (1994) and Shavlik (1990) our method is empirical and does not use an explanation or proof structure. It also differs from the empirical method of Zelle and Mooney (1994) in the use of the syntactic restrictions.

While the scope of our predicates is more limited we are able to guarantee correctness and efficiency.

5.2.2. Internal state. We further extend the class of strategies learned, allowing a strategy to include a constant number of internal state variables. The learnability of state machines has been studied under a variety of conditions, e.g., in (Angluin, 1987; Kaelbling, 1993). As for the invented predicates our result uses a simple enumeration technique and shows that a small amount of hidden information can be tolerated; it is of interest mainly as an add on to the learnability of strategies. PRS with internal state machines may appear hard to learn at the outset. An internal state machine can make the impression that the output of the teacher is random, as in hidden variable problems discussed by Kearns and Schapire (1994). However, since the number of states is small, we can get a learning result for this class.

Lemma 5.10. *The number of k-IS-IP-SD-PPRS action strategies is bounded by* $M = m_2 \cdot m_3!$, *where* $m_0 = n(n+1)^k 6^k (a(k+1))^{a(k+1)}$, $m_2 = m_0^{2kc} 6^{kc} (a(k+1))^{ca(k+1)} 3^{c^2}$, *and* $m_3 = n(m_0^2 + c)^k 6^k (a(k+1))^{a(k+1)}$.

Proof: Again the same arguments apply. We only have to bound the number of state machines, that can be defined in this way. For each state variable we have to choose a conjunction of the state variables (3^c possibilities) and, a k-conjunction over the input and support predicates, which is bounded by $(m_0^2)^k 6^k (a(k+1))^{a(k+1)}$. So the number of state machines that can be defined in this way is at most $m_2 = m_0^{2kc} 6^{kc} (a(k+1))^{ca(k+1)} 3^{c^2}$. The number of predicates available to the main part of the PRS is $m_0^2 + c$, and the bound is derived as before. □

Lemma 5.11. *There is a polynomial time algorithm that finds a consistent k-IS-IP-SD-PPRS strategy for any set of runs taken from* $R(D, t)$, *where* t *is a k-IS-IP-SD-PPRS.*

Proof: To find a consistent PRS we can enumerate the set of state machines, and for each machine use the greedy algorithm to find a consistent strategy as before. While the strategies considered are not stationary, they become stationary if the correct state variables are added as part of the input. This allows the algorithm to find a consistent strategy even though it considers only situation-action pairs. □

We therefore get that the class is learnable. For reference, we describe a high level description of the learning algorithm in Table 3.

Corollary 5.12. *There is a* learn to act *algorithm with respect to the class of k-IS-IP-SD-PPRS action strategies.*

6. Hierarchical strategies

The expressiveness of the strategies considered so far, and in particular the invented predicates and internal state is somewhat restricted. Some of the limitations on the expressiveness of strategies can be overcome using a hierarchical structure.

Table 3. The algorithm *learn-PRS*.

```
Initialize the strategy to the empty list.

Do for each possible state machine allowed by the restrictions.

    Compute all possible support predicates for each example.
    Separate the example runs into a set $S$ of situation-action pairs.
    Repeat
        Find a consistent rule $R = C \rightarrow A$.
        Remove from $S$ the examples for which $R$ is used.
        Add $R$ at the end of the strategy.
    Until $S = \emptyset$ or there are no consistent rules.
    If $S = \emptyset$ then output the strategy collected so far and stop.
    Otherwise, initialize the strategy to the empty list,
      and go to the next iteration.
```

Consider a scenario in which the learner has acquired some subroutines, say $S_1, \ldots, S_l$, and is trying to learn a new strategy which uses the subroutines as primitive actions. We consider two possible control structures for such strategies. The first can be thought of as "interruptible hierarchical strategies": in each time step the conditions are tested top-down and if a subroutine is used it is given control for a single time step (just like a basic operation). The subroutine will be used in the next time step only if all the conditions of higher priority did not match, and its own invocation condition still matches. This is exactly the control structure used by the "teleo-reactive" programs (Nilsson, 1994). In this model, the execution of a subroutine is verified one step at a time, and therefore *for stationary strategies* learning can be done using the greedy algorithm as before. Namely, for each rule it is possible to decide whether the rule is consistent with the examples. If a subroutine is used, one just needs to check which action is taken by the subroutine on the current input. Notice, though, that this only works if the number of basic rules the learning algorithm has to consider is still polynomial. Namely, we must restrict the number of possible new actions as exhibited by the subroutines. As defined so far, a PRS may be associated with an exponential number of goals, since any conjunction of predicates is in principle allowed as a goal. Therefore, there are too many possibilities for calling a subroutine, or in other words too many new actions. There are several possibilities for such restrictions; in particular we could concentrate on "short goals". Fix such a restriction and denote this class of hierarchical strategies with no internal state by *k*-RIH-IP-SD-PPRS and the class with internal state by *k*-RIH-IS-IP-SD-PPRS (RIH stands for restricted interruptible hierarchical). We therefore get:

Corollary 6.1. *There is a* learn to act *algorithm with respect to the class of k-RIH-IP-SD-PPRS action strategies.*

Internal state can also be handled if more information is supplied in the examples. A helpful teacher will supply examples of the complex strategy, and will annotate each action with the main strategy or the name of the subroutine that is responsible for it. It is clear that with such annotated examples we can learn action strategies with the same algorithm as before. To do that, simply rewrite the examples by taking out sub-sequences which belong to subroutines and replacing a sub-sequence with the corresponding new action name S_i.

Corollary 6.2. *There is a* learn to act *algorithm with respect to the class of k-RIH-IS-IP-SD-PPRS action strategies*, *when given access to a source of annotated random examples.*

In the second control structure, once a subroutine is invoked it is responsible for choosing the actions until its goal has been achieved. After that, control is returned to the main strategy. Thus, subroutines are used just as in standard programming languages. We therefore get some sequential structure to the runs of the strategy, since the choice of actions depends on the current subroutine being run. Denote this class of strategies by k-RH-IS-IP-SD-PPRS (RH stands for restricted hierarchical). It is easy to see that annotation is sufficient to guarantee learnability of this class as well.

