Learning Models of Environments with Manifest Causal Structure

by

Ruth Bergman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995

Author .... [illegible] ....
Department of Electrical Engineering and Computer Science
May 18, 1995

Certified by .... [illegible] ....
Ronald L. Rivest
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .... [illegible] ....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Learning Models of Environments with Manifest Causal Structure

by

Ruth Bergman

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 1995, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

This thesis examines the problem of an autonomous agent learning a causal world model of its environment. The agent is situated in an environment with manifest causal structure. Environments with manifest causal structure are described and defined. Such environments differ from typical environments in machine learning research in that they are complex while containing almost no hidden state. It is shown that in environments with manifest causal structure learning techniques can be simple and efficient.

The agent learns a world model of its environment in stages. The first stage includes a new rule-learning algorithm which learns specific valid rules about the environment. The rules are predictive as opposed to the prescriptive rules of reinforcement learning research. The rule learning algorithm is proven to converge on a good predictive model in environments with manifest causal structure. The second learning stage includes learning higher level concepts. Two new concept learning algorithms learn by (1) finding correlated perceptions in the environment, and (2) creating general rules. The resulting world model contains rules that are similar to the rules people use to describe the environment.

This thesis uses the Macintosh environment to explore principles of efficient learning in environments with manifest causal structure. In the Macintosh environment the agent observes the screen of a Macintosh computer which contains some windows and buttons. It can click in any object on the screen, and learns from observing the effects of its actions.

In addition this thesis examines the problem of finding a good expert from a sequence of experts. Each expert has an "error rate"; we wish to find an expert with a low error rate. However, each expert's error rate is unknown and can only be estimated by a sequence of experimental trials. Moreover, the distribution of error rates is also unknown. Given a bound on the total number of trials, there is thus a tradeoff between the number of experts examined and the accuracy of estimating their error rates. A new expert-finding algorithm is presented and an upper bound on the expected error rate of the expert is derived.

Thesis Supervisor: Ronald L. Rivest
Title: Professor of Electrical Engineering and Computer Science

להורי,
חוה וחיים

Acknowledgments

I would first like to thank Ron Rivest, without whom both the research and this document would not have come to be. Thanks for years of support and advice, and for hours of discussion. I thank him especially for being supportive of my extra-curricular activities as well as my Computer Science work.

Thanks to my readers for their advice in the last stages of the research and preparation of this document, and for working around my unusual circumstances. I want to thank Patrick Winston for his career counseling and life advice. Thanks to Lynn Stein for her help in relating my work to other work in the field, for corrections on drafts of this thesis, and for being a great role model.

Thanks to Eric Grimson for being much more than an academic advisor.

I thank Jonathan Amsterdam and Sajit Rao for introducing me to MIT and AI Lab life and for being good friends in and out of the office. Thanks to Jonathan for letting me complain — two minutes at a time. Thanks to Sajit for his great optimism about AI and for reminding me why I started graduate school when I needed the reminder.

Thanks to Libby Louie for her help in turning in this document and everything else through the years (especially birthday cakes). Thanks to Margrit Betke and Mona Singh for listening to practice talks, reading drafts of this thesis, and encouraging me every step of the way.

Thanks to my friends at the AI Lab and LCS for the many good times, from discussions over a beer at GSB, to trivial pursuit games, to dancing the night away, to jogging in sub-freezing temperatures. In particular I want to thank Maja Mataric for many good suggestions over the years and for being a good friend. Also thanks to Jose Robles, Ian Horswill, and Paul Viola who helped through the hurdles of my graduate career.

Thanks to all my good friends in the Boston area who made my years as a graduate student unforgettable and the winters... bearable.

Thanks to the Physical Education department at MIT and to Gordon Kelly for giving me the best stress control there is — teaching aerobics.

Thanks to my family for always being there. I am grateful to my parents, Chaim and Chava, for everything you have done for me and especially for telling me that it's okay to get a 70 in History if I keep the 90+ in Mathematics. Thanks to my brother, Dan, for always setting high standards to reach, and to my sister, Tammy, for making sure I don't turn into a nerd in the process.

Last and most important I would like to thank my husband, Oren, for patience and support through every step of preparing this thesis. Thanks for help with the math, for reading and re-reading sections, and for learning more than you ever wanted to know about agents and experts. But most of all thanks for (almost) five truly wonderful years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The twentieth century is full of science fiction dreams of robots. We have Asimov's Robot series, Star Trek's Data, and HAL from "2001: A Space Odyssey". Recent Artificial Intelligence research on autonomous agents made the dream of robots that interact with humans in a human environment a goal. The current level of autonomous agent research is not near the sophistication of science fiction robots, but any autonomous agent shares with these robots the ability to perceive and interact with the environment. At the core of this direction of research is the agent's self sufficiency and ability to perceive the environment and to communicate with (or manipulate) the environment.

Machine learning researchers argue that a self sufficient agent in a human environment must learn and adapt. The ability to learn is vital both because real environments are constantly changing and because no programmer can account for every possible situation when building an agent. The ultimate goal of learning research is to build machines that learn to interact with their environments. This thesis is concerned with machines that learn the effects of their actions on the environment — namely, that learn a world model.

Before diving into the specifics of the problem addressed in this thesis, let us imagine the future of learning machines. In the following scenario, a robot training technician is qualified to supervise learning robots. She describes working with a secretary robot that has no world knowledge initially and learns to communicate and perform the tasks of a secretary.

September 22, 2xxx

Dear Mom,

I just finished training a desk-top secretary — one of the high-tech secretaries that practically run a whole office by themselves. It's hard to believe that in just a week, a pile of metal can learn to be such a useful and resourceful tool. I think you'll find the training process of this robot interesting.

The secretary robots are interesting because they learn *how to do their job. Unlike the simple communicator that you and I have, you don't have to input all of the information the secretary robot needs (like addresses and telephone numbers). Rather it learns the database as you use it. For example, if you want to view someone whose location isn't in your database, the secretary would find it and get him on view for you. It can also learn new procedures as opposed to our hardwired communicators, which means that it improves and changes with your needs. I can't wait until these things become cheap enough for home use.*

The training begins by setting up the machine with the input and output devices it will find in its intended office, and letting the machine experiment almost at random. I spent a whole day just making sure it doesn't cause any major disasters by sending a bad message to an important computer. After a day of mindless playing around the secretary understood the effects of its actions well enough to move to the next training phase. It had to learn to speak first, which it did adequately after connecting to the Language Acquisition Center for a couple of hours. I believe learning language is so fast because the Language Acquisition Center downloads much of the language database.

At this point my work began. I had to train the secretary's office skills. I gave it tasks to complete and reinforced good performance. If it was unsuccessful I showed it how to do the task. At this stage in training the robot is also allowed to ask for explanations, which I had to answer. This process is tedious because you have to repeat tasks many times until the training is sufficiently ingrained. It still doesn't perform perfectly, but the owners understand that she will continue to improve.

The last phase of training is at the secretary's future work place. The secretary's owner trains it in specific office procedures, and it accumulates the database for its office.

I am anxiously awaiting my next assignment — a mobile robot...

Ruti

The secretary robot in the above scenario is different from current machines (robots or software applications) because it leaves the factory with a learning program (or programs) but without world knowledge or task oriented knowledge. Current technology relies on programming rather than learning. Machines leave the factory with nearly complete, hardwired knowledge of their task and necessary aspects of their work environment. Any information specific to the work-place must be given to the machine manually. For example, the user of a fax program must enter fax numbers explicitly. Unlike the secretary robot, the program cannot learn additional numbers by accessing a directory independently.

To date it is impossible and impractical to produce machines that learn, especially with as little as the secretary robot has initially. Learning is preferable to preprogramming, even at a low level, when every environment is different, e.g. different devices or a different office layout for a mobile robot. Machine learning researchers hope that, due to better learning programs and faster hardware, learning machines will be realistic in the future. This thesis takes a small step toward developing such learning programs.

We examine the problem of an autonomous agent, such as the secretary robot, with no a priori knowledge learning a world model of its environment. Previous approaches to learning causal world models have concentrated on environments that are too "easy" (deterministic finite state machines) or too "hard" (containing much hidden state). We describe a new domain — *environments with manifest causal structure* — for learning. In such environments the agent has an abundance of perceptions of its environment. Specifically, it perceives almost all the relevant information it needs to understand the environment. Many environments of interest have manifest causal structure and we show that an agent can learn the manifest aspects of these environments quickly using straightforward learning techniques. This thesis presents a new algorithm to learn a rule-based causal world model from observations in the environment. The learning algorithm includes a low level rule-learning algorithm that converges on a good set of specific rules, a concept learning algorithm that learns concepts by finding completely correlated perceptions, and an algorithm that learns general rules. The remainder of this section elaborates on this learning problem, describes unfamiliar terms, and introduces the framework for our solution.

The agents in this research, like the robot in the futuristic letter, are autonomous agents. An autonomous agent perceives its environment directly and can take actions, such as move an object or go to a place, which affect its environment directly. It acts autonomously based on its own perceptions and goals, and makes use of what it knows or has learned about the world. Although people may give the agent a high level goal, the agent possesses internal goals and motivations, such as survival and avoiding negative reinforcement.

