Discovery as Autonomous Learning from the Environment

Wei-Min Shen

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, TX 78759
wshen@mcc.com

September 9, 1994

Abstract

Discovery involves collaboration among many intelligent activities. However, little is known about how and in what form such collaboration occurs. In this paper, a framework is proposed for autonomous systems that learn and discover from their environment. Within this framework, many intelligent activities such as perception, action, exploration, experimentation, learning, problem solving, and new term construction can be integrated in a coherent way. The framework is presented in detail through an implemented system called LIVE, and is evaluated through the performance of LIVE on several discovery tasks. The conclusion is that autonomous learning from the environment is a feasible approach for integrating the activities involved in a discovery process.

1 Introduction

Learning from the environment requires integration of a variety of activities. A learning system must be able to explore, to plan, to experiment, to adapt, and to discover. These activities should be studied together in a coherent way so that they can benefit each other. For example, cooperation between prediction, model construction, problem solving, exploration, experimentation, and new term construction can be seen as follows: prediction is used as an evaluation criterion for model construction; model construction (refinement) provides a means for improving prediction; problem solving makes use of the approximate model and detects when and where exploration and experimentation are needed to further improve the model; new term construction provides more building blocks for construction, prediction, and problem solving.

The main purpose of this paper is to define the problem of learning from the environment and show how various intelligent activities can be integrated. The paper is organized as follows. Section 2 discusses related work on learning from the environment. Section 3 defines the problem of learning from the environment. Section 4 introduces the notation for prediction sequences and their key role in integration. Sections 5 through 8 present an implemented system called LIVE and describe how each of the integrated activities works. I will use a variation of the Tower of Hanoi puzzle as the main example to illustrate most of the ideas. The paper concludes with a discussion on the strengths and limitations of this approach.

Figure 1: The basic definition of learning from the environment

Figure 1 shows the relation between the learner and the environment. The learner is a system that can perform a set of actions to the environment and perceive a set of percepts from the environment. It has a set of goals, expressed in terms of percepts, that are either self-generated or given by external commands. Its objective is to construct a model of the environment so that it can drive the environment into states that match its goals.

The environment is a system that changes its states when acted upon. The environment changes its state according to some set of unknown rules or functions. The state of the environment is not necessarily

completely perceived by the learner; it depends on the physical ability of the learner. For example, a color blind learner cannot perceive the color of an object even though most objects in the environment may have colors.

In this paper, I assume that the environment is only manipulated by the learner and there is no noise in the observations and actions. Readers interested in "noisy" environments may find adaptive control theories [Goodwin and Sin, 1984] useful. Moreover, I assume that the learner has sufficient computational resources to process all of the perceived percepts. Readers interested in the problem of "focus of attention" can find more information in [Whitehead and Ballard, 1991].

To be more precise about the learner, we need to define the terms action, percept, mental language, goal, and model. An *action* is a physical change that occurs inside the learner. It is typically a signal sent to the interface devices that will effect the environment. For example, it can be a muscle contraction inside a human's arm, or a signal sent to the motor that controls a robot arm. Note that an action is separated from its external consequence. A learner can execute its action in any environment regardless of the consequence of the action. For instance, a contraction of one's arm muscle can be executed regardless of whether the arm is free to move. Ideally, actions are innate to learners while consequences depend on environments. In this paper, however, actions may still have external objects as their parameters.

A *percept* is a representation inside the learner of information received from the environment. The devices for creating such an internal representation from the environment are innate to the learner. For humans, such devices are the sensing organs, and percepts are the signals that are sent to the brain from the sensing organs. In this paper, I assume that a percept can be a representation of an object (e.g., a book), a feature (e.g., blue), a function (e.g., *Color: object→feature*), or a relation (e.g., *On: object×object*). Of course, a percept can be something about the learner itself, such as whether the hand is open and how much the left arm is bent. We shall call the set of all percepts perceived from a state of the environment an *observation*.

A *goal* is a specific set of percepts representing some state of the environment. We say that a goal is satisfied in an observation if the percepts of the goal are a subset of the observation. For example, a goal that is represented by a singleton set of percept {"blue-triangle"} is satisfied in all observations that contain the percept "blue-triangle."

Actions and percepts are part of a *mental language* of a learner. Besides the actions and percepts, the language may also have other primitive symbols, such as logical quantifiers and connectives (e.g., $\exists, \wedge, \neg$), predicates (e.g., $=, >$), or functions (e.g., $+, -, \times$). In this paper, I assume each learner has a fixed mental language. The learner uses this language to build new mental concepts. For example, if "force" f and "distance" d are percepts and multiplication $\times$ is a function, then a term "moment" can be defined as $(f \times d)$. Mental concepts can be built using not only percepts but also actions. For example, if "push" is an action, and "location" is a percept, then the concept "heavy" (not directly perceivable) can be built as "the location before pushing is equal to the location after pushing." Concepts like this example are conjectures made dynamically by the learner; they are not directly perceived from the environment. The major use of the mental language is to define models of the environment.

A *model* of the environment is defined as a set of *prediction rules*. A prediction rule is a triple ⟨condition action prediction⟩, where *condition* and *prediction* are expressions in the mental language that represent sets of percepts that have certain properties (e.g., what they are, how they are related, and how they have come about). A condition or a prediction is satisfied in an observation if the observation satisfies the properties specified in the condition or the prediction. A prediction rule says that if the action is applied to a state whose observation satisfies the condition, then the observation of the resulting state *should* satisfy the prediction. It is a *prediction failure* if the observation of the resulting state does not satisfy the prediction. Note that prediction rules are different from STRIPS operators [Fikes and Nilsson, 1971] in the way they are used. A prediction in a prediction rule is only a template to be matched to observations of states. It does not change states as the add/delete list would in STRIPS's case. A prediction can fail, while add/delete lists, as they are used in STRIPS, cannot.

An interesting property of this kind of model is its bidirectional usage. Used in the forward direction, it can make predictions to monitor the consequences of executing actions (to detect the deficiency of the

current model). Used in the backwards direction, it can decompose a goal into subgoals in the planning process, thus selecting "control" actions.

Now, with all the definitions in place, the precise definition of learning from the environment can be given as follows:

> *Given an Environment, a Learner, defined as a set of Actions, Percepts, and a Mental Language, is to learn a set of Prediction Rules that enable it to achieve its Goals deliberately (i.e., no prediction failure should occur in the solution plan).*

To illustrate the definition of learning from the environment, consider a special implementation of the Tower of Hanoi puzzle. There are three balls and three plates. The balls have different sizes and they can be moved from one plate to another according to the following rules: (1) only one ball can be picked up from the plates at a time. (2) A ball can be put onto a plate only if that ball is smaller than all the balls on that plate. (Attempts to put a ball into a plate that contains any smaller ball will cause the larger ball to be popped onto the table). (3) A ball can be picked up from a plate only if it is the smallest on that plate. The reason I use balls and plates instead of disks and pegs is that these three rules are not given to the puzzle solver. Instead, they must be learned through the interaction with the device. This environment may seem a little artificial, but I chose the puzzle for its familiarity in the literature of Artificial Intelligence. For convenience, let us call the puzzle the "Plate of Hanoi."

Let us define a learner, RML, in this environment as follows. It can see certain relations between objects. In particular, it can see ON(ball,plate/table), SIZE>(ballx,bally), and INHAND(ball). It can PICK balls up from or PUT balls on the plates and the table. Let us assume that the learner's mental language also has the primitives $\wedge$, $\neg$ and $\exists$, and its goal is to move all the balls to a particular plate, say PLATE3. (Let us assume BALL1 is smaller than BALL2, and BALL2 is smaller than BALL3.)