Corollary 6.3. *There is a* learn to act *algorithm with respect to the class of k-RH-IS-IP-SD-PPRS action strategies*, *when given access to a source of annotated random examples.*

Unfortunately, as we show below, without annotation the task is computationally hard, even for propositional PRS with $k = 1$, two actions, (and no hidden literals or internal state), and even when only one subroutine is used. Intuitively, the greedy algorithm fails since the rules in the main part of the strategy do not seem consistent due to the sequential running mode of the subroutine. In some sense one has to resort to finding a good annotation of the examples which is hard.

Recall that by k-PPRS we denote the class of propositional PRS as discussed in Section 5.1. In the following discussion we assume that there are only two actions denoted by A and B. We define the problem H-PRS as follows:

H-PRS: Hierarchical PRS Consistency

Input: a subroutine S in 1-PPRS form, and a set E of example runs.
Output: Yes iff there is an action strategy in 1-PPRS form which may use S as a subroutine, and which is consistent with all the examples in E.

Example: A possible input for H-PRS is the subroutine $x_1 \rightarrow A$; $x_2 \rightarrow B$; $\texttt{True} \rightarrow A$, with goal $x_4 = 1$, and where the priority is from left to right, and the two example runs: $R_1 =$ 01000, A, 01100, B, 10100, A, 11011, and $R_2 =$ 10100, B, 11100, A, 11010, A, 10011. The goal of the PRS being learned is to achieve $x_5 = 1$ which is indeed satisfied in the last state of the example runs. As discussed above given the right annotation for the examples it is easy to determine whether there is a consistent strategy. For example, using the information that for R_1 the second and third actions are taken by S, and for R_2 the second action is taken by S, it is easy to see that the 1-PPRS $\bar{x}_3 \rightarrow A$; $x_2 \rightarrow S$; $x_1 \rightarrow B$ is consistent with the example runs. However, as the following theorem shows, without this information the problem is hard.

Theorem 6.4. *The problem H-PRS is NP-complete.*

A "representation dependent" hardness of learning follows from standard arguments (Pitt & Valiant, 1988; Haussler, 1989). The proof of the theorem appears in the Appendix.

7. Practical considerations

The running time of our learning algorithm is exponential in the number of free variables and the width of the conditions in rules. We have assumed that these are bounded by small constants so as to obtain polynomial bounds. The question arises therefore whether these assumptions are not too restrictive. That is, whether the algorithm can be applied in practice and whether the approach is feasible for problems of interest.

The above bounds ignored various optimizations that may be performed, though possibly hindering simple analysis. Such techniques are obviously needed in a practical setting. An experimental evaluation of the applicability of these results has been recently performed (Khardon, 1997) and various such techniques implemented. The system L2ACT essentially implements the algorithm for learning relational strategies (without support predicate and internal state) as in Corollary 5.6. This algorithm is applied to small planning domains that have been studied before, including a four-operator version of the blocks world, and the logistics domain (Veloso, 1992). The experiments demonstrate that our results are indeed applicable, that rule-based strategies are useful for such domains, and that the algorithm is even robust to some extent to "noise" in the examples, namely to cases where there is no strategy that is exactly consistent with the examples. Furthermore, the relational representation enables the use of the learned strategies on instances where the number of objects is much larger than the number of objects in the training examples. While an extensive discussion is beyond the scope of this paper, we briefly discuss some of the issues and illustrate the results. More information regarding the statistical setup and parameters in the experiments, as well as other practical issues are described elsewhere (Khardon, 1997).

The system uses several techniques for efficient enumeration of rules and bindings. One of those utilizes the fact that the running time of the system strongly depends on the time to check whether an example matches a rule. This can be reduced considerably by sharing information between matchings of different rules. Another technique controls complexity by considering only rules that cover a non-negligible part of the examples. This pruning method can be performed using ideas developed recently for data mining of association rules (Agrawal, Imielinski, & Swami, 1993) where conjunctive conditions are enumerated. A significant improvement can be gained in domains where object types are important, as in the logistics domain. In such cases the type information can be automatically gleaned from the examples and reduce both the number of rules enumerated and the time spent on solving the binding problem (since combinations that are ruled out by the type information can be avoided). Finally, our results showed that the agent does not need to have models of the actions (i.e., what happens when an action is taken and some conditions hold). For planning problems, however, this information is readily available and can be incorporated into the algorithm. In particular, by incorporating the preconditions of an action into the condition of every rule that uses this action, we effectively reduce the size of the conditions that have to be searched for. Thus, while in principle one can do without action models they can reduce the complexity considerably.

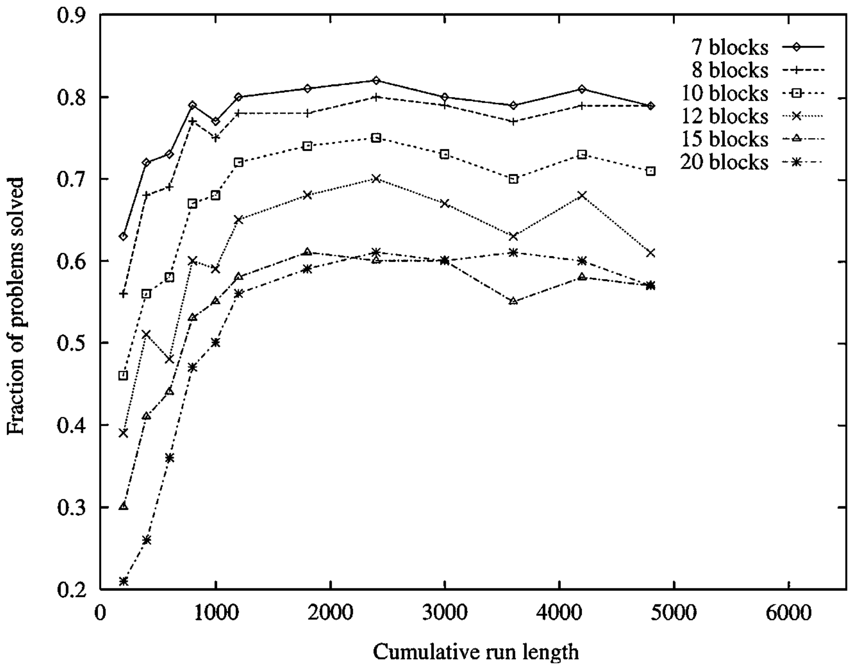

Figure 2. Learning in the blocks world domain.