Any autonomous agent must perceive its environment and select and perform an action. Optionally, it can plan a sequence of actions that achieve a goal, predict changes to the environment, and learn from its observations or external rewards. These activities may be emphasized or de-emphasized in different situations. For example, if a robot is about to fall off a cliff, a long goal oriented planning step is superfluous. The action selection, therefore, uses a planning and decision making algorithm which relies heavily on world knowledge. The agent can learn world knowledge from its observations, and from mistakes in predicting effects of its actions. Learning increases or improves the agent's world knowledge, thus improving all of the action selection, prediction, and planning procedures.

An autonomous agent must clearly have a great deal of knowledge about its environment. It must be able to use this knowledge to reason about its environment, predict the effects of its actions, select appropriate actions, and plan ahead to achieve goals. All the above problems — prediction, action selection, planning, and learning — are open problems and important areas of research. This thesis is concerned with how the agent learns world knowledge, which we consider a first step to solving all the remaining problems.

As we see in the secretary robot training scenario, there are several stages in learning. In the initial stage, the agent has little or no knowledge about the environment and it learns a general world model. In later stages the agent already has some understanding of the environment and it learns specific domain information and goal-oriented knowledge. This thesis deals with the initial stages of learning, where the agent has no domain knowledge. The agent uses the perceptual interface with its environment and a learning algorithm to learn a world model of its environment.

The robot training scenario also indicates that there are several learning paradigms. Initially, the agent learns by experimenting and observing. Subsequent learning stages include learning from examples, reinforcement learning, explanation-based learning, and apprenticeship learning. This thesis addresses autonomous learning, as in the early stages of learning, from experiments and observations. In the autonomous learning paradigm, the agent cannot use the help of a teacher. For example, in the early training of the secretary robot, the trainer plays the role of a babysitter more than that of a teacher. The trainer is available in case of an emergency; this is especially important for mobile robots that can damage themselves. Rather than learn from a teacher, the agent learns through the perceived effects of its own actions. It selects its actions independently; the goal of building an accurate world model is its only motivation.

Because known learning algorithms are successful when the agent learns a simple environment or begins with some knowledge of the environment, but the learning techniques do not scale for complex domains, we examine a class of environments in which learning is "easy" despite their complexity. These environments have *manifest causal structure* — meaning that the causes for the effects sensed in the environment are generally perceptible. In more common terms, there is little or no locally "hidden state" in the environment — or rather in the sensory interface of the agent with the environment. Although many environments have manifest causal structure, this class of environments in unexplored in the machine learning literature. This thesis hypothesizes that environments with manifest causal structure allow the agent to use simple learning techniques to create a causal model of its environment, and presents, in support of this hypothesis, algorithms that learn a world model in a reasonable length of time even for realistic problems.

In this thesis an autonomous agent "lives" in a complex environment with manifest causal structure. The agent begins learning with no prior knowledge about the environment and learns a causal model of its environment from direct interaction with the environment. The goal of the work in this thesis is to develop learning algorithms that allow the agent to successfully and efficiently learn a world model of its environment.

The agent learns the world model in two phases. First it learns a set of rules that describe the environment in the lowest possible terms of the agent's perceptions. Once the perception-based model is adequate, the agent learns higher level concepts using the previously learned rules. Both the rule learning algorithm and the concept learning algorithms are novel. The concept learning is especially exciting since previous learning research has not been successful in learning general concepts in human readable form.

This thesis demonstrates the learning algorithms in the Macintosh Environment — a simplified version of the Macintosh user-interface. The Macintosh Environment is a complex and realistic environment. Although the Macintosh user-interface is deterministic, an agent perceiving the screen encounters some non-determinism (see Sections 1.4 and 2.2 for a complete discussion). While the Macintosh Environment is complex and non-deterministic, it has manifest causal structure and therefore is a suitable environment for this thesis. In the Macintosh Environment, like the secretary robot scenario, the agent learns how the environment (the Macintosh operating system) responds to its actions. This knowledge can then be used to achieve goals in the environment.

The remainder of this chapter has two parts. The first part (comprised of Sections 1.1, 1.2, and 1.3) discusses the motivation for this research. Section 1.1 discusses the manifest causal structure property in detail and illustrates its usefulness and relation to human and animal environments. Section 1.2 gives a brief overview of work on learning world models and contrasts previous approaches to learning causal world models with the approach of this thesis. Section 1.3 presents some of the large body of previous work in Artificial Intelligence that uses causal world models to plan, predict, and reason.

The second part of this chapter (comprised of Sections 1.4 and 1.5) is more technical and presents, without detail, the salient ideas of the thesis. Section 1.4 overviews the Macintosh Environment in which the agent experiments and learns. Section 1.5

describes the structure of the world model the agent learns, the methodology for learning the causal rules that make up the world model, and the two concept-learning paradigms: collapsing correlated perceptions and generalizing rules. For a complete discussion of the algorithms mentioned in this chapter refer to the respective chapter for each topic.

1.1 Manifest Causal Structure

Definition **manifest**: readily perceived by the senses and esp. by the sight. *synonyms: obvious, evident* [Webster's dictionary]

This thesis addresses learning in environments with manifest causal structure. As the name indicates, in such environments the agent can in general directly sense the causes for any perceived changes in the environment. In particular, the agent can sense (almost all) the information relevant to learning the effects of its actions on the environment.

The restriction of environment types to environments with manifest causal structure contrasts with research on learning in environments with hidden information, such as (Drescher 1989, Rivest & Schapire 1989, Rivest & Schapire 1990). The manifest causal structure of the environment eliminates the need to search beyond the perceptions for causes to changes in the environment. We claim that the strategies for learning the world model can therefore be fast and simple (compared with other techniques, such as the schema mechanism (Drescher 1989)).

While the agent may need a great deal of sensory information to achieve the manifest causal structure property, the sensory interface does not necessarily capture the complete state of the environment. This direction of research is in contrast with much of the work on autonomous agent learning, such as Q-learning (Sutton 1990, Sutton 1991, Watkins 1989), where states of the world are enumerated and the agent perceives the complete state of the environment. For the manifest causal structure property, locally complete sensory information usually suffices since changes to the local environment can in most cases be explained by the local information. For example, consider an environment with several rooms. An agent in this environment needs to perceive only its local room not the state of the other room in order to explain most perceived change to the environment.

This thesis draws an important distinction between the true environment in which the agent lives and the environment as the agent perceives it. The true environment is the environment the in which the agent lives, and it may be deterministic or non-deterministic. Notice that a non-deterministic environment is not completely manifest, i.e. its causal structure cannot be captured in all cases. For generality, environments with manifest causal structure can exhibit some unpredictable events as long as they occur relatively rarely. The environment, as the agent perceives it, is a product of the underlying environment and the agent's perceptual interface. The perceptual interface can make the underlying environment manifest or partially hidden. Typically the perceptual interface will map several underlying world states to one perceptual state, thereby hiding some aspects of the environment. Such environments have manifest causal structure if effects are predictable almost all the time.

The manifest causal structure property, therefore, is a property of the causal structure of the environment together with the agent's perceptual interface. In simple environments very little sensory data is sufficient to achieve manifest causal structure. For example, consider a room with a single light and a light switch that can be in either on or off position, and an agent that is interested in predicting if the light is on or off. One binary sensor is sufficient to perceive the relevant aspect of the environment – light on/off. In more complex environments the sensory interface must be much more complicated. For example, in the real world people and other animals have developed very effective sensory organs that perceive the environment (such that they achieve the manifest causal structure property), and they are able to understand the causal structure and react effectively.

I believe the restriction of the problem to environments with manifest causal structure is a natural one. People, as well as other animals, do not cope well with environments that are not manifest. In fact, it is so important to people that their environment be manifest that they go a step beyond the perceptual abilities with which nature endowed them. People build sensory enhancement tools such as microscopes to perceive cellular level environments, night vision goggles for dark environments, telescopes for very large environments, and particle accelerators for sub-atomic environments. Many agent-environment systems with appropriate sensory interfaces, such as animals in the real world, have manifest causal structure. In an artificial environment it is straightforward to give the agent sufficient sensory data to achieve the manifest causal structure property.

While it is believable that software environments with a single agent can have manifest causal structure, it is not clear that the notion of manifest causal structure generalizes to more complex environments and real-world settings. One complication is the presence of multiple actors in the environment. Very few environments are completely private. Even in one's office the phone may ring or someone could knock on the door. In some cases, such as a private office, occurrences due to other actors may be rare enough that the environment is still predictable almost all the time. Often other actors are continually present and affect one's environment. An environment with multiple actors can have manifest causal structure if the agent can perceive the actions of other actors and predict the results of their actions.

Another complication in real-world environments is the abundance of perceptual stimuli. An agent in an environment with manifest causal structure likewise has many perceptions. The advantage of the large number of perceptions is that the causes of events are found in the perceptions; the disadvantage is that the search space for the causes is large. In a complex environment it is possible for the agent to perceive too much. That is, the agent may perceive irrelevant information that makes the environment appear probabilistic or even random, when a more focused set of perceptions would show a predictable relevant subset of the environment.

People are very good at attending to relevant aspects of their environment. For example, when I work alone in my office I would immediately respond to a beep on my computer (indicating new mail), but when I am in a meeting I do not seem to hear these beeps at all. People are similarly good at recognizing when they do not perceive enough of the environment, and extend their perceptions by turning on a light or using a magnifying glass, for example. Such smart perceptual interfaces may be one way to achieve manifest causal structure in complex environments.