The RML Learner in the Plate of Hanoi environment	
Environment:	The "special" balls and plates.
Percepts:	ON(ball,plate/table), SIZE>(ballx,bally), INHAND(ball).
Actions:	PICK(ball,plate/table), PUT(ball,plate/table).
Primitives:	$\wedge$, $\neg$ and $\exists$.
Goals:	{ON(BALL1,PLATE3),ON(BALL2,PLATE3),ON(BALL3,PLATE3)}.

An example of prediction rule, Rule0, learned by RML may look like the following:

Condition: INHAND(*ballx*)$\wedge$ $\neg$ON(*ballx plate/table*)
Action: PUT(*ballx plate/table*) **[Rule0]**
Prediction: ON(*ballx plate/table*)$\wedge$ $\neg$INHAND(*ballx*)

It says that the action PUT can put a ball in hand onto a plate or the table if the ball is not already there. The reader can see that this rule, although legitimate, is clearly incomplete.

Note that in this environment, other learners can be defined too. For example, I can define a learner to be a robot with a hand and an eye, with its percepts as features of objects (location, size, color, shape) and its actions as the movement of its hand and the rotation of its arm. In this paper, however, I will concentrate on the RML learner due to space limitations. Interested readers may find more information on other types of learners in [Shen, 1989]. In general, the more primitive a learner's percepts and actions are, the more general the learner is. For example, the actions of the robot learner involve only its own body parts, it can learn from any environment in which it can move its arm and hand. While the actions of the RML learner involves objects in this particular environment (such as balls and plates), it can only learn from environments where balls are moved among plates and table. Naturally, the more general a learner is, the more difficult its learning task is.

4 Integration via Prediction Sequences

As I mentioned earlier, the main objective of learning from the environment is to integrate model application with model construction. However, model application is a diverse notion in itself. A model can be used

to solve problems (planning), to gather new information (exploration), or to find out why the model is incorrect (experimentation).

Regardless of the superficial differences, planning, exploration, and experimentation are all sequences of actions with predictions. Recall that a prediction, relative to a condition and an action, is a statement that describes the expected observation from the result state. In learning from the environment, predictions can be sequenced as follows:

$$\langle S_0, a_1, P_1, a_2, P_2, \cdots, a_n, P_n \rangle,$$

where S_0 is the observation from the current state, a_i is an action, and P_i is a prediction.

As the actions in this sequence are executed, observations from the environmental states, S_i, are perceived in a sequence: $S_0, a_1, S_1, a_2, S_2, \cdots, a_n, S_n$. A prediction failure occurs as soon as a prediction P_i is not satisfied in the correspondent observation S_i. Notice that a prediction failure is different from a failure/success for achieving a goal. With respect to a goal, an action can be successful by accident (i.e., the result state is the goal) but the prediction still fails (i.e., the result state does not satisfy the prediction). Thus, learning from prediction failures means both learning from failures and learning from successes.

With a prediction sequence so defined, planning, exploration and experimentation are all special cases:

- A *plan* is a prediction sequence whose accumulated prediction[1], $\sqcup_{i=1}^{n} P_i$, satisfies the goal G (recall that a goal is a set of percepts). A plan may or may not produce prediction failures. A plan succeeds only if no prediction failure occurs.
- An *exploration* is a prediction sequence in which some of the predictions are "false." Since a false prediction cannot be satisfied by any observation, an exploration is guaranteed to produce prediction failures, thus improve the model. The motivation of this is that when you don't know what might happen after an action (i.e., when you explore an action), whatever happens will be valuable information for improving the model.
- An *experiment* is a prediction sequence whose final prediction is expected to fail. This is an effective way to seek counterexamples (prediction failures) when the current model is known to have particular errors. For example, if the model says $(P_{n-1}, a) \rightarrow P_n$ but there is a strong reason to believe this is false, then experiments of $\langle \cdots, P_{n-1}, a, P_n \rangle$ will be very useful for revising the model because it may produce a prediction failure.

```
Integration Loop:
  1. Generate a prediction sequence using the current model;
  2. Execute actions and perceive results in the environment;
  3. If there is a prediction failure,
        call model construction to expand or revise the model.
```

Figure 2: The Integration Loop

The integration of model application and model construction is accomplished by the loop in Figure 2. We can see that the learner's default activity is model application (i.e., using the model to construct prediction sequences), and it switches to model construction only when there is a prediction failure. When the model is revised, the learner starts model application again. This loop goes on until a successful plan for the goals is constructed and executed.

It should be clear that learning from the environment is essentially rule induction from examples in the context of problem solving. The examples are the observed consequences of actions in the environment (i.e., the triples of (state action state)). The quality of the learned rules is then measured by their utility for problem solving. From this point of view, the integration loop is a special case of the theory of unifying problem solving with rule induction, first proposed in [Simon and Lea, 1974].

[1] The operator $\sqcup$ is a set union with undos. For example, if $P_i = \{\neg b\}$ and $P_{i+1} = \{b\}$, then $P_i \sqcup P_{i+1} = \{b\}$ because b undoes $\neg b$.

Figure 3: The LIVE Architecture

The prediction sequence generator is responsible for generating a prediction sequence, for both the model and the goals. It has three submodules: the Planner, the Explorer, and the Experimenter. The submodules are coordinated as follows. The Planner is called first to construct a solution for the goals using the current model. If it fails, then either the Explorer or the Experimenter will be called. The Explorer is called if the failure of plan construction is due to the lack of prediction rules in the model (i.e., there is, given the current rule set, no chaining path between the current state and the goals.) The Experimenter is called when prediction rules cause errors during planning. Two common types of planning errors are *regression deadlock* or a *regression loop*. A regression deadlock means that subgoals proposed by rules conflict with each other no matter how they are ordered. A regression loop means that the same subgoal is repeatedly proposed forever. How the Explorer and the Experimenter generate their prediction sequences is described in later sections.

The executor/perceiver module executes the generated prediction sequence in the environment. This module compares the result of each action with the corresponding prediction. If there is a prediction failure, it calls the builder/reviser module. At this point, the executor/perceiver will normally relinquish its control and wait for a new prediction sequence. But, if the current sequence is an exploration, it will resume execution after the model is revised. It relinquishes the control after the whole exploration sequence is completed.

The builder/reviser module has two submodules. If the failed prediction is equal to "false" (i.e., the action is an exploration), the builder submodule will be called to create a new prediction rule. Otherwise, the reviser submodule will be called to revise the rule that made the incorrect prediction. If necessary, the reviser may also call its submodule, the term constructor, to define new relations and terms.

To illustrate the interrelation of all these modules, let us go through an example to see how the RML learner learns a model and solves a problem in the Plate of Hanoi environment (defined in Sec-

tion 3). Suppose the observation from the initial state is ON(BALL1,PLATE1) ON(BALL2,PLATE1) ON(BALL3,PLATE1), and the given goals are ON(BALL1,PLATE3) ON(BALL2,PLATE3) ON(BALL3 PLATE3). Since RML starts with an empty model, the Planner fails and the Explorer is called. An exploration {PICK(BALL1 PLATE1), False; PUT(BALL1 PLATE2), False} is generated (see Section 6). Two prediction failures will occur in executing this sequence, out of which two rules, Rule0 (listed in Section 3) and Rule1 (see Section 6), will be created by the Rule Builder.