Figure 2 illustrates the performance of the system on the four operator version of the blocks world. This domain captures the same problem discussed earlier in the paper, but involves more predicates and operations and hence makes for a more demanding learning problem. Examples in this experiment were generated by running a planner to solve random problems each with 8 blocks; hence, no consistent strategy exists and the examples are "noisy" as discussed above. The learned strategies were then evaluated on problems of several sizes. A strategy is deemed to have succeeded only when it finds a complete solution to the problem (i.e., no partial credit was given). The X-axis in figure 2 measures the cumulative run length of examples seen[7] and the Y-axis describes the success rate. As can be observed the learning algorithm indeed converges and about 80% of problems of the same size and 60% of large problems (with 20 blocks) were solved by the strategies learned. The latter problems are beyond the size that can be solved by the planner that produces the examples. In addition other measurements demonstrated that the solutions produced by the learned strategies were better than known heuristics for the problem in terms of the length of the solutions.

For the logistics domain, examples produced by a planner were too "noisy" for our learning algorithm. Examples generated by a hand coded strategy (for which again there was no consistent strategy) produced learning performance similar to the blocks world domain.

The experiments were run with large numbers of examples and the results indeed demonstrate that our approach is feasible and may be used in small planning domains of interest.

The tradeoff between expressiveness and running time is still an important issue and more efficient algorithms if found will certainly be of use.

8. Related work

This work draws on several previous lines of research on learning, planning, and rule-based systems of which only some were mentioned above. We next discuss our model in light of related work emphasizing some differences and thus perhaps clarifying when it might be useful.

8.1. Computational learning theory

From a learning theory perspective (Valiant, 1984; Kearns & Vazirani, 1994) our work extends the scope of problems studied into the domain of goal directed agents, acting in structural non-stationary domains. An important aspect of the current paper is the adoption of a "PAC-semantics" for the problem of acting. In Valiant's (1984) model it is assumed that the world may be very complex and hard to describe, whereas the agent tries to find simple classification rules that will help it cope with this world. The world is thus modeled as a stationary probability distribution over the input domain that may be arbitrarily complex. The agent sees examples of a particular concept drawn from this distribution and tries to find a classification rule that is good relative to the distribution, without knowing what the distribution actually is. In this way probably-approximately-correctness (PAC) is guaranteed regardless of the complexity of the world, and with no attempt to model it. We refer to this notion of correctness as PAC-semantics. Our formulation follows the same line defining the PAC-semantics relative to a non-stationary world, and the "contemporary state of procedural knowledge" as supplied by the teacher. We show that results on Occam algorithms still hold in this extended framework, and generalize Rivest's (1987) arguments for structural domains. On the other hand, new questions arise as for validity of other approaches to learning, and possible benefits or disadvantages that may exist in the extended framework.

8.2. Planning

Planning and acting have been mainly studied in AI with a logical perspective, where knowledge about the world is encoded in declarative form. In order to achieve goals, one proves that they are true in some world state, and as a side effect derives a plan for these goals (McCarthy, 1958). Similarly, in partial order planning declarative information is given, and search in plan space is performed to find a plan (Weld, 1994). However, the problems involved in these approaches are computationally hard (Cook, 1971; Bylander, 1994). Recently, the approach has been generalized to handle stochastic domains, but as this is a generalization of the planning problem similar computational difficulties arise. Since the planning problem is computationally hard we cannot hope to find a new efficient solution to the problem. The main line of research in planning tries to remedy this situation by finding algorithms which are efficient for real-world planning problems.[8] Instead, our approach is to reformulate the problem so that the competence required of an agent is defined relative to its learning interface (its teacher). The new formulation will naturally not fit all possible aspects of planning and in particular it is not intended as a generic optimization procedure. It offers, however, a complementary view on the problems which can thus lead to different kinds of solutions.

8.3. Universal plans

Our approach is also reminiscent of Schoppers' (1987) universal plans, as well as some other works on "reactive agents" (Georgeff & Lansky, 1987; Brooks, 1991; Maes, 1991; Nilsson, 1994). A universal plan describes an algorithm for a particular domain so that the action in each situation is in some sense pre-compiled and can be taken instantly. Our action strategies are similar in that respect; they provide an efficient way to choose an action in every situation, notably, without using a world model or performing any search. Our work elaborates on that idea in two respects. First, we suggest that the strategies be learned. (Schoppers' original scheme tried to compile the universal plan in a manner not far from that of traditional planners.) Second, the semantics of "universality" of strategies is not the same. While universal plans should solve all instances, our strategies are only required to solve a fraction of them, similar to what the teacher can achieve. This is important since various negative results regarding strategies that solve all instances have been obtained (Selman, 1994; Jonsson & Bäckström, 1996). In fact, it was suggested (Schoppers, 1989) that it may be sufficient for universal plans to solve only a subset of problem instances. Our formulation indicates how this might be done using PAC-semantics to choose which subset of instances to consider, and how to acquire such a strategy. As mentioned above our results hold for Nilsson's (1994) teleo-reactive programs, supplying one form of learnable universal plans under the PAC-semantics.