In addition, hidden state can sometimes become manifest by extending the perceptual interface with memory. For example, if there is a pile of papers on the desk it is impossible to know which papers will be revealed by removing papers from the top of the pile. The memory of creating the pile makes the hidden papers known. An agent can use the memory of previous perceptions, like it uses current perceptions, to explain the effects of actions.

Currently, endowing machines in the real world (i.e., autonomous robots) with perception is very problematic. We do not have the technology to give robots the quality of information that is necessary to achieve an environment with manifest causal structure. Most of the sensors used in robotics are very low level and give only limited information. The more complex sensors such as cameras give a large amount of information, but we do not have efficient ways to interpret this information. As a result, much of the environment remains hidden. Thus, although I believe that if we were able to create a sensory interface for robots that achieves the manifest causal structure property then the techniques for learning and performing in the environment would be useful in robots, I do not expect these techniques to be practical for any robots currently in use.

To summarize, many environments have manifest causal structure with careful selection of the perceptual interface. The problem of determining the necessary perceptions in any environment is difficult and remains up to the agent designer. The following discussion summarizes the possible environment types and under what conditions environments have manifest causal structure. (In the remainder of this thesis *environment* refers to the agent's perceived environment and *underlying environment* to the true environment.)

There are four types of underlying environment/perceptual interface combinations. Table 1.1 shows the type of the perceived environment for each of the four combination types. The environment can either be deterministic or probabilistic.

When the underlying environment is deterministic and the perceptual interface is

	perceptual interface	
underlying environment	*manifest interface*	*hidden interface*
deterministic	1. deterministic	2. probabilistic
non-deterministic	3. probabilistic	4. probabilistic

Table 1.1: Four Types of Environments

manifest, the environment is deterministic from the agent's perspective. The environment is essentially a finite automaton (assuming a finite number of perceptions) and therefore it has completely manifest causal structure. The three remaining environment types appear probabilistic to the agent. In the second environment type there are probabilistic transitions when two underlying states that collapse to one perceptual state have different successor states following an action. The effect of this action in the perceptual state appears to probabilistically choose one of the two effects from the underlying states. In the third environment type, the probabilistic effects are due to the underlying environment, and the fourth environment type is probabilistic for both of the above reasons.

We say that probabilistic environments have manifest causal structure if the degree of non-determinism is small. The degree of non-determinism of the environment can be anywhere from 0 (deterministic environment) to 1 (random environment). (The degree of non-determinism of an environment is not always well-defined — see Chapter 3 for an extended discussion of this issue.) Although our intuition tells us that environments with manifest causal structure should have a small amount of non-determinism (e.g., unpredictable events occur with probability at most .2), we do not impose a bound on the uncertainty of the environment. Rather the learning algorithm uses the known degree of non-determinism of the environment $(1-\Theta)$. The algorithm learns only causal relation in the environment that are true with probability Θ. As the degree of non-determinism increases, the correctness of the world model decreases.

Although it seems intuitive that environments with manifest causal structure should be easy to learn, since all the relevant information is available, the idea has not been explored by researchers. Figure 1-1 compares the domain of environments

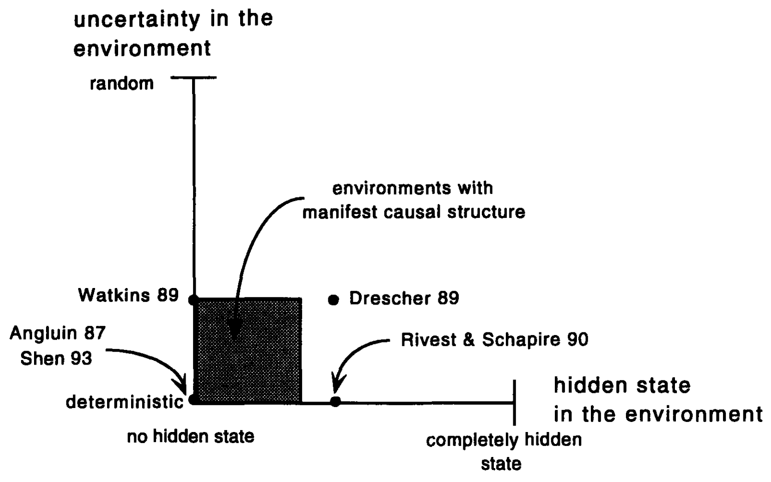


Figure 1-1: A comparison of the domain of environments with manifest causal structure with environments explored by other machine learning research.

with manifest causal structure with environments explored by other machine learning researchers. The graph compares these environments on two aspects: the degree of uncertainty and the amount of hidden state in the environment. First note that it is impossible and not interesting to learn in environments with a high degree of uncertainty or with a large amount of uncertainty. Therefore, most of the research activity is concentrated in a small section of the graph. Environments with manifest causal structure are represented by the shaded region. Such environments allow a restricted amount of hidden state and uncertainty.

Much of the research on learning is concerned with deterministic environments with no hidden state (Angluin 1987, Shen 1993). These learning algorithms cannot learn models of environments with any uncertainty, so they are not applicable to learning in environments with manifest causal structure which permit some uncertainty. Reinforcement learning research, such as Q-learning (Watkins 1989, Sutton 1990, Sutton 1991), can cope with some uncertainty but assume complete state information which is not guaranteed in environments with manifest causal structure. Rivest & Schapire (1990) and Drescher (1989) explore environments with a fair amount of hidden state. The learning algorithm developed by Rivest & Schapire (1990) is not applicable to environments with manifest causal structure since it assumes that the underlying environment is deterministic. Dean, Angluin, Basye, Engelson, Kaelbling, Kokkevis & Maron (1992) (not in Figure 1-1) use a deterministic environment with some sensory noise which similarly is more restrictive than environments with manifest causal structure. The schema mechanism (Drescher 1989) is applicable in environments with manifest causal structure, but the learning technique is slow.

The restriction of the learning problem to environments with manifest causal structure does not trivialize the problem. The inherent difficulties of learning (such as the need for many trials, the large search space, the problem of representing and using the learned information) remain, but the learning strategies do not have to be smart about inventing causes, only about grasping what is perceived.

This thesis shows that in environments with manifest causal structure the agent learns efficiently using straightforward strategies. The learning techniques are simple to implement and efficient in practice, and the techniques should extend to environments which are more complex than the kinds of environments dealt with in past research.

1.2 Learning World Models

Autonomous agents typically learn one of two types of world models. The first is a mapping from states of the world (or sets of sensations) to actions (formally $S \rightarrow A$). The second is a mapping from states and actions to states (formally $S \times A \rightarrow S$). We call the first mapping a *goal-directed world model*, since it prescribes what action to take with respect to an assumed goal, and the second a *causal world model*, since it indicates the resulting state when taking an action in a given state.

There are several known techniques for learning a goal-directed world model. Among these are reinforcement learning algorithms such as genetic algorithms and the bucket brigade algorithm (Holland 1985, Wilson 1986, Booker 1988), temporal differencing techniques (Sutton & Barto 1987, Sutton & Barto 1989), interval estimation (Kaelbling 1990), Q-learning (Watkins 1989, Sutton 1990), and variants of Q-learning (Sutton 1991, Mataric 1994, Jaakkola, Jordan & Singh 1994). These techniques are useful for some applications but do not scale well and suffer from the following common limitation. Since the agent learns a goal directed world model it throws away a great deal of the information it perceives and keeps only information that is relevant to its goal. If the agent's goal changes it has to throw away all its knowledge and re-learn its environment with this new perspective.

For example, suppose a secretary robot needs to contact a client. It quickly learns a goal directed model which prescribes the proper sequence of digits to dial on the telephone. Six months later the client moves to a new location with a new telephone number. The secretary is unable to contact the client using its current model. It now has a new goal (dialing the new number sequence) and it must re-learn the entire procedure for contacting the client. If the secretary learns a causal world model, then it spends some time making the right plan each time it calls the client. When the client's number changes, however, it still knows that the proper tool for communication is the telephone and it learns only the new number. Thus following a change in the environment, the secretary can patch a causal world model, but if it uses a goal-directed world model it must learn a completely new model.

The advantage of a causal world model is that it stores more information about how the environment behaves. Therefore a local change in the environment forces small adjustments in the model, but does not require learning a new model. In addition the causal knowledge can be used to reason about the environment, and, specifically, to predict the outcome of actions.

The disadvantage of a causal world model is that the abundance of information leads to slower planning, predicting, and learning in such models compared with these operations in goal-directed world models. For example, using a causal world model to plan requires planning, which is a long operation, for every goal (even goals that have been achieved previously). However, regenerating plans for a goal can be avoided by *chunking* plans (Laird, Newell & Rosenbloom 1978). Saving previous plans by chunking increases the efficiency of using causal world models.

This thesis concentrates on learning a causal world model because in the initial stages of learning the agent learns general domain knowledge that is relevant to many tasks. A goal directed world model is more appropriate for learning to perform specific tasks.

We are interested in efficient learning of causal world models. To date, causal world models have been efficiently learned for very restricted environments such as finite automata (Angluin 1987, Rivest & Schapire 1989, Rivest & Schapire 1990, Dean et al. 1992, Shen 1993) or with some prior information as in learning behavior networks (Maes 1991). Attempts to learn a causal model of more complex environments with no prior information, such as the schema mechanism (see Drescher (1989), Drescher (1991), and Ramstad (1992)) have not resulted in efficient learning. The algorithms this thesis presents lead to efficient learning for more types of environments.