Since the model now contains two rules, the Planner generates a prediction sequence intended to put balls on PLATE3 one by one, in an order that is arbitrarily chosen. As a result, BALL1 is successfully put on PLATE3, but not BALL2. After PUT(BALL2 PLATE3), a prediction failure occurs because BALL2 is ON the TABLE. The Rule Reviser is called to revise Rule0, which made this failed prediction. This revision causes Rule1 to be replaced by two new rules, Rule2 and Rule4.

In the same fashion, Rule3 and Rule5 are learned because of a prediction failure caused by Rule1 (see Section 7.1). Now the Planner tries again to construct a plan. This time it fails because of a regression deadlock caused by Rule2, which claims that in order to put a ball on a plate, the plate must be empty (see Section 7.2). Thus, the Experimenter is called and an experiment is constructed to try to put a ball on a nonempty plate. The experiment succeeds and produces a prediction failure for Rule4 (the sibling rule of Rule2), which triggers the Rule Reviser to replace Rule2 and Rule4 by Rule6 and Rule8 (see Section 7.1). After that, a new plan is constructed again. This time, the plan is successfully executed without producing any prediction failure. At the end of the plan, all the goals are accomplished and LIVE begins to generate new goals.

In the following sections, I discuss the submodules in detail according to the flow of control. Section 6 describes the Explorer and the Rule Builder, which are responsible for creating new prediction rules. Section 7 describes the Rule Reviser, the Planner, and the Experimenter, which are responsible for revising prediction rules. Finally, Section 8 describes the the term constructor submodule, which is responsible for creating new relations and new terms.

6 Exploration and Rule Creation

Exploration in an unknown environment is a difficult task because the number of possibilities is potentially too large to be exhaustively explored. Here, a partial solution is presented as a set of three heuristics. The general idea is to direct the exploration towards the goals whenever possible.

> *Heuristic 1 (goal-seeking): If an action B is known to change the feature F of objects, then explore B to change the value of F of the learner to be equal (or some other relation) to the value of F of some goal objects.*

For example, if the action *Rotate* is known to change the direction of an arm, then the arm should be rotated to the direction of a goal ball.

> *Heuristic 2 (anomolous behavior resolution): Explore actions that appearently have no effect in the environment.*

For example, PICK has no effect when the ball to be picked is not the smallest on its plate, and PUT has no effect when there is no ball in the hand. Such actions need to be explored until effects are observed. The philosophy behind this heuristic is that all actions ought to have some effect on the environment.

> *Heuristic 3 (curiosity): Once in a while, randomly explore some not-yet-explored actions with random parameters.*

To illustrate these heuristics, suppose the RML is exploring the Plate of Hanoi environment. Heuristics 2 and 3 force RML to repeatedly try PICK and PUT on randomly selected balls and plates until PICK

picks up a ball. After that, Heuristic 2 will force RML to keep exploring Put until the ball in hand is put down on some plate.

The purpose of exploration is to create new prediction rules. This is the task of the Rule Builder. When triggered by a prediction failure in exploration, the Rule Builder creates a new prediction rule by the algorithm in Figure 4.

---

1. Compare the condition observation (before the action) with the result observation (after the action). Let the "vanished" percepts be those that are in the condition observation but not in the result observation, and the "merged" percepts be those that are in the result but not in the condition.

2. Create a new prediction rule:

   Condition: the vanished percepts in conjunction with the negations of the merged percepts;

   Action: the executed action with parameters specified in terms of objects which were acted upon;

   Prediction: the merged percepts in conjunction with the negations of the vanished percepts.

   and generalize all objects in the rule, except those belonging to the learner (such as Hand and Arm), into variables.

Figure 4: Create new predcition rules

Table 1: A prediction failure in exploration

Action: PICK(BALL1 PLATE1)		
Condition Observation	Prediction	Resulting Observation
ON(BALL1 PLATE1)	False	INHAND(BALL1)
ON(BALL2 PLATE1)		ON(BALL2 PLATE1)
ON(BALL3 PLATE1)		ON(BALL3 PLATE1)
SIZE>(BALL3 BALL2)		SIZE>(BALL3 BALL2)
SIZE>(BALL3 BALL1)		SIZE>(BALL3 BALL1)
SIZE>(BALL2 BALL1)		SIZE>(BALL2 BALL1)

As an example, suppose RML meets the prediction failure specified in Table 1. (The prediction "False" is not satisfied by the result observation.) It is easy to see that the vanished percept is ON(BALL1 PLATE1) and the emerged percept is INHAND(BALL1). A new rule is created as follows:

Condition: ON($ballx\ plate$) $\wedge$ $\neg$INHAND($ballx$)
Action: PICK($ballx\ plate$) **[Rule1]**
Prediction:INHAND($ballx$) $\wedge$ $\neg$ON($ballx\ plate$)

In real world situations, a single action may cause many changes, and finding the relevant changes is essentially a qualification problem [Ginsburg and Smith, 1988]. Here, LIVE takes a pragmatic approach called *incremental enlargement* [Shen, 1989]. The idea is to focus changes that are related to the learner and use as few changes as possible in the new rule. Technically, the condition observation and the result observation are viewed as graphs with nodes being objects and links being relations. A comparison of the two graphs begins from a small set of nodes that correspond to the learner itself (e.g., the arm, the hand, and the body) or objects that are mentioned in the action. For example, when the action is $Rotate(Arm, 30°)$, the comparison starts from the object ARM. When the action is PICK(BALL1 PLATE1), the comparison starts from BALL1 and PLATE1. If changes are found in these small subgraphs, the search will terminate. Otherwise, the subgraphs are enlarged incrementally through links until changes are found. Furthermore, not all changes so found are used in the new rule. One one of each type of change is used by LIVE. For example, if a robot's hand is above a stack of disks Disk_1, Disk_2, ..., Disk_i, then moving the hand away will cause all the relations ABOVE(HAND,Disk_i) to "vanish." In this case, only one (any one) of them will be used in the new rule.

7 Model Revision during Model Application

The newly created rules, like the one produced in the last section, are clearly over-general and incomplete. However, they serve as a springboard for problem solving and further learning. Because of their generality, LIVE has chances to attempt goals, to meet prediction failures, and hence to increase its knowledge about the environment. This section describes how these steps are accomplished by the Rule Reviser, the Planner and the Experimenter.

7.1 Revising Rules by Complementary Discrimination

Rule revision is the task of the Rule Reviser. It is triggered by a prediction failure. The rule that made the prediction is called the *faulty rule*. The reviser uses Complementary Discrimination Learning [Shen, 1990] to explain prediction failures and revise the rules. The prediction rules in the model are organized as pairs of sibling rules. Two rules are siblings if they have the same action but have complementary conditions and different predictions. The purpose of rule revision is to adjust the boundary between sibling rules' conditions. In this paper, we assume each rule has at most one sibling. Extensions to multiple predictions can be found in [Shen, 1992].

Previous applications of each rule are remembered. Each application contains a rule index, a condition state observation (to which the rule is applied), and a set of variable bindings. When a prediction failure indicates a faulty rule, LIVE will search for the rule's previous, successful applications, and find the *difference* between the condition states now and then. The difference is used either to split the rule into two new rules (if the faulty rule has no siblings), or to revise the rule and its sibling together. Notice that sibling rules help each other in future development. If a rule's condition is too general (i.e., causes a prediction failure), then it will be specialized and its sibling condition is generalized. If a rule's condition is too specific (its sibling must be too general), then it will be generalized when its sibling rule causes a prediction failure. The algorithm for revising prediction rules is given in Figure 5.