8.4. Reinforcement learning

The problem of learning and acting in stochastic worlds is studied in reinforcement learning (RL) (Kaelbling, Littman, & Moore, 1996). In fact our world model as a dynamic stochastic partially observable state machine is borrowed from the RL paradigm. However, our formalization differs in important aspects. Most importantly, the learning model in RL is unsupervised, that is, no teacher or examples are given; the agent receives information by acting in the world and receiving some reinforcement from the world as a result of its actions. Another difference is that normally the learner is intended to find an optimal strategy for acting in the world. Our formalization makes the task easier in both these aspects. Several interesting theoretical and empirical results have been obtained for RL (Sutton, 1988, 1990; Watkins & Dayan, 1992; Kaelbling, 1993; Fiechter, 1994; Tesauro, 1995). In particular, the success of Tesauro's (1992, 1995) backgammon playing program is remarkable. However, the unsupervised learning problem being solved in RL is very general, and the formulation does not admit efficient solutions.[9] Theoretically derived solutions typically enumerate the state space. Our work provides an alternative formulation of the problem that allows for provably correct and efficient solutions. This is obtained by using examples for behavior and by relaxing the requirement for optimality. As discussed above the reformulation is not intended to solve the original RL problem. Instead, supervised learning being easier than unsupervised learning, the model and results would be useful whenever our conditions hold. In particular it may be useful for structural domains whose state space is large and where the number of objects is not fixed in advance.

8.5. Rule-based systems

Our model can be seen to suggest an engineering principle where all agents use a single language for representing their strategies, and the language is chosen so as to ensure learnability. In this way, when facing a new problem an agent can use any unsupervised or search method at its disposal, but once a strategy is acquired by one agent it can be transferred to other agents by way of demonstration through examples and learning from these examples. Rule-based systems are particularly interesting on this account since several algorithms for using and learning such systems have already been studied. In particular, algorithms for classifier systems (Booker, Goldberg, & Holland, 1989) applied to problem of acting in a dynamic world (Grefenstette, Ramsey, & Schultz, 1990; Baum, 1996) are rule-based and can therefore be used in combination with our algorithm in this manner. Recently, Lin (1993) demonstrated that ideas of supervised learning can be useful in RL tasks, by incorporating examples and using a hierarchical decomposition of tasks, in combination with a temporal difference algorithm. Our work can be seen as a partial formalization of this effort, quantifying the utility of supervision; a combination of the two approaches can again be done via the rule-based systems of (Grefenstette, Ramsey, & Schultz, 1990; Baum, 1996). Another nice property of rule-based systems is that it allows for a combination of reactive condition-action rules, and declarative knowledge that can be used for search, under the same framework. Our representation indeed incorporates both reactive rules and declarative rules but does not use search.

8.6. Speedup learning

Our work is also closely related to work in Explanation Based Learning (EBL) and speedup learning (Rosenbloom & Laird, 1986; DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986; Minton, 1990; Veloso et al., 1995). Generally speaking this line of work tries to compile declarative knowledge into a more procedural form via some form of learning. In EBL, solved problems are “explained”, namely, the declarative knowledge is used to find minimal conditions so that the solutions are still valid; these generalized explanations are then used in the system as guidance when solving new problems. The explanations are used either as control knowledge or just added as new rules into the knowledge base. In some of this work (DeJong & Mooney, 1986) solved problems are supplied by a teacher as in our case, but others use a general search engine to find such solutions. Our approach is similar in the effort to use learning for the purpose of finding efficient strategies for acting (or solving problems). It differs however in the method of learning and more importantly in the fact that the output of the learner is not used as part of a search engine. Instead, the rule-based strategies that we show learnable can operate without search, in the spirit of universal plans discussed above. In this line, our model is closest to several works by Natarajan and Tadepalli (Natarajan, 1989; Natarajan & Tadepalli, 1988; Tadepalli, 1991, 1992; Tadepalli & Natarajan, 1996) who formalize learnability of action strategies in deterministic domains, combining some aspects of speedup learning with the PAC learning framework. Our formalization in fact generalizes the model presented by Tadepalli and Natarajan (1996) to deal with stochastic domains. A related approach considering bias in the context of EBL is discussed by DeJong and Bennett (1995).

8.7. Inductive logic programming

The rules used in our strategies incorporate first order conjunctive conditions. The learning problem is therefore technically similar to that of Inductive Logic Programming (ILP) (Muggleton, 1994; Muggleton & De Raedt, 1994). However, the models differ in details that are crucial. One source of difference is the structure of examples. An example in the standard form of ILP (Quinlan, 1990; Dzeroski, Muggleton, & Russell, 1992, Muggleton, 1994; Mooney & Califf, 1995) includes a single ground instance of a relation and the rest of the information on this example is provided through the background knowledge. In contrast an example in our model describes a complete situation and the ground action taken in that situation, and is therefore more explicit. On the other hand, since the state information changes from one step to the next, in some sense our examples have “changing background knowledge” in ILP terms. The non-monotonic setting of ILP (De Raedt & Dzeroski, 1994; Muggleton & De Raedt, 1994) uses interpretations as examples and is thus similar to our form, but the task there is different. Our formalization comes closest to Cohen’s (1995) use of extended instances as examples. Another difference results from the fact that clauses in ILP are taken as generally applicable logical rules and therefore any instantiation of a rule must be valid in the examples. On the other hand our rules are procedural and support a single action in each situation, namely one of the instantiations is preferred to others. Despite those differences, the structure of induced expressions is similar, and similar techniques can be used in both models. Our learning approach is similar to the covering method in FOIL (Quinlan, 1990), and the representation is similar to the first order decision lists studied by Mooney and Califf (1995). Moreover, our arguments are similar to the ones in (De Raedt & Dzeroski, 1994; Valiant, 1985) and can yield positive results on learning first order decision lists in the ILP context. On the other hand, several sophisticated methods for learning have been applied in ILP that may be useful in our framework.