1.3 Using Causal World Models

Although this thesis concentrates on the problem of learning causal models, it is important to note that there is a large body of work in AI using causal models for planning, predicting, and causal reasoning.

Planning research is concerned with using a causal world model to find a sequence of actions that will reach a goal state (see STRIPS (Fikes & Nilsson 1971) and GPS (Newell, Shaw & Simon 1957)). The main issues in planning are the efficiency of the search, and robustness of the plans to failing actions, noise, or unexpected environmental conditions (see Kaelbling (1987), Dean, Kaelbling, Kirman & Nicholson (1993), and Georgeff & Lansky (1987) for discussion on reactive planning). With the exception of unexpected environmental conditions, which are rare in environments with manifest causal structure, these planning issues are important for an agent using the world model learned in this thesis.

Causal reasoning paradigms solve prediction and backward projection problems. Prediction problems are: "given a causal model and an initial state, what will be the final state following a given sequence of actions?" Backward projection problem are: "given a causal model, an initial state, and a final state, what actions and intermediate states occurred?". Much of the research toward causal reasoning systems involves defining a sufficiently expressive logical formalism to represent causal reasoning problems (see, e.g., (Shoham 1986, Shoham 1987, Allen 1984, McDermott 1982)). Shoham (1986) presents the logic of chronological ignorance which contains causal rules that are closely related to the representation of the world model in this thesis.

Early work on causal reasoning uncovered the *frame problem*: knowing the starting state and action does not necessarily mean that we know everything that is true in the resulting state. A simple solution to the frame problem is to assume that any condition that is not explicitly changed by the action remains the same. This simple solution is inadequate when there is incomplete information about the state or actions. For example, Hanks & McDermott (1987) propose the Yale shooting problem where a person is alive and holds a loaded gun at time 1, he shoots the gun at time 2, and the question is if he is alive following the shooting. Two solutions exist for this problem. The first solution is the natural solution where the person shot is not alive, and in the second solution the gun is unloaded prior to shooting and the person remains alive. There are many approaches to solving this problem in the nonmonotonic reasoning literature (among them Stein & Morgenstern (1991), Hanks & McDermott (1985), and Baker & Ginsberg (1989)).

In an environment with manifest causal structure an agent is typically concerned with prediction problems not with backward projection problems, since it perceives relevant past conditions. (Such relevant past conditions are rarely not present.) The agent also perceives all the actions that take place in the environment, so prediction is straightforward given an accurate world model. For example, in the Yale shooting problem it is not possible for the *unload* action to take place without observing this action, so the only feasible solution is the correct one — that the person shot is not alive. Thus, due to the restriction of the environment type, the learning and prediction algorithms in this thesis use the assumption that conditions remain unchanged unless a change is explicit in some rule.

At this point we have discussed, at length, the problem that this thesis addresses. We will now introduce a specific environment, in which the agent in this thesis learns, and the approach this thesis takes to solve the problem of learning a world model in an environment with manifest causal structure.

1.4 The Macintosh Environment

This thesis uses the Macintosh Environment, which is a restricted version of the Macintosh user-interface, to explore principles of efficient autonomous learning in an environment with manifest causal structure. In the Macintosh Environment the agent "observes" the screen of an Apple Macintosh computer (e.g., see Figure 1-2) and learns the Macintosh user-interface — i.e., how it can manipulate objects on the screen through actions. This learning problem is realistic; many people have learned the Macintosh interface, which makes this task an interesting machine learning problem. The Macintosh user-interface has had great success because it is manifest and therefore easy to use. The Macintosh Environment fulfills the requirements for this thesis since it is a complex environment with a manifest causal structure.

Learning the Macintosh user-interface is more challenging for an agent with no prior knowledge than for people because people are told much of what they need to know and do not learn tabula rasa. Many people find the structure of windows natural because it simulates papers on a desk. Bringing a window to the front has the same effect as moving a paper from a pile to the top of the pile and so on. The learning agent in this thesis has no such prior knowledge. Furthermore, when people learn to work on a computer they typically have a user manual or tutor to tell them the tricks of the trade and the meaning of specific symbols. By contrast, the agent learns the meaning (and function) of the symbols and boxes on the screen, as well as how windows interact, strictly through experimentation.

Learning the Macintosh Environment suggests the possibility of machines learning the operation of complex computer systems. Although a very general application of the learning algorithm, such as the secretary robot, is overly ambitious at this time, some applications seem realistic. For example, there has been considerable interest in interface agents recently (Maes & Kozierok 1993, Sheth & Maes 1993, Lieberman 1993). Research on interface agents to date concentrates on agents that assist the user of computer software or networking software. The interface agents learn procedures that the user follows frequently, and repeats these procedures automatically or on

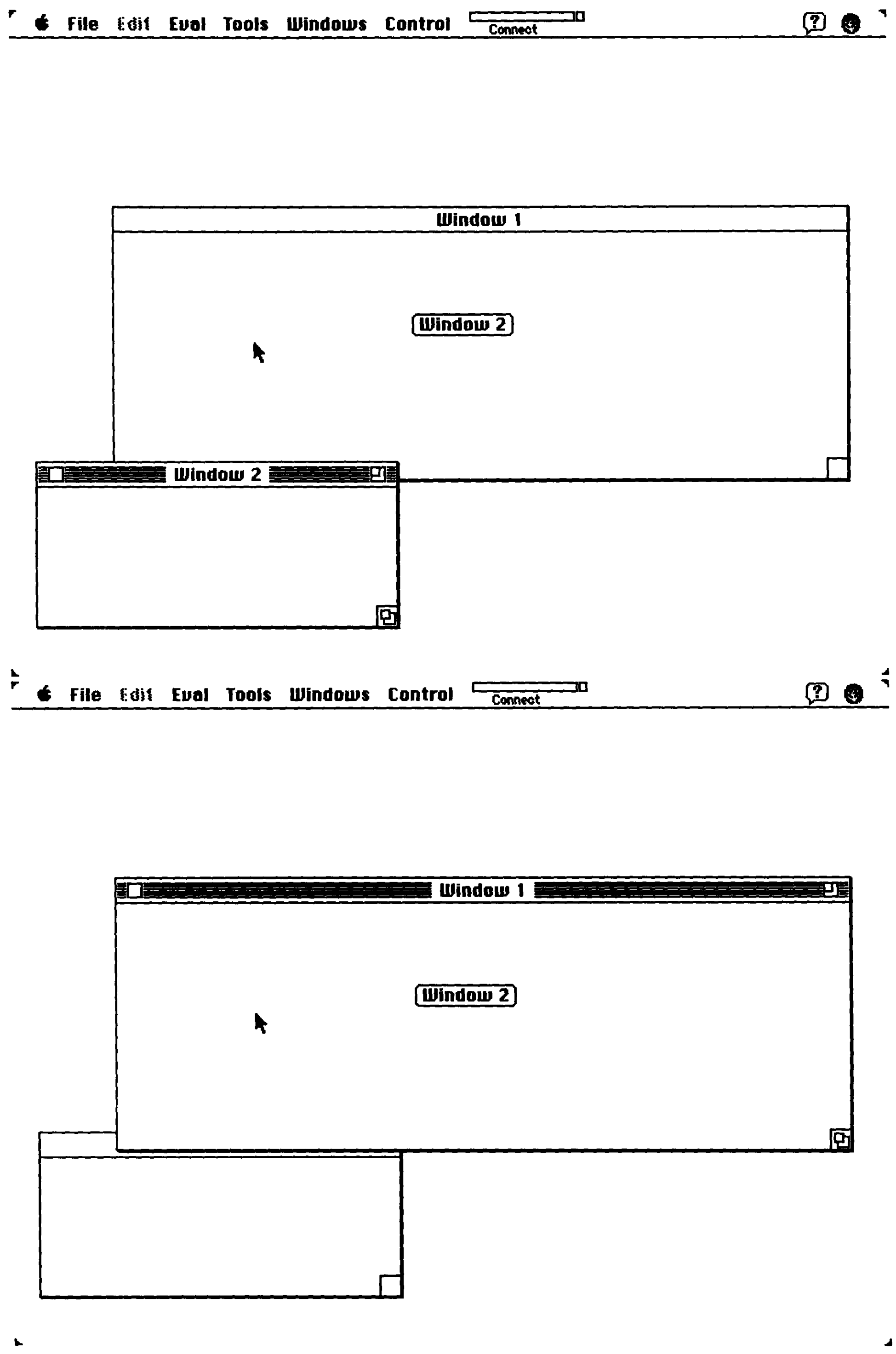


Figure 1-2: Macintosh screen situations before and after a click in *Window 1*

demand. In this way the agent takes over some tedious tasks, such as finding an interesting node on the network.

The learning agent in this thesis can be part of a "smarter" interface agent. The smarter agent can learn about the software environment, and can use this knowledge to act as a tutor or advisor to a user. The agent can spend time learning about the environment in "screensaver" mode, where the agent uses the environment at those intervals when a screen-saver would run. It then uses the learned model to answer the user's question about the software environment. The implementation of such an application is outside the scope of this thesis, but it is an interesting direction for future research.