To illustrate the algorithm, suppose RML meets the following prediction failure:

Action: PICK(BALL2 PLATE1)		
Rule: Rule1		
Bindings: (*ballx*=BALL2) (*plate*=PLATE1)		
Condition Observation	Prediction	Resulting Observation
INHAND(BALL1)	–	INHAND(BALL1)
ON(BALL2 PLATE1)	INHAND(BALL2)	ON(BALL2 PLATE1)
ON(BALL3 PLATE1)	–	ON(BALL3 PLATE1)
SIZE>(BALL3 BALL2)	–	SIZE>(BALL3 BALL2)
SIZE>(BALL3 BALL1)	–	SIZE>(BALL3 BALL1)
SIZE>(BALL2 BALL1)	–	SIZE>(BALL2 BALL1)

To explain this prediction failure, the rule's last application, which was illustrated in Table 1 with the bindings *ballx*=BALL1 and *plate*=PLATE1, is fetched. After comparing these two applications, RML finds the difference to be ¬INHAND(BALL1) and generalizes it to ¬INHAND(*bally*). Based on the difference, Rule1 is then split into the following two sibling rules:

Index: **Rule3**
Condition: ON(*ballx plate*) ∧ ¬INHAND(*ballx*) ∧ ¬INHAND(*bally*)
Action: (Pick *ballx plate*)
Prediction: INHAND(*ballx*) ∧ ¬ON(*ballx plate*)
Sibling: Rule5

Index: **Rule5**
Condition: ON(*ballx plate*) ∧ ¬INHAND(*ballx*)∧INHAND(*bally*)
Action: (Pick *ballx plate*)
Prediction: ON(*ballx plate*)
Sibling: Rule3

To illustrate how sibling rules are revised together, consider the following two rules:

1. Explain the prediction failure by finding the relation differences between the faulty rule's previous success and its current failure (using the incremental enlargement heuristic described before).

2. If no difference is found, call Search-Rel-Terms (see Section 8) to find the "hidden" differences.

3. Let the result be a set $(D_1\ D_2\ \ldots\ D_j)$, where D_j is either a conjunction of relations or a negation of conjunctive relations that is true in the successful state but false in the surprising state.

4. If the faulty rule m does not have a sibling (this means the rule has not failed since its creation), then it is split into a pair of sibling rules as follows (the condition of rule m will be called the I-Condition for the purpose of explanation):

		Index m'
		I-Condition $\wedge D_1 \wedge D_2 \ldots \wedge D_j$
	$\Longrightarrow$	Action
Index m		Prediction
I-Condition		Sibling n
Action		
Prediction		Index n
Sibling ()		I-Condition $\wedge\neg(D_1 \wedge D_2 \ldots \wedge D_j)$
	$\Longrightarrow$	Action
		The unexpected effects
		Sibling m'

5. If the faulty rule m already has a sibling n, then modify both of them as below. Notice that sibling rule n's non-I-conditions $O_1, \ldots, O_i$ are replaced by $\neg(C_1 \ldots \wedge C_i \wedge D_1 \ldots \wedge D_j)$. This guarantees the result rules are still siblings.

Index m		Index m
I-Condition $\wedge C_1 \ldots \wedge C_i$		I-Condition $\wedge C_1 \ldots \wedge C_i \wedge D_1 \ldots \wedge D_j$
Action	$\Longrightarrow$	Action
Prediction		Prediction
Sibling n		Sibling n
Index n		Index n
I-Condition $\wedge O_1 \ldots \wedge O_i$		I-Condition $\wedge\neg(C_1 \ldots \wedge C_i \wedge D_1 \ldots \wedge D_j)$
Action	$\Longrightarrow$	Action
Prediction		Prediction
Sibling m		Sibling m

Figure 5: The algorithm for revising prediction rules

Index: **Rule2**
Condition: INHAND($ballx$) $\wedge$ $\neg$ON($ballx\ plate$) $\wedge$ $\neg$ON($bally\ plate$)
Action: PUT($ballx\ plate$)
Prediction: ON($ballx\ plate$) $\wedge$ $\neg$INHAND($ballx$)
Sibling: Rule4

Index: **Rule4**
Condition: INHAND($ballx$) $\wedge$ $\neg$ON($ballx\ plate$)$\wedge$ON($bally\ plate$)
Action: PUT($ballx\ plate$)
Prediction: ON($ballx$ TABLE) $\wedge$ $\neg$INHAND($ballx$)
Sibling: Rule2

Suppose RML now applies the action PUT(BALL2 PLATE1) in the state whose observation is: INHAND(BALL2) ON(BALL3 PLATE1) ON(BALL1 PLATE2) SIZE>(BALL3 BALL1) SIZE>(BALL2 BALL1) SIZE>(BALL3 BALL2). The prediction of the action is ON(BALL2,TABLE), which is made by Rule4 with variable bindings: (*ballx*=BALL2, *plate*=PLATE1, *bally*=BALL3). After executing the action, RML is surprised because BALL2 is now on PLATE1. To explain this prediction failure, Rule4's last application is fetched, which, in this case, is in an identical state observation but with a different set of bindings: (*ballx*=BALL2, *plate*=PLATE2, *bally*=BALL1). Comparing these two applications, RML finds the difference to be (SIZE> *ballx bally*). Based on the difference, Rule4 is then revised into Rule6, and its sibling Rule2 is revised into Rule8. Notice that the first two condition elements of Rule4 are not changed because they are I-conditions. Rule6's condition is made by putting the difference in conjunction with Rule4's condition. Rule8's condition is made by putting the I-Conditions in conjunction with the negation of non-I-Conditions of Rule6.

Index: **Rule6**
Condition: INHAND($ballx$) $\wedge$ $\neg$ON($ballx\ \ plate$)$\wedge$ON($bally\ \ plate$)$\wedge$SIZE>($ballx\ \ bally$)
Action: PUT($ballx\ \ plate$)
Prediction: ON($ballx$ TABLE) $\wedge$ $\neg$INHAND($ballx$)
Sibling: Rule8

Index: **Rule8**
Condition: INHAND($ballx$) $\wedge$ $\neg$ON($ballx$ $plate$) $\wedge$ $\neg$[ON($bally$ $plate$)$\wedge$SIZE>($ballx$ $bally$)]
Action: PUT($ballx$ $plate$)
Prediction: ON($ballx$ $plate$) $\wedge$ $\neg$INHAND($ballx$)
Sibling: Rule6

7.2 Generating Plans and Experiments

Plans and experiments are generated by the Planner and the Experimenter respectively. Since the Planner uses the standard goal regression method [Waldinger, 1977; Genesereth and Nilsson, 1987], I will focus our discussion on the Experimenter.

As I mentioned in Section 5, the Experimenter is called when the Planner meets either a regression deadlock or a regression loop. This indicates that the rule that causes such problems is erroneous (or the problem is unsolvable, in which case LIVE will fail). The objective of the Experimenter is to design a situation in which a prediction failure may occur in such a way that the faulty rule will be revised.

An error at planning time does not provide enough information about how to fix a faulty rule. It only indicates that the condition of the rule is too restrictive. For example, Rule2 claims that in order to put a ball on a plate, the plate must be empty ($\neg$ON($bally$ $plate$)). This causes a regression deadlock because no matter how the subgoals (to put each ball on the same plate) are arranged, they always conflict with each other. Since it has been observed before that more than one ball can be on the same plate, there must exist situations in which Rule2's action, currently forbidden to apply, can indeed realize Rule2's prediction.