9. Conclusions

We presented a new framework, called *learning to act*, for the study of supervised learning of action strategies in dynamic stochastic domains. When learning to act, one does not use a general problem solver in order to choose its actions. Instead, some interaction with a teacher enables the agent to learn about the domain in question. Using the information thus collected, the agent can efficiently solve future instances of the problem. Furthermore, the performance is measured relative to the teacher so that the agent does not need to achieve optimal performance. We have shown that in this model Occam algorithms, which find strategies which are consistent with the examples, are good learning algorithms. We have also shown that some rule-based strategies enjoy such learning algorithms, and derived positive and negative results for the learnability of hierarchical strategies.

Our model reformulates the problem of acting in the world diverging both from the planning and the reinforcement learning perspectives. In particular our assumptions on the existence of examples and restricted classes of strategies made for tractable learning problems. The important point is that the supervised learning approach can be used for these complex problems and that analysis can be performed. More research is needed to find better algorithms and analysis, other representations for strategies, and refinements of the model. The questions of learnability of randomized strategies, and cases where there is "noise" in the examples, can be pursued. Another issue is whether ideas from EBL or ILP can be used to derive better results.

Another direction for further work is the use of different models of interaction with the environment. An interesting model is suggested by Natarajan (1989) who studies learning to act in deterministic domains. A notion of exercises is formulated where the learner does not get solved problems as examples but instead it first has to tackle easy problems and the difficulty is increased gradually. It seems that several experimental systems of learning to act have used this idea implicitly. A formalization of this idea for stochastic domains may be useful in studying the behavior of such systems.

Finally, the success of the system L2ACT (Khardon, 1997) and similarly of many AI systems relies heavily on the selection of a small set of predicates for describing the domain in question. In any large-scale system, however, one might have many predicates of which only a small number may be relevant to a particular task. Therefore, the algorithms used must be efficient even when many irrelevant predicates exist in the system. This issue is discussed by Valiant (1996) who suggests that linear threshold elements be used to represent prioritized rule-based systems and that in this way learning algorithms that can tolerate irrelevant attributes can be used. Preliminary progress in this direction was made (Khardon, 1997) by adapting Littlestone's (1988) Winnow algorithm to deal with relational rule-based action strategies.

Appendix

A. Proof of Theorem 6.4

First observe that the problem is in NP since we can guess a 1-PPRS and check whether it is consistent with E. We reduce the satisfiability problem 3SAT (Garey & Johnson, 1979) to H-PRS.

We are given a 3-CNF expression, with m clauses, $f = c_1 \wedge c_2 \wedge \cdots \wedge c_m$, on n variables $x_1, \ldots, x_n$, and translate it into a set of $m + 2n$ example runs, where we use $2n + 3$ variables. The variables for the H-PRS problem would be $y_1, \ldots, y_n$, $z_1, \ldots, z_n$, and q_1, q_2, q_3. Intuitively, y_i corresponds to x_i, and z_i corresponds to $\bar{x}_i$. The variables q_i serve for special functions; q_1 is the goal of the strategy being learned, q_2 is the goal of the subroutine S, and q_3 is an additional variable.

Table 4. The constructions used for the H-PRS reduction.

Examples runs used in the reduction:

- Type (1) includes $1_{y_i,z_i}$, A, $1_{y_i,z_i}$, A, 1_{q_1}, for $i \le n$.
- Type (2) includes 1_{c_i}, A, 1_{q_3}, A, 1_{q_1}, for $i \le m$.
- Type (3) includes 1_{y_i}, A, 1_{z_i}, B, $1_{q_1,q_2}$, for $i \le n$.

The Subroutine S (goal is q_2):	**The Main PRS H_1 (goal is q_1):**
• $q_3 \to B$	• $q_3 \to A$
• $z_1 \to B$	• $\alpha_1 \to A$ ($\alpha_1 \in \{y_1, z_1\}$)
• $z_2 \to B$	• $\alpha_2 \to A$ ($\alpha_2 \in \{y_2, z_2\}$)
• ...	• ...
• $z_n \to B$	• $\alpha_n \to A$ ($\alpha_n \in \{y_n, z_n\}$)
• `True` $\to A$	• `True` $\to S$

The subroutine S whose goal is q_2 is described in Table 4, and consists of the following 1-PPRS: $q_3 \to B$; $z_1 \to B$; $z_2 \to B$; ...; $z_n \to B$; `True` $\to A$. Namely, if q_3 is 1, or any z_i is 1, it outputs B and otherwise it outputs A.

We produce three types of example runs, for which we use the following notation (Pitt & Valiant, 1988; Haussler, 1989): for $\alpha \subseteq \{y_1, \ldots, y_n, z_1, \ldots, z_n, q_1, q_2, q_3\}$, the assignment 1_α is the assignment in which all the variables in α are assigned 1, and all other variables are assigned 0. For example, if $n = 2$ then $1_{y_2,z_1,q_1} = (01\ 10\ 100)$. The runs are listed in Table 4, and are all of length 2.

Runs of type (1) include the runs $1_{y_i,z_i}$, A, $1_{y_i,z_i}$, A, 1_{q_1}, for $i \le n$.
Runs of type (2) include the runs 1_{c_i}, A, 1_{q_3}, A, 1_{q_1}, for $i \le m$, where the literals in c_i are translated to the corresponding y_j, z_j. For example, if $n = 2$ and $c_i = (x_1 \vee \bar{x}_2)$ then $1_{c_i} = (10\ 01\ 000)$.
Runs of type (3) include the runs 1_{y_i}, A, 1_{z_i}, B, $1_{q_1,q_2}$, for $i \le n$.