In the Macintosh Environment, the agent can manipulate the objects on the screen with *click-in object* actions (other natural actions for this environment, *double click* and *drag*, will not be implemented in this thesis.) The actions affect the screen in the usual way (see Section 2.2 for a summary of the effects of action in the Macintosh Environment). Notice that although time is continuous in this environment, it can be discretized based on when actions are completed.

The agent's perceptions of the Macintosh Environment can be simulated in several ways. People view the screen of a Macintosh as a continuous area where objects (lines, windows, text) can be in any position. Of course, the screen is not continuous: it is made up of a finite number of pixels. The agent could perceive the value of each pixel as a primitive sensation, but this scheme is not a practical representation for learning high level concepts. For this thesis the screen is represented as a set of rectangular objects with properties and relationships among them. (The perceptual representation is presented in full in Section 2.3.)

The agent, in the Macintosh Environment, learns how its actions affect its perceptions of the screen. Before we discuss the methodology for learning, consider what the agent should learn in the Macintosh Environment. Figure 1-2 shows two screen situations from the Macintosh Environment. In the first one *Window 2* is active and overlaps *Window 1*, and the second situation shows that following a click in *Window 1*, *Window 1* is active and overlaps *Window 2*.

This example demonstrates two important facts:

- a click in a window makes that window active, and
- if a window is under another window, then clicking it brings it in front of the other window.

We set these rules as sample goals for the learning algorithm. By the end of this thesis we will show how the algorithm learns these rules and other rules of similar complexity.

1.5 Learning the World Model

This section describes the representation of the world model and the approach of the learning algorithms.

1.5.1 The Agent's World Model

Recall that this thesis develops an algorithm for learning a causal world model efficiently for environments with manifest causal structure. The structure of the world model is based on schemas from the schema mechanism (Drescher 1989), although we refer to them as rules. As in the schema mechanism, the world model is a collection of rules which describe the effects of actions on perceptual conditions. We write rules as follows:

$$\text{precondition} \rightarrow \text{action} \rightarrow \text{postcondition}$$

where the precondition and postcondition are conjunctions of the perceptual conditions of the environment, and higher level concepts that the agent learns.

A rule describes the effects of the action on the environment. It indicates that if the precondition is currently true in the environment, then if the action is taken, the postcondition will be true. Notice that the rules in this model are not production rules, which suggest taking the action, or STRIPS operators (Fikes & Nilsson 1971), which add or remove conditions in the environment. Rather, rules remember a causal relationship that is true for the environment, and taken as a set they form a causal world model that is goal independent.

Once the agent learns a reliable set of rules it can use the world model to predict and plan. Known algorithms such as GPS (Newell et al. 1957) and STRIPS (Fikes & Nilsson 1971) can be adapted to plan and predict using these causal rules.

In Section 1.4 we discussed two rules we want the learning algorithm to learn. Now that we selected the representation for the world knowledge, we can describe the rules in more detail within the representation of rules. The first rule "a click in a window makes that window active" becomes

$$() \rightarrow \text{click-in } Window_x \rightarrow Window_x \text{ is active}$$

where () means an empty conjunction of preconditions. (This rule has the implied precondition that $Window_x$ is present because one cannot click in a window that is not on the screen.) The second rule "f a window is under another window, then clicking it brings it in front of the other window" becomes

$$Window_y \text{ overlaps } Window_x \rightarrow \text{click-in } Window_x \rightarrow Window_x \text{ overlaps } Window_y.$$

The description of these rules is high level and uses concepts, such as *active* that are unknown to the agent initially. The rule that the agent learns will be expressed in terms of its perceptions of the screen, and in terms of higher level concepts when the agent learns such concepts. A discussion on learning concepts follows in Section 1.5.3. For the time being we will discuss the Macintosh Environment with high level descriptions, and we can assume that some set of perceptual conditions captures the description. A complete description of the perceptual interface is given in Chapter 2.

The next two sections discuss the algorithms that learn the above rules. Like a child, the agent in this thesis learns specific low-level knowledge first, then builds on this knowledge with more advanced learning. Thus, the approach of this thesis uses two phases of learning. In the first phase, the rule-learning algorithm learns specific rules whose pre- and post-conditions are direct perceptions. A second learning phase uses the specific rules learned by the first phase to learn general rules with higher-level concepts.

1.5.2 Learning Rules

This section discusses an algorithm for learning rules about specific objects. In the example situation in Figure 1-2, where *Window 1* becomes active following a click in *Window 1*, the rule-learning algorithm learns rules such as

() → click-in *Window 1* → *Window 1* is present

() → click-in *Window 1* → *Window 1*'s active-title-bar is present

and

Window 2 overlaps *Window1*
→ click-in *Window 1* → *Window 1* overlaps *Window 2*.

The algorithm in this section learns such specific rules from observing the effects of actions on the environment. This algorithm performs the first phase of learning.

Our autonomous agent repeats the following basic behavior cycle:

Algorithm 1 Agent

```
repeat forever
    save current perceptions
    select and perform the next action
    predict
    perceive
    learn
```

The remainder of this section discusses the learning step of this cycle. The learning step executes at every cycle (trial), and at every trial the learning algorithm has access to the current action and the current and previous perceptions. The algorithm uses the observed differences between the current and previous states of the environment to learn the effects of the action. The learning algorithm in this thesis does not use prediction mistakes to learn, but uses prediction to evaluate the correctness of the world model.

The rule learning algorithm for the Macintosh Environment begins with an empty set of rules (no prior knowledge). After every action the agent takes, it proposes new rules for all the unexpected events due to this action. Unexpected events are perceptions whose value changed inexplicably following the action. At each time-step the learning algorithm also evaluates the current set of rules, and removes "bad" rules, i.e., rules that do not predict reliably.

The main points of the rule learning algorithm are discussed below.

Creating new rules The procedure for creating new rules has as input the following: (1) the postcondition, i.e. an observation to explain, (2) the last action the agent took, and (3) the complete list of perceptions before the action was taken.

The key observation in simplifying the rule learning algorithm is that because the environment has a manifest causal structure, the preconditions sufficient to explain the postcondition are present among the conditions at the previous time-step. The task of this procedure is to isolate the right preconditions from the previous perceptions list. The baseline procedure for selecting preconditions picks perceptions at random. Because of the complexity of the Macintosh environment there are many perceptions. Therefore, it is worthwhile to use some heuristics which trim the space of possible preconditions. (The heuristics are general, not problem-specific, and are described in Chapter 3.)

Separating the good rules from the bad After generating a large number of candidate rules the agent has to save the "good" rules and remove the "bad" ones. Suppose the environment the agent learns is completely deterministic. The perceptions in the current state are sufficient to determine the effects of any actions and there are no surprises. In such environments there is a set of perfect rules that never fail to predict correctly. Distinguishing good rules from bad ones is easy under these circumstances: as soon as a rule fails to predict correctly the agent can remove it.

Unfortunately the class of deterministic environments is too restrictive. Most environments of interest do not have *completely* manifest causal structure. For example, in the Macintosh Environment one window can cover another window completely, and if the top window is closed the hidden window is surprisingly visible. To cope with a small degree of surprise the rule reliability measure must be probabilistic.

The difficulty in distinguishing between good and bad probabilistic rules is that at any time the rule has some *estimated* reliability from its evaluation in the world. The agent must decide if the rule is good or bad based on this estimate rather than the true reliability of the rule. This problem is common in statistical testing, and several "goodness" tests are known. The rule-learning algorithm in this thesis uses the sequential ratio test (Wald 1947) to decide if a rule is good or bad.

Mysteries In most environments some situations occur rarely. The algorithm uses "mysteries" to learn about rare situations. The agent remembers rare situations (with surprising effects) as mysteries, and then "re-plays" the mysteries, i.e., repeatedly tries to explain these situations. Using mysteries the algorithm for creating rules executes more often on these rare events. Therefore, the rules explaining the events are created earlier. Mysteries are similar to the world model component of the Dyna architecture (Sutton 1991), which the agent can use to improve its goal directed model.

For the complete algorithm see Chapter 3. Chapter 3 also contains the results of learning rules in the Macintosh environment, and shows that the rule learning algorithm converges to a good model of the environment.

1.5.3 Learning New Concepts

The rules learned by the algorithms in the previous section are quite different from the rules we discussed as goals in Section 1.5. The differences are that (1) these rules refer to specific objects, such as *Window 1*, rather than general objects, such as $Window_x$, and (2) these rules do not use high-level concepts, such as *active*, as pre- and post-conditions — only perceptions are used. This section describes the concept-learning algorithms that bridge the gap between the specific rules learned by the rule-learning algorithm and our goal rules. We discuss two concept-learning algorithms, which find correlated perceptions and general rules.

Correlating perceptions is a type of concept learning which addresses the problem of redundant rules and finding the cause of an effect. For example, in the screen situation of Figure 1-3, *Window 1* disappears and *Window 2* becomes active. There is a simple rule to explain that *Window 1* disappears:

() → click-in *Window 1* close-box → *Window 1* is not visible.

(No preconditions are needed since clicking in *Window 1*'s close-box implies that the close-box exists which implies that *Window 1* is active.) Many rules explain why *Window 2* became active, among them the following:

Window 2 is visible → click-in *Window 1* close-box → *Window 2* is active

Window 2 interior is visible → click-in *Window 1* close-box → *Window 2* is active

Window 2 title-bar is visible → click-in *Window 1* close-box → *Window 2* is active.