The purpose of an experiment is to find a situation in which the condition of a faulty rule F can be proven to be too restrictive. Moreover, such a "proof" should be a prediction failure that "involves" F, possibly through another rule R, so that F can be revised using complementary discrimination learning. Thus, an experiment must specify a situation S and an existing rule R such that (1) R can be applied to S but F cannot, (2) R's action is the same as F's, and (3) R's prediction is complementary to F's so that whenever F's prediction is realized where will be a corresponding prediction failure for R.

Fortunately, the relationship between F and R is exactly the "sibling" relation of the prediction rules. For example, for sibling rules Rule2 and Rule4, their respecitve predictions and conditions are complements of each other and their actions are the same. This is not a coincidence. It is an advantage of learning by complementary discrimination. If a rule is known to be overly specific, then its sibling rule must be overly general. Since they are siblings, the only thing the learner needs to do is to find a chance to specialize the sibling rule so that the faulty rule will be generalized as a by-product.

Therefore, an experiment is nothing but an instantiation of the faulty rule's sibling. The sibling's condition specifies a set of states S_e as the experiment's setting; its action a_e specifies the action to be performed in the experiment; and its prediction P_e specifies the experiment's prediction. Once these are identified, the Experimenter calls the Planner to find a plan $J =< s, a_1, P_1, \cdots, a_i, S_e >$ to reach the experiment's condition states. The concatenation of this plan and the experiment's action and prediction, $< J, a_e, P_e >$, is the final prediction sequence.

An experiment is successful if the outcome of its action produces a prediction failure. Otherwise, new experiments must be proposed by instantiating the sibling rule differently. In general, experiments are instantiated from a faulty rule's sibling by alternately assigning different objects to the parameters of the action. For instance, if the action in an experiment is PICK($ball$, $plate$), then different balls will be assigned to the variable $ball$, and different plates to $plate$. When instantiating a rule in an experiment, the heuristic of trying to keep the required "preparing sequence," J, as short as possible is taken into consideration.

Back to our example, Rule4 can be instantiated to construct the following experiment:

Experiment: Instantiated from Rule4
Condition: INHAND(BALL2)$\wedge\neg$ON(BALL2 PLATE1)$\wedge$ON(BALL3 PLATE1)
Action: PUT(BALL2 PLATE1)
Prediction: ON(BALL2 TABLE)$\wedge\neg$INHAND(BALL2)
Bindings: (($ballx$. BALL2) ($plate$. PLATE1) ($bally$. BALL3))

This experiment is preferred to others because its condition can be easily established from the current

state, i.e., simply picking up BALL2 from TABLE, and the action has not been previously performed. The experiment will be a success because it will indeed cause a prediction failure. Based on the prediction failure, Rule2 and Rule4 are revised into Rule6 and Rule8 as described in Section 7.1.

8 Constructing New Relations and Terms

No matter how many percepts a learner is capable of, there are always entities in the environment that it cannot perceive. New relations and terms must be constructed when such invisible entities are essential for building a correct model of the environment.

New relations and terms are necessary when the learner finds that executing the "same" action in the "same" state produces different consequences. In LIVE, this is when the Rule Reviser (see Figure 5) fails to find any differences between a successful state and a failed state. Consider, for example, a new learner RML2 modified from RML as follows. RML2 cannot perceive the relation SIZE>. Instead, it can perceive "size" of objects (e.g., size(BALL3)=3). To RML2, the two states listed at the end of Section 7.1 are perceived as follows:[2]

S_0 = {BALL1, BALL2, BALL3, PLATE1, PLATE2, PLATE3, INHAND($ballx$), ON($bally\ plate$), ON(BALL1 PLATE2), size($bally$)=3, size($ballx$)=2, size(BALL1)=1}, where $ballx$=BALL2, $plate$=PLATE1, $bally$=BALL3.

T_0 = {BALL1, BALL2, BALL3, PLATE1, PLATE2, PLATE3, INHAND($ballx$), ON(BALL3 PLATE1), ON($bally\ plate$), size(BALL3)=3, size($ballx$)=2, size($bally$)=1}, where $ballx$=BALL2, $plate$=PLATE2, $bally$=BALL1.

Since SIZE> is not perceivable, RML2 cannot find any relation difference between S_0 and T_0 according to the incremental enlargement heuristic (see Section 6).

Given two such states, where no differences are found according to the learner's percepts, the search for new relations and terms is accomplished as follows. The learner applies its mental relations and functions to the objects in S_0 and T_0 to see if any new relations or terms can be defined to distinguish S_0 and T_0. For example, suppose RML2 has mental relations ">" and "=", then applying them to S_0 and T_0 will result the following expanded states:[3]

S_0 = {BALL1, BALL2, BALL3, PLATE1, PLATE2, PLATE3, INHAND($ballx$), ON($bally\ plate$), ON(BALL1 PLATE2), size($bally$)=3, size($ballx$)=2, size(BALL1)=1, >(size($bally$) size($ballx$)) >(size($bally$) size(BALL1)), >(size($ballx$) size(BALL1))} where $ballx$=BALL2, $plate$=PLATE1, $bally$=BALL3.

T_0 = {BALL1, BALL2, BALL3, PLATE1, PLATE2, PLATE3, INHAND($ballx$), ON(BALL3 PLATE1), ON($bally\ plate$), size(BALL3)=3, size($ballx$)=2, size($bally$)=1, >(size(BALL3) size($ballx$)), >(size(BALL3) size($bally$)), >(size($ballx$) size($bally$))} where $ballx$=BALL2, $plate$=PLATE2, $bally$=BALL1.

From these two enlarged states, the difference can be easily found. The relation >(size($ballx$) size($bally$)) is true in T_0 but false in S_0. Since it is the relation >(size(objx) size(objy)) makes the difference, a new relation REL(objx objy) is defined as >(size(objx) size(objy)), and the difference REL($ballx\ bally$) is then returned to the Rule Reviser (see Section 7.1). Subsequently, RML2 will always apply the relation REL to objects. In some sense, its perceptual ability is improved.

In general, mental functions are applied to features of objects to define new terms, and then mental relations are applied on these new terms to form new relations. For example, the term "torque" is defined as a function $\times$ on visible features "distance" and "weight", i.e., torque(x)=distance(x)$\times$weight(x)). Then the relation "torque>(x y)" is defined as >(torque(x),torque(y)). The process of defining new relations and terms is essentially a process of search. The learner systematically selects mental functions and relations and applies them to objects until such applications result in differences between states. At present, LIVE uses only a breadth-first strategy for this search.

The procedure for searching for new relations and terms is listed in Figure 6. Relating this procedure to the example of RML2 above, the relation r specified in the algorithm is ">", the function f is identity, and feature functions are $p_1 = size$ and $p_2 = size$. Related to the example of torque>, the relation r is

[2] Objects in relations are replaced by variables if they are bound.

[3] All the plates are of the same size and they are larger then the balls. But for simplicity these relations are not included in the description.

Procedure Search-Rel-Terms(S,T):
Let Difference=Search-Action-Independent-Rel-Terms(S,T),
return Difference to the Rule Reviser (Figure 5).

Procedure Search-Action-Independent-Rel-Terms(S,T):
1. Select a relation r (of arity n) and a function f (of arity m);
2. Select m unary functions $p_1, p_2, \cdots, p_m$ whose domains are features;
3. If there exists n objects $o_1, o_2, \cdots, o_n$ such that
$r[v_1^S, v_2^S, \cdots, v_n^S] \neq r[v_1^T, v_2^T, \cdots, v_n^T]$
where $v_i^S = f[p_1(o_i), p_2(o_i), \cdots, p_m(o_i)]$ in state S,
and $v_i^T = f[p_1(o_i), p_2(o_i), \cdots, p_m(o_i)]$ in state T;
then return $r[v_1^S, v_2^S, \cdots, v_n^S]$.
4. If all the sections are exhausted, then return FAIL, else goto 1.