We now show that there is a 1-PPRS consistent with this set of runs if and only if the CNF expression f is satisfiable. The main idea is that runs in types (1), and (3) force a consistent strategy to choose exactly one of $y_i \to A$, $z_i \to A$, to be on the list. Moreover, these are the only rules on the list that can produce the action A (except for usage of S). Therefore, runs of type (2), for which S is not consistent, must be produced by these rules, and since runs in type (2) encode the clauses of f, the choice of y_i or z_i for the rule constitutes a satisfying assignment for f.

More formally, let $v \in \{0, 1\}^n$ be a satisfying assignment for f, and let $\alpha_i = y_i$ if $v_i = 1$, and $\alpha_i = z_i$ if $v_i = 0$. Then, the main strategy H_1 consisting of the list $q_3 \to A$; $\alpha_1 \to A$; $\alpha_2 \to A$; ...; $\alpha_n \to A$; `True` $\to S$ is consistent with the runs. For reference the main strategy H_1 is described in Table 4. The strategy is consistent with runs of type (1) since each run has both y_i, z_i in each state, and one of y_i, z_i appears in the list of α_i. For runs of type (2), observe that since each clause is satisfied by v, the action A on the first step is chosen by the list of α_i, and that in the second step the action A is chosen by the first rule. For runs of type (3), there are two cases. If $v_i = 0$, $\alpha_i = z_i$ and the first step is consistent with S (which is the rule chosen by the strategy). The next step is also consistent with S. If $v_i = 1$, $\alpha_i = y_i$ and the first step is consistent with the rule $\alpha_i \to A$. The second step is consistent with S (which is the rule chosen by the strategy).[10]

For the other direction, assume that there is a strategy H_2 consistent with the example runs. We would argue that the structure of H_2 is very similar to H_1 and show the satisfiability of f.

Notice that $q_3 = 1$ appears only in one place in the runs, and that every time it appears the action A is chosen. We may therefore assume that if q_3 appears in any rule then it is of the form $q_3 \to A$. Also, observe that the value of q_1, q_2 is fixed at 0 for all the runs on which actions are chosen. Therefore without loss of generality we may assume that they do not appear in H_2. We therefore concentrate on rules that do not involve q_1, q_2, q_3.

Consider the first such rule $\alpha \to O$ in H_2. Observe that $O \neq B$ since otherwise H_2 is not consistent with at least one of the runs of type (1). For the same reason we also get $O \neq S$, and therefore $O = A$. Furthermore, we claim that α must be a positive literal (either y_i or z_i for some i). This is true since otherwise runs of type (3) with index different from α would not be consistent with H_2. (That is if $\alpha = \bar{y}_i$ then for $j \neq i$, the run $1_{y_j}, A, 1_{z_j}, B, 1_{q_1,q_2}$ is not consistent with H_2 in the second step.) Therefore, the first rule is of the form $\alpha_i \to A$ where $\alpha_i \in \{y_i, z_i\}$.

Consider the next rule $\alpha \to O$ in H_2 (which does not have q_i). As before we argue that $O \neq B$, and $O \neq S$, and $\alpha \in \{y_i, z_i\}$ for some i. However, we claim that if y_i appeared in some rule $y_i \to A$ earlier on the list then $z_i \to A$ cannot be the current rule. This is true since otherwise the second step of the i'th run of type (3) would not be consistent with H_2. Similarly, if z_i appeared earlier on the list, then the current rule cannot be $y_i \to A$. Therefore the next rule is of the form $\alpha_i \to A$, and it uses a new index i.

We can continue this argument to show that H_2 has a list of n rules similar to the one in H_1; call this the first part of H_2. The next rules on the list cannot choose the action A (since otherwise as argued above runs of type (3) would not be consistent). Therefore they choose either B or S. Notice that if z_i appeared in the first part of H_2 then $y_i \to B$ would not be consistent with runs of type (3). So the possible rules are of the form $z_i \to B$ where y_i appears in the first part of H_2, and any rule choosing S. (Notice that, as in H_1, simply appending S is sufficient.)

We now claim that, since H_2 is consistent with runs of type (2), the expression f is satisfiable. Runs of type (2) are of the form $1_{c_i}, A, 1_{q_3}, A, 1_{q_1}$. Notice that action A can either be chosen using the first part of H_2 or using S in the second part of H_2. However, for runs of type (2) the first action must be chosen using the first part of H_2, since otherwise the second step would be performed by S which chooses B on 1_{q_3}. Therefore each clause must have a literal which is on the first part of the list. Hence, the list of α_i constitutes a satisfying assignment for f.

We therefore get that there is a 1-PPRS consistent with this set of runs if and only if the CNF expression f is satisfiable, completing the proof.

Acknowledgments

I am grateful to Les Valiant for many discussions that helped in developing these ideas, and to Prasad Tadepalli for helpful comments on an earlier draft. I wish to thank Jude Shavlik and the referees of this paper for their comments that helped improve the presentation.

Notes

1. For ease of reference, we use the word *teacher* in a loose sense to refer to the example provider. The model is not intended to capture notions of teaching as such.
2. This is in contrast with previous work (Khardon & Roth, 1995) where a special semantics is given to partial assignments.
3. In fact, since we do not assume anything about the Markov process, the results presented hold for any stochastic process (where the random variable at time t depends on the entire history). It seems useful, however, to think in terms of states and therefore we present the model in this way.
4. The merits of explicit symbolic reasoning on the one hand, and reactive operation on the other, have been debated (Brooks, 1991; Hayes, Ford, & Agnew, 1994; Vera & Simon, 1993; Maes, 1991; Ginsberg, 1989; Chapman, 1989; Schoppers, 1989). However, as recently argued (Vera & Simon, 1993; Hayes, Ford, & Agnew, 1994), neither approach can succeed on its own; ultimately a system must have some reactive features but must also retain forms of symbolic computation. Our work exemplifies this point.
5. Anderson (1983) includes a third component of declarative memory and the operation of the system is intuitively similar, though it differs a lot in details. In outline we will follow the Soar system (Newell, 1990; Rosenbloom, Laird, & Newell, 1993), though not in full detail.
6. The first rule makes constructive moves. The next rules clear the way in case a constructive move cannot be taken. If the block y is already in place and the block x is to be placed on it, then the rules 2 and 4 clear the tower above x (notice that apart from the use of *sad* they are identical) while the rule 3 clears the tower above y. Therefore to an external observer that does not know when the agent is *sad* the choice of which tower to clear first might seem non-deterministic.
7. There were 315 runs (on average) for a cumulative run length of 4800 steps. The learning time for these experiments was roughly 130 minutes on a SUN/20 workstation.
8. See for example the discussion by Kambhampati (1995).
9. This for example follows since RL is a generalization of classical propositional planning.
10. Notice that the hardness of the problem is encoded into the annotation of runs of type (3). The runs of types (1) and (2) are always performed by the main strategy, and for type (3) there is a choice between using the subroutine for both steps, and using the subroutine just for the second step. This choice encodes the assignment for the variable x_i.