Clearly most of these rules are redundant since whenever *Window 2* is visible and not active it has an interior and a title-bar, etc. Pearl & Verma (1991) makes the distinction between correlated conditions, such as the second and third rules above, and true causality, such as the first rule. (Note that the above rules are only true when *Window 1* and *Window 2* are the only windows on the screen. The examples throughout this thesis use situations with these two windows, and Chapter 3 discusses learning with additional windows.)

The algorithm to find correlated perceptions relies on the observation that some perceptions always occur together. To learn which perceptions are correlated the agent first learns rules such as

precondition → NOACTION → postcondition

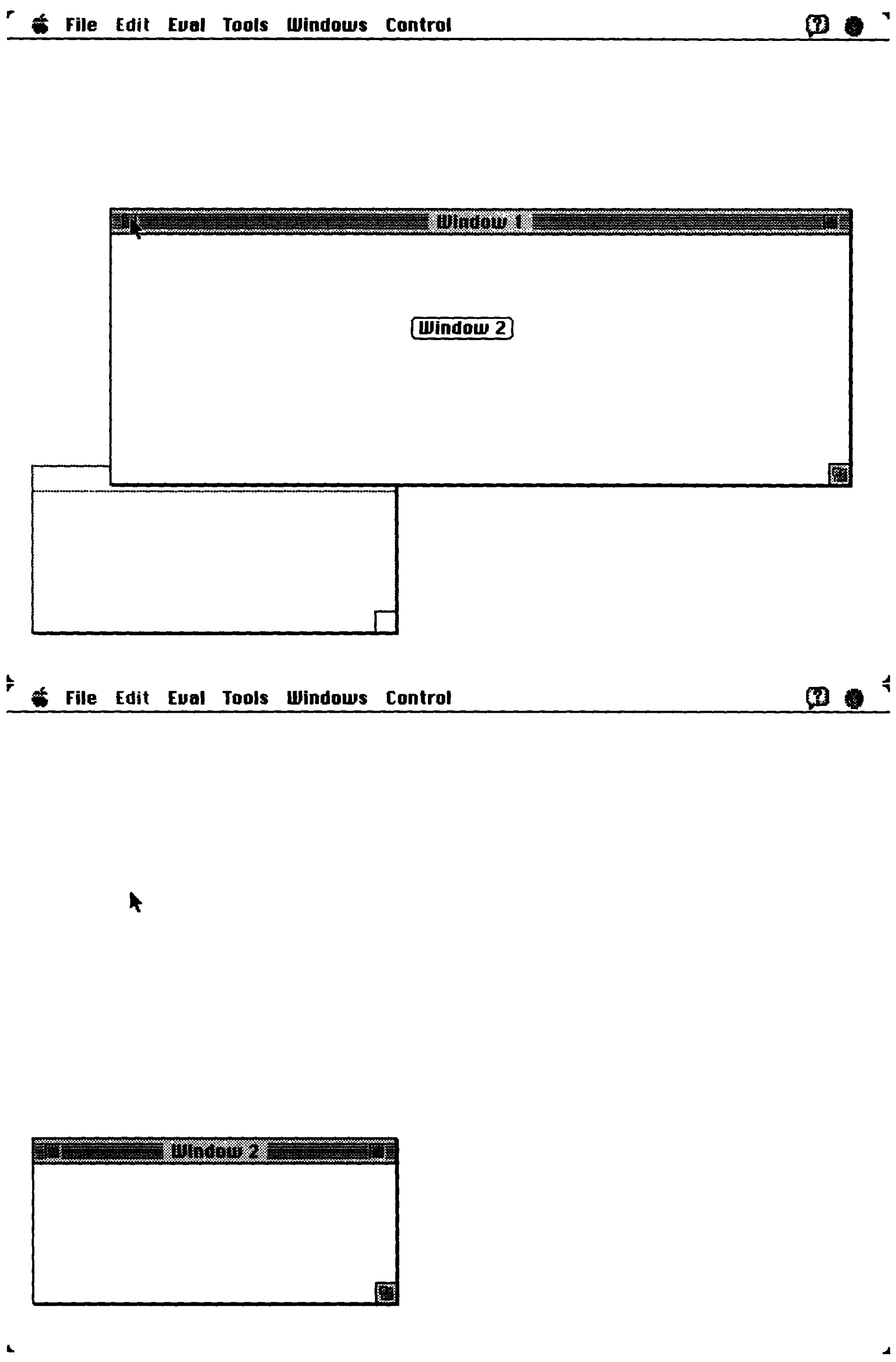


Figure 1-3: Macintosh screen before and after a click in *Window 1* close-box

when no action is taken. These rules mean that when the precondition is true the postcondition is also true in the same state. Notice that a set of NOACTION rules defines a graph in the space of perceptions. The agent finds correlated perceptions by looking for strongly connected components in this graph. A new concept is a component in the graph, i.e. a shorthand for the perceptions that co-occur.

The second type of concept learning addresses the problem of rules that are specific to particular instances in the environment. For example, consider a room with three light switches. The agent learns rules that explain how each of the light switches works, but when the agent moves to a different room with different light switches it has to learn how these light switches work. Instead, the agent should learn that there is a concept *light switch* and some rules that apply to all light switches. Similarly in the Macintosh Environment, there is a concept *window* and rules that apply to all windows.

The agent learns general concepts by finding similar rules and generalizing over the objects in those rules. When it generalizes over objects it adds the attributes of the objects to the preconditions of the general rule. For example consider the rules

$$() \rightarrow \text{click-in } \textit{Window 1} \text{ close-box} \rightarrow \textit{Window 1} \text{ is not visible}$$

$$() \rightarrow \text{click-in } \textit{Window 2} \text{ close-box} \rightarrow \textit{Window 2} \text{ is not visible.}$$

These rules are similar and indicate that the objects *Window 1* close-box and *Window 2* close-box should be generalized. The agent adds as precondition the attributes that the general object is a close-box and that this new object is part of the generalized window object. The generalized rule becomes

$$\textit{object}_x \text{ is an active window} \wedge \textit{object}_y \text{ is a close-box} \wedge \textit{object}_y \text{ is part of } \textit{object}_x \rightarrow \text{click-in } \textit{object}_y \rightarrow \textit{object}_x \text{ is not visible.}$$

See Chapter 5 for the complete algorithm for generalizing rules.

1.5.4 Evaluating the World Model

To examine the effectiveness of the learning algorithm we need to evaluate how good the model is. There are three ways of evaluating a world model: to compare it with a correct model, to test it as a predictor, or to use it as a basis for planning. This section examines the plausibility of each method of evaluation in turn.

The first form of evaluation is to compare the learned model with a correct model. In this form of evaluation the set of learned rules is compared with a set of a priori known rules and the evaluation returns the percentage of correct rules the agent learned. This method suffers from two drawbacks: (1) it assumes that a single correct model exists, and (2) someone (I) must encode the correct model manually. In many environments any number of non-identical world models are equally good, and the number of rules in a world model often prohibits manually coding the model. Thus, in this thesis, the world model is not compared with the "right" model. Rather, we examine examples of learned rules.

The second form of model evaluation is to predict the next state of the world given the current state and an action. This thesis primarily uses this method of evaluation. The prediction algorithm predicts postconditions from all the applicable rules, and assumes no change as a default when no rules apply (as discussed in Section 1.3).

The final form of model evaluation is to test the agent's ability to achieve a goal. This algorithm uses a backward chaining search to find action sequences that achieve goals. The planning algorithm is simplistic and not efficient enough for general use, but it is sufficient to demonstrate that the world model has the knowledge to achieve the goal.

1.6 Overview

The remainder of this thesis contains an extended discussion of the algorithms mentioned in this chapter with results from experiments in the Macintosh Environment. Chapter 2 discusses the Macintosh Environment and how the agent perceives the Macintosh screen. We present the rule-learning algorithm in complete detail in Chapter 3, along with a proof that in environments with manifest causal structure this algorithm converges to a correct world model. Chapters 4 and 5 contain the two concept learning algorithms: finding correlated perceptions and generalizing rules.

In addition, this thesis presents results on picking a good expert from a sequence in Chapter 6. This direction of research is tangentially related to the research on learning world models. It stems from the work on deciding if rules are good. In this research, we can examine experts (for example the experts may be rules) one at a time and we want to discard bad experts and keep the best one (i.e., the expert that makes fewest mistakes). Much like rule learning the experts come from an unknown distribution and their error probability is unknown. Unlike rules, whose "goodness" is determined by a known parameter, we cannot assume how good the best experts are. Chapter 6 presents an algorithm that finds an expert that is almost as good as the best expert we would expect to find if the experts error probabilities were known, given the same length of time.

Chapter 2

The Perceptual Interface

One of the main problems of Artificial Intelligence is knowledge representation: how to represent world knowledge in a complete and useful manner. This thesis encounters the knowledge representation problem at two levels. The first is the representation of the agent's perceptions of its world, and the second is the representation of the knowledge the agent learns. The representation of the learned world model is discussed in Chapter 3. This chapter presents the representation of the agent's perceptions.