Figure 6: Searching for action-independent relations and terms

">," the function f is $\times$, the feature functions are $p_1 = distance$ and $p_2 = weight$, and the new term is torque(x)≡distance(x)×weight(x)). Notice that all the terms defined in this fashion are *action-independent*, for they can be defined without actions.

Discovering action-independent terms is not the end of the story in learning from the environment. When all the mental relations and functions cannot help the learner to find any difference between states (this is possible when the learner does not have enough percepts to start with), then the procedure Search-Action-Independent-Rel-Terms(S_0, T_0) will return FAIL. For example, suppose RML2 is further restricted to become RML3. The learner RML3 is the same as RML2 except it cannot see the size of objects. To RML3, balls are of the same size and the states S_0 and T_0 are perceived as:

S_0 = {BALL1, BALL2, BALL3, PLATE1, PLATE2, PLATE3, INHAND($ballx$), ON($bally\ plate$), ON(BALL1 PLATE2)},
where $ballx$=BALL2, $plate$=PLATE1, $bally$=BALL3.

T_0 = {BALL1, BALL2, BALL3, PLATE1, PLATE2, PLATE3, INHAND($ballx$), ON(BALL3 PLATE1), ON($bally\ plate$)},
where $ballx$=BALL2, $plate$=PLATE2, $bally$=BALL1.

Since RML3 cannot perceive any features of objects, applying the mental functions and relations will not result any difference between these two states.

When situations like this arise, we say that the environment has hidden features that are *action-dependent*. The learner must search back into the history of S_0 and T_0 to find the differences there. New terms must be defined in terms of not only percepts but also actions. These terms are defined to carry the difference to the present so that the learner can predict the future. In the case of RML3, what must be discovered are the following conditions for the actions PICK and PUT:

PICKable(x,p)$_{(t)}$ ← ON(x,p)$_{(t)}$ ∧ ¬INHAND(z)$_{(t)}$ ∧ ¬[ON(y,p)$_{(t)}$ ∧ [y was put on p more recently than x]]

[y was put on p more recently than x] ← ∀(n)∃(n')[PUT(x,p)$_{(t-n)}$ ∧ PUT(y,p)$_{(t-n')}$ ∧ (n'<n)]

PUTable(x,p)$_{(t)}$ ← INHAND(x)$_{(t)}$ ∧ ¬[ON(y,p)$_{(t)}$ ∧ [y was pickable from p' when ON(x,p')]]

[y was pickable from p' when ON(x,p')] ← ∃(n)[ON(y,p')$_{(t-n)}$ ∧ ON(x,p')$_{(t-n)}$ ∧ PICK(x,p')$_{(t-n)}$ ∧ INHAND(y)$_{(t-n+1)}$]

The predicate PICKable$(x,p)_t$ says that x can be picked up from p at time t, if x is on p, the hand is empty, and there is no y on p such that y was put on p more recently than x. The predicate PUTable$(x,p)_t$ says that x can be put on p at time t, if x is in hand and there is no y on p such that y was pickable from a plate p' at a previous point in time when x was on p'. From these definitions, one can see that action-dependent terms are defined in terms of both actions and percepts. Moreover, their values may depend on the values in previous states and can be changed by actions. (In this sense, they are also known as "recursive theoretical terms," see [Shen and Simon, 1990].)

To discover action-dependent terms, LIVE searches back into the history of S_0 and T_0 when Search-Action-Independent-Rel-Terms(S_0, T_0) returns FAIL. It then identifies two "relevant" historical sequences of S_0 and T_0 and finds the difference between them. This difference then becomes the definition of a new predicate that is returned to the Rule Reviser.

t_n	History	S_0's Ancestors x=2,p=2,y=1		T_0's Ancestors x=2,p=1,y=3	
t_1	ON(1 1)ON(2 1)ON(3 1) PICK(1 1)	S_{-3}	ON(y 1)ON(x 1) PICK(y 1)		
t_2	INHAND(1)ON(2 1)ON(3 1) PUT(1 2)	S_{-2}	INHAND(y)ON(x 1) PUT(y p)		
t_3	ON(1 2)ON(2 1)ON(3 1) PICK(2 1)	S_{-1}	ON(y p)ON(x 1) PICK(x 1)	T_{-3}	ON(y p)ON(x p) PICK(x p)
t_4	INHAND(2)ON(1 2)ON(3 1) PUT(2 2)	S_0	ON(y p)INHAND(x) PUT(x p)	T_{-2}	ON(y p)INHAND(x) PUT(x 2)
t_5	ON(1 2)ON(2 Tbl)ON(3 1) PICK(2 Tbl)			T_{-1}	ON(y p)ON(x Tbl) PICK(x Tbl)
t_6	INHAND(2)ON(1 2)ON(3 1) PUT(2 1)			T_0	ON(y p)INHAND(x) PUT(x p)

Table 2: Search for historical differences

In the current example, the history of LIVE is listed in the first column of Table 2. As described in Section 7.1, each historical item contains a rule index (omitted in this column), a state observation to which the rule's action was applied, and a set of variable bindings (also omitted in this column). As we can see, the item t_6 is what we call T_0. It is the current state in which the application of Rule4 results in a prediction failure. The item t_4 is what we call S_0 for it was the last successful application of Rule4 in the history. The second and third columns in Table 2 are the ancestor states of S_0 and T_0 respectively. They are "views" of the items in the first column of the same row through the varaible bindings of S_0 and T_0 (listed in the top rows of these two columns). Such views are created by copying the corresponding item from the first column, replacing the balls and plates with the variables according to the bindings, then deleting those elements that have no variables. For example, S_{-1} is a view of t_3, according to the bindings (x=2 p=2 y=1), in which ON(1 2) is replaced by ON(y p), ON(2 1) is replaced by ON(x 1), ON(3 1) is deleted, and PICK(2 1) is replaced by PICK(x 1). Likewise, T_{-3} is also a view of t_3, but according to the bindings (x=2 p=1 y=3).

This particular way of "viewing" ancestor states from a particular rule application is a general heuristic, and a very powerful one, to identify the relevant histories. It focuses the learner's attention to the history of the objects that are related to the current action. When comparing two histories, LIVE identifies the relevant historical sequences, say from time t_{0-u} to t_{0-v}, where $(0-u)<(0-v)$ and they are relative to S_0 and T_0), and find the difference there.

The time t_{0-u} is identified by seaching back from T_0 and S_0 to the first states, say $T_{t_{0-u}}$ and $S_{t_{0-u}}$, where the objects (now represented as varaibles) that do not have apparent relations in T_0 and S_0 had some visible relations. In our current example, searching back from T_0 in this way leads to T_{-3} because the object x and y, which do not related in T_0, were both on the plate p. Likewise, searching from S_0 in the same way leads to $S{-}3$.

The time t_{0-v} is identified by seaching back from T_0 and S_0 to the first states, say $T_{t_{0-v}}$ and $S_{t_{0-v}}$, where the difference between $T_{t_{0-v}}$ and $S_{t_{0-v}}$ first become visible. In our example, this leads to T_{-2} and S_{-2} where INHAND(x) was true in S_{-2} while INHAND(x) was true in T_{-2}.