References

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *Proceedings of the ACM Conference on Management of Data (SIGMOD)* (pp. 207–216). Washington, DC: ACM Press.

Allen, J., Hendler, J., & Tate, A. (1990). *Readings in planning*. San Mateo, CA: Morgan Kaufmann.

Anderson, J. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, *75*, 87–106.

Baum, E. (1996). Toward a model of mind as a laissez-faire economy of idiots. *Proceedings of the International Conference on Machine Learning* (pp. 28–36). Bari, Italy: Morgan Kaufmann.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M.K. (1987). Occam's razor. *Information Processing Letters*, *24*, 377–380.

Booker, L., Goldberg, D., & Holland, J. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, *40*, 235–282.

Brooks, R.A. (1991). Intelligence without representation. *Artificial Intelligence*, *47*, 139–159.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, *69*, 165–204.

Chapman, D. (1989). Penguins can make cake. *AI Magazine*, *10*(4), 45–50.

Cohen, W. (1995). PAC-learning recursive logic programs: Efficient algorithms. *Journal of Artificial Intelligence Research*, *2*, 501–539.

Cook, S.A. (1971). The complexity of theorem proving procedures. *Proceedings of the 3rd Annual ACM Symposium of the Theory of Computing* (pp. 151–158). Shaker Heights, Ohio: ACM Press.

DeJong, G., & Bennett, S. (1995). Extending classical planning to real world execution with machine learning. *Proceedings of the International Joint Conference of Artificial Intelligence* (pp. 1153–1159). Montreal, Canada: Morgan Kaufmann.

DeJong, G., & Mooney, R. (1986). Explanation based learning: An alternative view. *Machine Learning, 1*, 145–176.

De Raedt, L., & Dzeroski, S. (1994). First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, *70*, 375–392.

Dzeroski, S., Muggleton, S., & Russell, S. (1992). PAC-learnability of determinate logic programs. *Proceedings of the Conference on Computational Learning Theory* (pp. 128–135). Pittsburgh, PA: ACM Press.

Fiechter, C.N. (1994). Efficient reinforcement learning. *Proceedings of the Conference on Computational Learning Theory* (pp. 88–97). New Brunswick, NJ: ACM Press.

Garey, M., & Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: W.H. Freeman.

Georgeff, M., & Lansky, A. (1987). Reactive reasoning and planning. *Proceedings of the National Conference on Artificial Intelligence* (pp. 677–682). Philadelphia, PA: AAAI Press.

Ginsberg, M. (1989). Universal planning: An (almost) universally bad idea. *AI Magazine*, *10*(4), 40–44.

Grefenstette, J., Ramsey, C., & Schultz, A. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, *5*, 355–381.

Gupta, N., & Nau, D. (1991). Complexity results for blocks world planning. *Proceedings of the National Conference on Artificial Intelligence* (pp. 629–633). Anaheim, CA: AAAI Press.

Haussler, D. (1989). Learning conjunctive concepts in structural domains. *Machine Learning*, *4*, 7–40.

Hayes, P., Ford, K., & Agnew, N. (1994). On babies and bathwather. *AI Magazine*, *15*(4), 15–26.

Jonsson, P., & Bäckström, C. (1996). On the size of reactive plans. *Proceedings of the National Conference on Artificial Intelligence* (pp. 1182–1187). Portland, Oregon: AAAI Press.

Kaelbling, L. (1993). *Learning in embedded systems*. Cambridge, MA: MIT Press.

Kaelbling, L., Littman, M., & Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kambhampati, S. (1995). A comparative analysis of partial order planning and task reduction planning. *SIGART Bulletin*, *6*(1), 16–25.

Kearns, M.J., & Schapire, R.E. (1994). Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Sciences*, *48*, 464–497.

Kearns, M., & Vazirani, U. (1994). *An introduction to computational learning theory*. Cambridge, MA: MIT Press.

Khardon, R. (1997). *Learning action strategies for planning domains* (Tech. Rep. TR-09-97). Harvard University: Aiken Computation Lab.

Khardon, R., & Roth, D. (1995). Learning to reason with a restricted view. *Proceedings of the Conference on Computational Learning Theory* (pp. 301–310). Santa Cruz, CA: ACM Press.

Khardon, R., & Roth, D. (1997). Learning to reason. *Journal of the ACM*, *44*, 697–725.

Klahr, D., Langley, P., & Neches, R. (1986). *Production system models of learning and development*. Cambridge, MA: MIT Press.

Korf, R.E. (1985). Macro operators: A weak method for learning. *Artificial Intelligence*, *26*, 35–77.

Laird, J., Rosenbloom, P., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Lin, L. (1993). Scaling up reinforcement learning for robot control. *Proceedings of the International Conference on Machine Learning* (pp. 182–189). Amherst, MA: Morgan Kaufmann.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2,* 285–318.