An appropriate representation of any problem is a crucial step toward its solution. The "right" representation can make a difficult environment learnable, whereas the wrong representation can make the environment either too hard or trivial. In many cases a low-level representation creates a large search space, which prohibits effective learning algorithms. On the other hand, a representation can contain the difficult aspects of a complex environment reducing it to a trivial learning problem. In finding a representation for the perceptions of the agent, we must avoid both representations that are too low level and ones that are too high level. People's perceptions are a good guideline for appropriate representation — in particular perceptions of people who are not familiar with a given situation.

This thesis uses the Macintosh Environment as an example environment. In the Macintosh Environment the agent learns the Macintosh user-interface. That is, the effects of manipulating the screen of a Macintosh computer with clicking actions on windows and other objects on the screen. To learn in the Macintosh Environment the agent must perceive the screen of the Macintosh. This chapter develops a perceptual interface for the Macintosh Environment that follows the guideline for appropriate representations based on a layman's perceptions. For the Macintosh Environment these guidelines translate to the way a person who has never used a window interface perceives the screen, or even the way a young child perceives the screen. Such people may view the screen as a collection of rectangular objects with properties (such as the shape of the icons in the rectangles). Furthermore, the rectangular objects interact with one another by overlapping, being next to or above each other, etc. The perceptual representation in this chapter expresses these ideas in detail.

While the Macintosh Environment helps us to understand the representation problem in detail, we must remember that we are seeking a general representation. The learning algorithm this thesis develops is intended to be general enough to learn in a wide range of environments, with the Macintosh Environment merely serving as an example. Thus the perceptual representation must be sufficiently powerful to represent many problems.

With these issues in mind this thesis develops a general representation using mathematical-like relations on objects described in Section 2.1. Section 2.3 contains a lengthy discussion of the Macintosh Environment and the specific breakdown of the Macintosh screen into objects and relations.

2.1 Mathematical Relations as Perceptions

The representation of perceptions is a general mathematical formulation. The agent perceives **objects** and **relations** on objects. Each perception of the world is a relation on objects and the associated value — denoted as

$$R(o_1, \ldots, o_n) = v$$

where R is a relation on n arguments, $o_1, \ldots, o_n$ are perceived objects, and v is the value of R on arguments $o_1, \ldots, o_n$. Note that this representation is more general than standard mathematical relations, where v can only be true or false. We assume, however, that the set of possible values for any relation is finite.

A great deal of research in AI has used a symbolic representation that is similar to the relation representation. For example, in the traditional blocks world environment all conditions in the world are expressed as $CONDITION(o_1 \ldots o_n)$, such as $ON(A, B)$ meaning that block A is on top of block B. This condition is easily translated to the relation representation as $ON(A, B) = T$ and, in general, conditions can be translated to binary relations. Unlike the traditional condition representation, the relation representation is useful for describing multi-valued conditions, such as the color of a block, or even real-valued conditions, such as the distance relation.

As required, the relation representation is general enough to describe a wide range of environments. A learning algorithm using this representation can therefore learn in a variety of environments with no change to the algorithm. Furthermore, the generality of the representation permits the learning algorithm to be uniform in treating the knowledge it amasses. For example, the algorithm can learn information beyond its immediate perceptions by creating new relations and new objects or generalizing relations over objects (e.g. $\forall o R(o) = v$). Chapters 4 and 5 contain implementations of such advanced learning.

2.2 The Macintosh Environment

Before we examine, in detail, the perceptions of the Macintosh Environment, let us expand our discussion of the Macintosh Environment. Recall that the Macintosh Environment is a restricted version of the Macintosh user-interface. The agent in the Macintosh Environment observes the screen of a Macintosh computer and takes actions that affect the screen. The effects of the agent's actions on the screen objects are the same as the responses of the Macintosh user-interface to a user taking these actions. Section 2.2.1 describes the "laws of nature" for the Macintosh Environment. We discuss some important characteristics of the Macintosh Environment in Section 2.2.2 and Section 2.2.3 gives the historical reasons for selecting the Macintosh

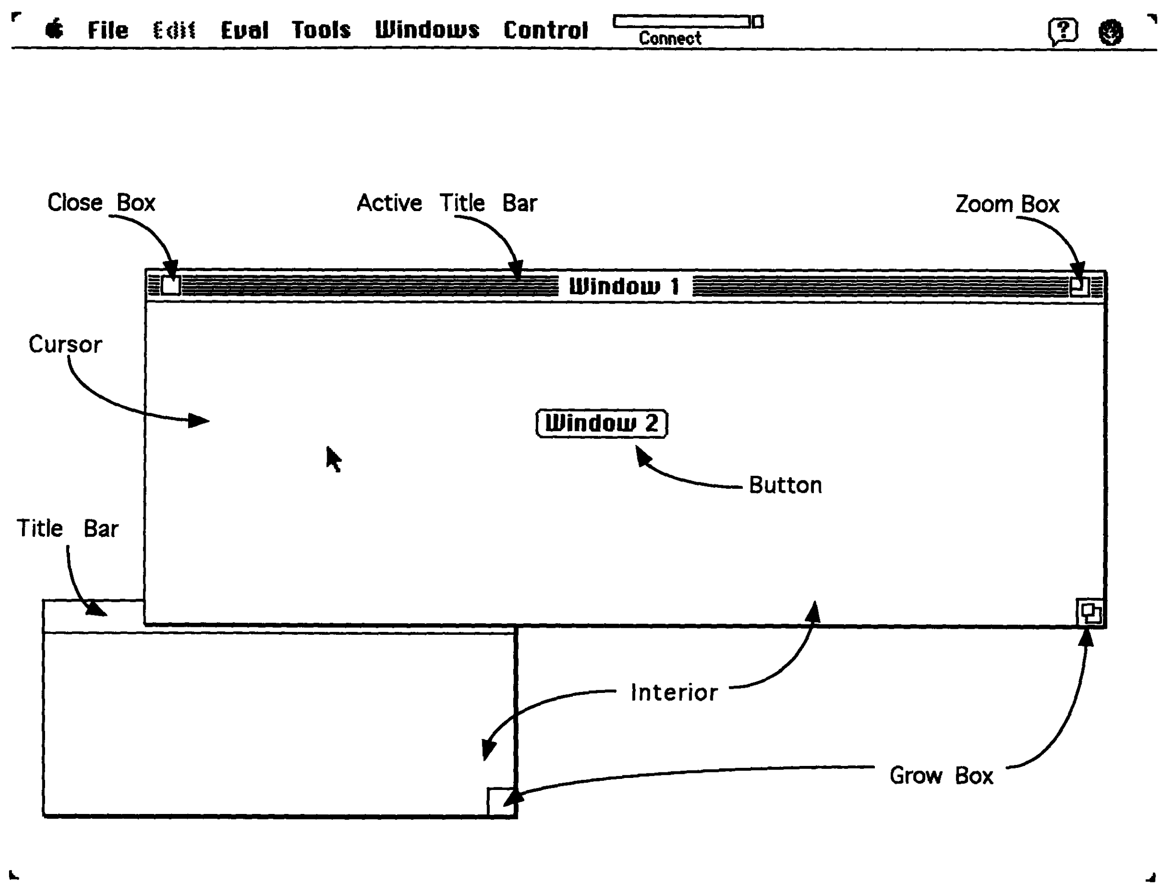


Figure 2-1: A description of a Macintosh screen situation

Environment for this research. The last section (Section 2.3) presents the perceptual interface of the Macintosh Environment in full detail.

2.2.1 The "Laws of Nature" in the Macintosh Environment

When the agent takes an action in the Macintosh Environment, the action (event) is handled by the Macintosh operating system. Thus the effects of actions in the Macintosh Environment are exactly the same effects that actions have in the Macintosh user-interface. This section describes the aspects of the Macintosh user-interface that are used in the Macintosh Environment. (For a complete description of the Macintosh user-interface see The Macintosh User's Guide.)

The key objects of the Macintosh user-interface are windows. In the screen situation of Figure 2-1 there are two windows. *Window 1* is active, and *Window 2*, whose title is hidden, is not active. At any time only one window is active. All windows have a title-bar and an interior where text and other information may appears. An active window has an active-title-bar, a close-box, a zoom-box, and a grow-box. These features have recognizable icons, such as the lines in the active-title-bar.

Clicking in any visible portion of an inactive window brings that window to the front and makes it the active window (for example, see Figure 2-2). A click in the active window causes no change, i.e., the active window remains active, unless the click occured in the close-box, the zoom-box, or an icon in the window interior. A click in the close-box of the active window closes that window. The window goes away and the window immediately under the active window becomes the new active window. (The Macintosh interface maintains an ordering of window layers at all times.)

A click in the zoom-box of the active window toggles the size of the window between its initial size and the biggest possible size that takes up the whole screen. (The two window sizes that the zoom-box toggles can be changed by re-sizing the window, but this feature is not used in the Macintosh Environment.)

The active-title-bar and the grow-box of an active window are important features for drag actions. Drag actions can re-size and move the active window, but these actions are not used in the Macintosh Environment.

The Macintosh Environment also uses buttons, such as the button labeled **Window 2** in Figure 2-1. Buttons in the active window are activated by a click action. In an inactive window, a click in the location of a button, like a click anywhere in the window, makes that window active. In the Macintosh Environment all the buttons have labels, such as **Window 2**, indicating that they open the corresponding window. The window opened by the button becomes active.