After $(S_{0-u} \cdots S_{0-v})$ and $(T_{0-u} \cdots T_{0-v})$ are identified, the *difference* between these two history sequences are those relations and actions (except the action at time $0-v$) that appear in $(S_{0-u} \cdots S_{0-v})$ but not in $(T_{0-u} \cdots T_{0-v})$. In our current example, this difference is [ON(y,1)$_{(0-3)}$ $\wedge$ ON(x,1)$_{(0-3)}$ $\wedge$ PICK(y,1)$_{(0-3)}$ $\wedge$INHAND(y)$_{(0-3+1)}$ $\wedge$ ON(x,1)$_{(0-3+1)}$]. Generalizing this difference, say plate 1 to p', time 0 to n, and time -3 to -n, we have the definition of a new predicate: [ON(y,p')$_{(t-n)}$ $\wedge$ ON(x,p')$_{(t-n)}$ $\wedge$ PICK(y,p')$_{(t-n)}$ $\wedge$INHAND(y)$_{(t-n+1)}$ $\wedge$ ON(x,p')$_{(t-n+1)}$], which is equivalent to the predicate "y was pickable from p' when ON(x,p')" as we described in PUTable. This new predicate is then returned to the Rule Reviser in Figure 5 and a correct rule for PUT will be built.

We can modify the procedure Search-Rel-Terms in Figure 6 to search for both action-independent and action-dependent terms. The new version of Search-Rel-Terms is illustrated in Figure 7. When the procedure Search-Action-Independent-Rel-Terms(S_0, T_0) returns FAIL, this new procedure will search into the history of S_0 and T_0, find the difference there, and return the difference in terms of some newly defined action-dependent predicates.

```
Procedure Search-Rel-Terms(S, T):
  Let Difference= Search-Action-Independent-Rel-Terms(S, T),
  If Difference≠FAIL, then return Difference to the Rule Reviser (Figure 5), else
    Identify the revelant history of S_0 and T_0,
    Define new predicates based on the difference between the identified history,
    Return the difference so found to the Rule Reviser.
```

Figure 7: Search for both action-independent and action-dependent terms

9 Performance of LIVE

LIVE has been tested in many different domains. These include the Plate of Hanoi, the Balance Beam problem, defined in [Siegler, 1983], the gene discovery experiments [Shen and Simon, 1990; Shen, 1989], the little prince world, defined in [Rivest and Schapire, 1987], the hidden bits register problem, also defined in [Rivest and Schapire, 1987], and randomly generated Moore machines with hidden states. Before I give the results of the experiments in the first two domains in detail, let me briefly describe the gene discovery experiments.

The gene discovery experiments are formalized as an environment in which LIVE can breed garden peas that have different colors. This is inspired by Mendel's experiments [Mendel, 1865] that lead him to discover genes. It is a case where the importance of action-depedent terms is illustrated naturally. In this environment, LIVE has a single action Breed, and can observe only the colors of peas. Its mental language includes relations such as $=$ and $<$ and functions such as max (maximum), min (minimum), and $EvenDistr$ ($EvenDistr(uv, xy) = (ux, uy, vx, vy)$.) LIVE's task is to predict the colors of offspring when two peas are bred. To make the task feasible for LIVE, the environment is simplified so that two parent peas produce four and only four offspring and their genes are evenly distributed into their children. LIVE is given a fixed sequence of actions instead of free to choose which peas to breed. In this environment, LIVE must define action-depedent terms because genes are not visible and two pairs of green peas may look exactly the same yet they produce offspring with different colors. The action-depedent terms are defined by searching back to the ancestors of peas and find the difference there. In this environment, LIVE successfully discovers the hidden genes of peas and incorporates them into the prediction rules. Interested readers may see [Shen and Simon, 1990; Shen, 1989] for details.

9.1 The RML Learner and the Plate of Hanoi

In the Plate of Hanoi environment, I have tested the RML learner by giving it (1) different goals, (2) different exploration plans, and (3) different numbers of balls. LIVE's performance is sensitive to the difficulty of the goals and to the order in which the goals are expressed. When the same goals are given but expressed in different orders, LIVE learns the same set of rules but the time spent is inversely proportional to the correctness of the order of the goals. The correct order of goals, in this particular environment, means that larger balls are put on the goal plate before smaller ones. Interestingly, LIVE spends less time when the goals are expressed in an incorrect order than when they are in the correct order. This is because the goals in an incorrect order force LIVE to meet prediction failures at an earlier stage of problem solving and thus correct rules are learned before wasting too much time on attempting to solve the goals with a set of bad rules. In other words, LIVE prefers to meet prediction failures as early as possible. This is consistent with the fact that learning correct rules is more important than making superficial progress in problem solving.

LIVE's performance is also sensitive to the difficulty of the goals. On the one hand, if the goals are difficult enough to achieve (i.e., they require the learner to know all the rules), LIVE will learn a complete set of rules to solve the Plate of Hanoi. This may require a longer time since LIVE makes more mistakes and designs more experiments. On the other hand, if the goals are too trivial, LIVE solves the problem quickly but may not have the chance to learn the complete rule set. For example, if a goal can be achieved by just moving the smallest ball to a different plate, then LIVE will be satisfied after learning two rules

that accomplish the goal. This behavior is consistent with the definition of learning from the environment: to construct a model adequate enough for solving the problems (not for mastering the whole environment).

The effect of exploration on LIVE's performance was tested by forcing LIVE to take different exploration actions. This resulted in two interesting observations. First, problem solving will take less time when exploration is more thorough. This is consistent with the earlier observation that learning and discovery enhances problem solving. Second, in some exploration, LIVE learns some rules that are not directly used in problem solving. For example, a rule says that the PUT action will cause no effect if the hand is empty. Although these rules are not used in solving problems, they prevent LIVE from creating similar rules again in future exploration.

Finally, I also tested the RML learner in problems with more than three balls. It is observed that the time for learning the correct rule set is the same regardless of the number of balls (although the total problem solving time increases). This has a simple explanation. The rules that LIVE learned do not depend on how many balls are on plates but how balls relate to each other in actions. The rules learned are general. Once they are learned, they can be used to solve the Plate of Hanoi problem irrespective of its size.

9.2 The Balance Beam Environment

A balance beam is a see-saw with pegs on both sides of the fulcrum where weights can be placed. The task is to predict whether the beam will tip to the left, to the right, or balance. In this environment, LIVE is given the ability to perceive the weights on each side of the beam and the distances from the weights to the center of the beam. To be able to predict correctly, LIVE must discover the invisible concept of "torque" (weight(x)$\times$distance(x)). LIVE is tested in three kinds of experiments: (1) different orders of training instances, (2) different orders of mental relations and functions, and (3) different number of mental relations and functions.

To test how LIVE behaves when training instances are given in different orders, I limited LIVE's mental functions to $\times$ and +, and mental relations to $>$ and $=$. LIVE is given a large number of randomly generated sets of prediction tasks. It is observed that LIVE's discovery of torque in this environment depends on when "informative"" prediction failures happen (i.e., when the procedure Search-Rel-Terms is called). From the experiments I have run, LIVE always discovers the torque concept, although sometimes earlier and sometimes later.

When mental relations and functions are given in different orders, LIVE's performance changes. For example, if the list of relations is $(= >)$ instead of $(> =)$, LIVE will consider the relation (w$\times$d)=(w$\times$d) first. The correct relation (w$\times$d)>(w$\times$d) is discovered later when a prediction fails after (w$\times$d)=(w$\times$d) is defined. Similarly, if the list of functions is $(+ \times)$ instead of $(\times +)$, then the useless relation (w+d)>(w+d) will be defined before (w$\times$d)>(w$\times$d).