Littman, M., Cassandra, A., & Kaelbling, L. (1995). Learning policies for partially observable environments: Scaling up. *Proceedings of the International Conference on Machine Learning* (pp. 362–370). Tahoe, CA: Morgan Kaufmann.

Maes, P. (1991). Situated agents can have goals. In P. Maes (Ed.), *Designing autonomous agents* (pp. 49–70). Cambridge, MA: MIT Press.

McCarthy, J. (1958). Programs with common sense. *Proceedings of the Symposium on the Mechanization of Thought Processes* (Vol. 1, pp. 77–84), National Physical Laboratory. Reprinted in R. Brachman and H. Levesque (Eds.), *Readings in Knowledge Representation*, 1985, Los Altos, CA: Morgan Kaufmann.

Minton, S. (1990). Quantitative results concerning the utility of explanation based learning. *Artificial Intelligence*, *42*, 363–391.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation based learning: A unifying view. *Machine Learning, 1*, 47–80.

Mooney, R.J., & Califf, M.E. (1995). Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research*, *3*, 1–24.

Muggleton, S. (1994). Inductive logic programming: Derivations, successes and shortcomings. *SIGART Bulletin*, *5*(1), 5–11.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *20*, 629–679.

Natarajan, B.K. (1989). On learning from exercises. *Proceedings of the Conference on Computational Learning Theory* (pp. 72–87). Santa Cruz, CA: Morgan Kaufmann.

Natarajan, B.K., & Tadepalli, P. (1988). Two new frameworks for learning. *Proceedings of the International Conference on Machine Learning* (pp. 402–415). Ann Arbor, Michigan: Morgan Kaufmann.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Newell, A., & Simon, H.A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Nilsson, N.J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*, 139–158.

Pitt, L., & Valiant, L.G. (1988). Computational limitations on learning from examples. *Journal of the ACM*, *35*, 965–984.

Quinlan, J.R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

Rivest, R.L. (1987). Learning decision lists. *Machine Learning*, *2*, 229–246.

Rosenbloom, P., & Laird, J. (1986). Mapping explanation based learning onto Soar. *Proceedings of the National Conference on Artificial Intelligence* (pp. 561–567). Philadelphia, PA: AAAI Press.

Rosenbloom, P.S., Laird, J.E., & Newell, A. (1993). *The Soar papers : Research on integrated intelligence*. Cambridge, MA: MIT Press.

Roth, D. (1995). Learning to reason: The non-monotonic case. *Proceedings of the International Joint Conference of Artificial Intelligence* (pp. 1178–1184). Montreal, Canada: Morgan Kaufmann.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. *Proceedings of the International Conference on Machine Learning* (pp. 385–393). Aberdeen, Scotland: Morgan Kaufmann.

Schoppers, M. (1987). Universal plans for reactive robots in unpredictable domains. *Proceedings of the International Joint Conference of Artificial Intelligence* (pp. 1039–1046). Milan, Italy: Morgan Kaufmann.

Schoppers, M. (1989). In defense of reaction plans as caches. *AI Magazine*, *10*(4), 51–62.

Selman, B. (1994). Near-optimal plans, tractability, and reactivity. *Proceedings of the International Conference on Knowledge Representation and Reasoning* (pp. 521–529). Bonn, Germany: Morgan Kaufmann.

Shavlik, J.W. (1990). Acquiring recursive and iterative concepts with explanation based learning. *Machine Learning*, *5*, 39–70.

Slaney, J., & Thiebaux, S. (1996). Linear time near-optimal planning in the blocks world. *Proceedings of the National Conference on Artificial Intelligence* (pp. 1208–1214). Portland, Oregon: AAAI Press.

Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R.S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the International Conference on Machine Learning* (pp. 216–224). Austin, Texas: Morgan Kaufmann.

Tadepalli, P. (1991). A formalization of explanation based macro-operator learning. *Proceedings of the International Joint Conference of Artificial Intelligence* (pp. 616–622). Sydney, Australia: Morgan Kaufmann.

Tadepalli, P. (1992). A theory of unsupervised speedup learning. *Proceedings of the National Conference on Artificial Intelligence* (pp. 229–234). San Jose, CA: AAAI Press.

Tadepalli, P., & Natarajan, B. (1996). A formal framework for speedup learning from problems and solutions. *Journal of Artificial Intelligence Research, 4,* 445–475.

Tesauro, G. (1992). Temporal difference learning of backgammon strategy. *Proceedings of the International Conference on Machine Learning* (pp. 451–457). Aberdeen, Scotland: Morgan Kaufmann.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*, 58–68.

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

Valiant, L.G. (1985). Learning disjunctions of conjunctions. *Proceedings of the International Joint Conference of Artificial Intelligence* (pp. 560–566). Los Angeles, CA: Morgan Kaufmann.

Valiant, L.G. (1994). *Circuits of the mind*. Oxford, UK: Oxford University Press.

Valiant, L.G. (1995). Rationality. *Proceedings of the Conference on Computational Learning Theory* (pp. 3–14). Santa Cruz, CA: ACM Press.

Valiant, L.G. (1996). A *neuroidal architecture for cognitive computation* (Tech. Rep. TR-11-96). Harvard University: Aiken Computation Lab.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence*, *31*, 1–40.

Veloso, M. (1992). *Learning by analogical reasoning in general problem solving*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University. Also appeared as Technical Report CMU-CS-92-174.

Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating learning and planning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, *7*, 81–120.

Vera, A., & Simon, H. (1993). Situated action: A symbolic interpretation. *Cognitive Science*, *17*, 7–48.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Weld, D. (1994). An introduction to least commitment planning. *AI Magazine*, *15*(4), 27–61.

Zelle, J.M., & Mooney, R.J. (1994). Inducing deterministic Prolog parsers from treebanks: A machine learning approach. *Proceedings of the National Conference on Artificial Intelligence* (pp. 748–753). Seattle, Washington: AAAI Press.

Received July 1, 1996
Accepted June 26, 1998
Final manuscript June 26, 1998