Since LIVE's current strategy for searching new terms is brute force, the order of constructs affects LIVE's performance dramatically. For example, LIVE can discover torque when the function and relation lists are $(\times, +, x^y, -, max, min)$ and $(>, =, <, \leq, \geq)$, respectively. However, it failed to do so when the first list is changed to $(max, min, x^y, -, +, \times)$. In this case, LIVE is overwhelmed by too many useless terms, such as max(weight,distance) and $\text{weight}^{distance}$, and it runs out of resources. Obviously, a better search strategy for new terms is essential.

10 Strengths and Weaknesses

There are three principal strengths in this learning framework. First, it defines the problem of learning from the environment as constructing a model of the environment in the context of problem solving. The definition makes a distinction between a learner's innate actions and the consequences of actions with respect to the environment. An approximate model of the environment is extracted from an immense "raw" space determined by the learner's innate physical abilities and prior knowledge. This extraction

process is guided by the information gathered during interactions with the environment.

Second, the framework provides an integrated way to coordinate various activities, such as perception, action, exploration, experimentation, problem solving, learning, discovery, and new term construction. To my knowledge, LIVE is the first implemented system that incorporates so many activities. By viewing discovery as a form of learning from the environment, this integration provides some insights on what activities may be involved in discovery processes and how they interact with each other.

Third, the framework identifies the key to integration to be the notion of prediction sequences and learning by complementary discrimination. Prediction sequences provide a unified view of planing, exploration, and experimentation. They link these activities to learning and discovery via prediction failures. Learning by complementary discrimination provides a very adaptive way for generalizing, or abstracting, a model from the environment and can be extended readily to new term construction.

The framework is still in its early developmental stage. It must overcome several weaknesses and limitations in order to become a general solution for learning from the environment. For example, the framework cannot deal with uncertainty of actions. A single "noisy" prediction failure will cause the model to be revised completely. It cannot react in real time: all its actions are deliberate and may take much time to decide.

The incremental enlargement heuristic is not a general way to find the correct difference between two states. It relies on the "hints" that are implied by the parameters of action. When actions do not have these parameters, the learner needs to determine how objects in the environment are related to actions. This is a question I have not addressed.

Perhaps the most severe limitation of LIVE is that the brute force search method for discovering new terms is too naive and relies on two strong biases. One is that the set of useful mental relations and functions must be given beforehand. The other is that the current way of discovering action-dependent terms is limited to consider only the features and the objects that are related to the current condition and action. For example, in defining genes of pea, LIVE must assume that children's color is determined only by their ancestors' color. While in the real world, features in one state may be determined by different features in its previous states.

Acknowledgments

This research was largely done at the School of Computer Science, Carnegie Mellon University. I thank Herbert Simon for his numerous suggestions and valuable advice throughout the course of this work. I also thank Jaime Carbonell, Tom Mitchell, Chuck Thorp, and many other friends for their warm support and valuable comments. Special thanks to Jan Zytkow and two anonymous reviewers, as well as Mark Derthick, Brian Falkenhainer, Mike Huhns, Ganesh Mani, Avijit Saha, Richard Sutton, and Jim Talley for their help in improving the earlier versions of this paper.

References

Angluin, D. 1978. On the complexity of minimum inference of regular sets. *Information and Control* 39:337–350.

Atkeson, C. G. 1989. Learning arm kinematics and dynamics. *Ann. Rev. Neurosci.* 12.

Barron, A. R. and Cover, T. M. 1991. Minimum complexity density estimation. *IEEE Trans. on Information Theory* 37(4).

Bundy, A.; Silver, B.; and Plummer, D. 1985. An analytical comparison of some rule-learning programs. *Artificial Intelligence* 27.

Drescher, Gary L. 1989. *Made-up Minds: A Constructivist Approach to Artificial Intelligence.* Ph.D. Dissertation, MIT.

Fikes, R.E. and Nilsson, N.J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Friedman, J. H. and Stuetzle, W. 1981. Projection pursuit regression. *Journal of the American Statistical Association* 76(376).

Genesereth, M.R. and Nilsson, N.J. 1987. *Logical Fundations of AI.* Morgan Kaufmann.

Ginsburg, M.L. and Smith, D.E. 1988. Reasoning about action II: The qualification problem. *Artificial Intelligence* 35:311–342.

Goodwin, G. C. and Sin, K. S. 1984. *Adaptive Filtering, Prediction, and Control.* Prentice Hall.

Hayes, J.R. and Simon, H.A. 1974. Understanding written problem instructions. In *Knowledge and Cognition.* Lawrence Erlbaum.

Kaelbling, Leslie P. 1990. *Learning in Embedded Systems.* Ph.D. Dissertation, Stanford university.

Laird, J.E.; Hucha, M.; Yager, E.S.; and Tuck, C.M. 1990. Correcting and extending domain knowledge using outside guidance. In *The Proceedings of the 7th International Machine Learning Conference.*

Mendel, G. 1865. Experiments in plant-hybridization. In Peters, J.A., editor 1865, *Classic Papers in Genetics.* Prentice-Hall.

Mitchell, T.M.; Utgoff, P.E.; and Banerji, R.B. 1983. Learning by experimentation: Acquiring and refining problem-solving heuristics. In *Machine Learning.* Morgan Kaufmann.

Nguyen, D. and Widrow, B. 1989. The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of IJCNN-89*, volume 2.

Rivest, R.L. and Schapire, R.E. 1987. Diversity-based inference of finite automata. In *Proceedings of 28th Foundation of Computer Science.* IEEE.

Rivest, R.L. and Schapire, R.E. 1989. Inference of finite automata using homing sequences. In *Proceedings of 21th Annual ACM Sympposium on Theory of Computing.*

Shen, W.M. and Simon, H.A. 1989. Rule creation and rule learning through environmental exploration. In *Proceedings of 11th IJCAI.* Morgan Kauffman.

Shen, W.M. and Simon, H.A. 1990. Fitness requirements for scientific theories containing recursive theoretical terms. *British Journal for the Philosophy of Science.*

Shen, W.M. 1989. *Learning from the Environment Based on Actions and Percepts.* Ph.D. Dissertation, Carnegie Mellon University.

Shen, W.M. 1990. Complementary discrimination learning: A duality between generalization and discrimination. In *Proceedings of Eighth AAAI*, Boston.

Shen, W.M. 1992. Complementary discrimination learning with decision lists. In *Proceedings of Tenth AAAI*, San Jose.

Shrager, J. 1985. *Instructionless Learning: Discovery of the Mental Model of a Complex Device.* Ph.D. Dissertation, Carnegie-Mellon University.

Siegler, R.S. 1983. How knowledge influences learning. *American Scientist* 71.

Simon, H.A. and Lea, G. 1974. Problem solving and rule induction: A unified view. In *Knowledge and Cognition.* Erlbaum, Hillsdale, N.J.

Stanfill, C. and Waltz, D. 1986. Towards memory-based reasoning. *Communication of ACM* 29:1213–1228.

Waldinger, R. 1977. Achieving several goals simultaneously. In *Machine Intelligence 8.* Academic Press, New York.

Watkins, C. 1989. *Learning from delayed rewards.* Ph.D. Dissertation, Cambridge University.

Whitehead, S.D. and Ballard, D.H. 1991. Learning to perceive and act by trail and error. *Machine Learning* 7(1).

Zytkow, J. M. 1991. Integration of knowledge and method in real-world discovery. *ACM SIGART Bulletin* 2(4).