2.2.2 Characteristics of the Macintosh Environment

The behavior of the Macintosh Environment, as we saw in the previous section, is directly controlled by the Macintosh user-interface. Since the Macintosh user-interface is a deterministic finite state machine, so is the Macintosh Environment. The Macintosh Environment, however, only appears deterministic if the observer has access to all the knowledge that the Macintosh operating system has. If the observer, like a person, can only perceive what is visible on the screen then some apparent non-determinism arises.

For example, consider the situation where there is one large window visible on the screen. Typically, when the agent closes this window we expect that only the background will be visible. It is possible, however, that a smaller window is completely obscured by the top window. After closing the top window the hidden window will be visible. The appearance of the small window cannot be predicted by the perceptions of the previous state. Therefore, the event following closing the top window (background is visible only vs. a small window is visible) appears to occur probabilistically. The above example and a few similar situations make the Macintosh Environment, with a perceptual interface that includes only the current perceptions, a probabilistic environment. The environment/perceptual interface type of the Macintosh Environment is a deterministic underlying environment and a (slightly) hidden perceptual interface. The Macintosh Environment has manifest causal structure since unpredictable events, such as the above example, rarely occur.

It would be straightforward to incorporate memory in addition to direct perceptions, so that the Macintosh Environment will remain deterministic. For example, if a small window is hidden because of a click in another window, the agent can save the memory of the small window. When closing the large window the agent can predict the appearance of the small window using this memory. In this thesis, however, memory is not implemented. For our purposes, the fact that unpredictable events occur rarely so that the environment has manifest causal structure suffices.

An important characteristic of the Macintosh Environment is that the learner is the only actor affecting the environment. In addition, this environment has discrete time and space. Time is actually continuous in the Macintosh Environment, but since the learner is the only actor in the environment, we can consider discrete times between one action and the next. Space is, of course, discrete because of the finite number of pixels making up the screen of the computer.

These characteristics will affect some of the strategies of the learning algorithm.

In particular, the learning algorithms use the assumption that the environment has manifest causal structure which implies that the causes for any change to the screen are visible in the previous screen situation.

2.2.3 Why the Macintosh Environment? — A Historical Note

When this research began many of the decisions for the direction of research were influenced by earlier autonomous learning research. At a high level, the problem that this thesis addresses was selected because previous research deals with deterministic environments or environments with a great deal of hidden state. The class of environments this thesis explores — environments with manifest causal structure — seems a valuable, unexplored direction for research.

At the lower level of selecting an example environment to demonstrate the results of this research, it seemed reasonable to use an environment that is similar to environments machine learning researchers typically use. Most research on autonomous learning is demonstrated on artificial (simulated) grid environments (see (Drescher 1989, Booker 1988, Wilson 1986, Sutton 1991) for some examples). These simulated grid environments capture some (though certainly not all) aspects of real-world environments, such as office buildings or factory floors. Furthermore, they are simple to implement and to endow with any desired characteristics, such as noise or hidden state. Therefore, a simulated grid environment is an obvious problem to choose for an autonomous agent learning a world model.

This research began with a grid environment as an example environment. It was straightforward to implement a grid environment with manifest causal structure, and to represent perceptions of the environment in terms of the relations in Section 2.1. Preliminary results in the grid environment seemed promising — the agent achieved efficient learning of a near perfect model for a small problem. Other researchers, however, were reluctant to believe the generality of these results. After much discussion of this issue, it became clear that results in such an artificial environment are not sufficiently motivating and convincing to support the claims this thesis makes. The ease of learning is attributed to the construction of the environment, rather than to the learning algorithms.

Clearly, a more natural example environment was needed. The obvious example environment would be the real world, for example, the eighth floor of the AI Lab at MIT. However, as we discussed in Chapter 1, an autonomous agent (robot) in a real-world setting cannot have a set of perceptions such that the environment has manifest causal structure. This thesis would have become a thesis on perceiving rather than on learning.

The search for a software environment that is realistic and complex, and has manifest causal structure led to the Macintosh Environment. People are sympathetic to the complexities of perception and learning in the Macintosh Environment because they also have learned this (or a similar) environment. Although the learning problem is different for people and for the agent in this thesis, the realistic nature of the Macintosh Environment makes it a motivating example for this research. This environment is further motivated by the increasing interest in interface agents (Maes & Kozierok 1993, Sheth & Maes 1993, Lieberman 1993) which must cope with similar user-interface environments. Finally, since the Macintosh Environment is complex and its behavior is evident on the screen, it is a perfect example environment.

Now we are ready to proceed with the full development of the perceptual interface in the Macintosh Environment.

2.3 Perceptions of the Macintosh Environment

The agent perceives every screen situation (for example Figure 2-1) as a set of perceptions. Let us consider the "right" representation of the Macintosh screen. To people who are familiar with window interfaces, the natural representation of the screen is as a set of windows that contain several subparts, such as the close-box and title-bar. People who perceive a screen in this way have had some experience working with windows and have incorporated their knowledge of the function of the objects into their perception of the screen. For example, there is no a priori reason to assume that the rectangle forming the title-bar is connected to the rectangle forming the working area of a window. It could have been a separate entity floating in front of the window rectangle.

In the opposite extreme, the perceptual representation of the Macintosh Environment can give the gray-scale value of each pixel on the screen. Since there is a finite number of pixels, each with known value, this representation would be easy to implement and would give the agent a complete picture of the screen. Unfortunately, if this thesis used the pixel perceptual representation this document would never get written. It would probably take the agent longer than my lifetime simply to learn that there are rectangles on the screen. The pixel representation is not only impractical, it is far removed from how people view the Macintosh screen.

We want the agent's perceptual representation to correspond to the way people perceive the screen the first time they see a Macintosh computer. A person who has never seen a window interface would not necessarily perceive windows as the primary functional unit. Instead, the screen would appear as a collection of rectangles with properties and relationships between rectangles. Following this style of perception the agent perceives the Macintosh screen as a list of relations on the specific objects (rectangles) with specific values.

2.3.1 Objects in the Macintosh Environment

Objects in the Macintosh Environment are rectangles. There is an object corresponding to every rectangle visible on the screen. For example, the screen in Figure 2-1 will have an object for *Window 1* ("*Window 1*") as well as each of its subparts: the active-title-bar ("*Window 1* ATB"), the close-box ("*Window 1* CB"), the zoom-box ("*Window 1* ZB"), the grow-box ("*Window 1* GB"), the button ("*Window 1* Button-Dialog-Item Window 2"), and the interior area of the window ("*Window 1* Interior"). These are rectangles with unique features that are immediately recognizable as separate and significant. Any active window is comprised of these objects. Inactive windows, such as *Window 2*, are comprised of the rectangle for the complete window

("*Window 2*"), the title-bar ("*Window 2* TB"), the grow-box ("*Window 2* GB"), and the interior ("*Window 2* Interior"). These names were chosen so that people can easily interpret the learned knowledge. The names do not affect the learning algorithms in any way.

We assume that objects are recognizable across space and time. That is, if *Window 1* is partially hidden the agent can still recognize that it is the same object, because it can perceive its unique title "Window 1". Windows in the Macintosh Environments are also associated with a unique color which together with the perceptible icons make every object identifiable. In general, this assumption is reasonable except in some robotics research. As we discussed in Section 1.1, given the current types of sensors and limited perceptual processing power, robots can only identify objects in restricted domains, but not in general real-world settings.

2.3.2 Relations in the Macintosh Environment

The relations in the Macintosh Environment give information about the objects as well as relationships between objects. The relations perceived in the Macintosh Environment are summarized below.

EXISTS a binary relation on one object indicating whether the object is visible on the screen.

$$EXISTS(o) = T \text{ iff the object is visible on the screen}$$

(once the agent builds a knowledge base it may know about some objects that do not appear on the screen)

TYPE the type of an object is one of rectangle, title-bar, active-title-bar, grow-box, zoom-box, and close-box.

$$TYPE(o) \in \{REC, TB, ATB, GB, ZB, CB\}$$

The type symbols are intended to be meaningful to us, but remember that to the learning agent they are only symbols. The *TYPE* of an object corresponds to its visible icon type. For example, in the top of Figure 2-2 the close-box of *Window 2* has *TYPE CB.* The grow-box of *Window 2* has *TYPE GB*, but the grow-box of *Window 1* has *TYPE REC* because *Window 1* is not active and the grow-box icon is not present.

OV a binary relation indicating that the first argument overlaps the second argument

$$OV(o_1, o_2) = \begin{cases} T & \text{iff } o_1 \text{ overlaps } o_2 \\ F & \text{otherwise} \end{cases}$$

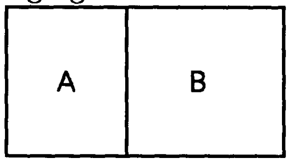
For example, in the top screen situation of Figure 2-2 it is clear that

$$\begin{aligned} OV(\textit{Window 1, Window 2}) &= F \\ OV(\textit{Window 2, Window 1}) &= T \end{aligned}$$

In the bottom situation of Figure 2-2 the overlap relationship is somewhat less clear. Since we perceive the situation such that *Window 1* overlaps *Window 2*, we define the *OV* relation as we perceive it. So

$$\begin{aligned} OV(\textit{Window 1, Window 2}) &= T \\ OV(\textit{Window 2, Window 1}) &= F. \end{aligned}$$

For A and B in the following figure



we define

$$\begin{aligned} OV(A, B) &= F \\ OV(B, A) &= F. \end{aligned}$$

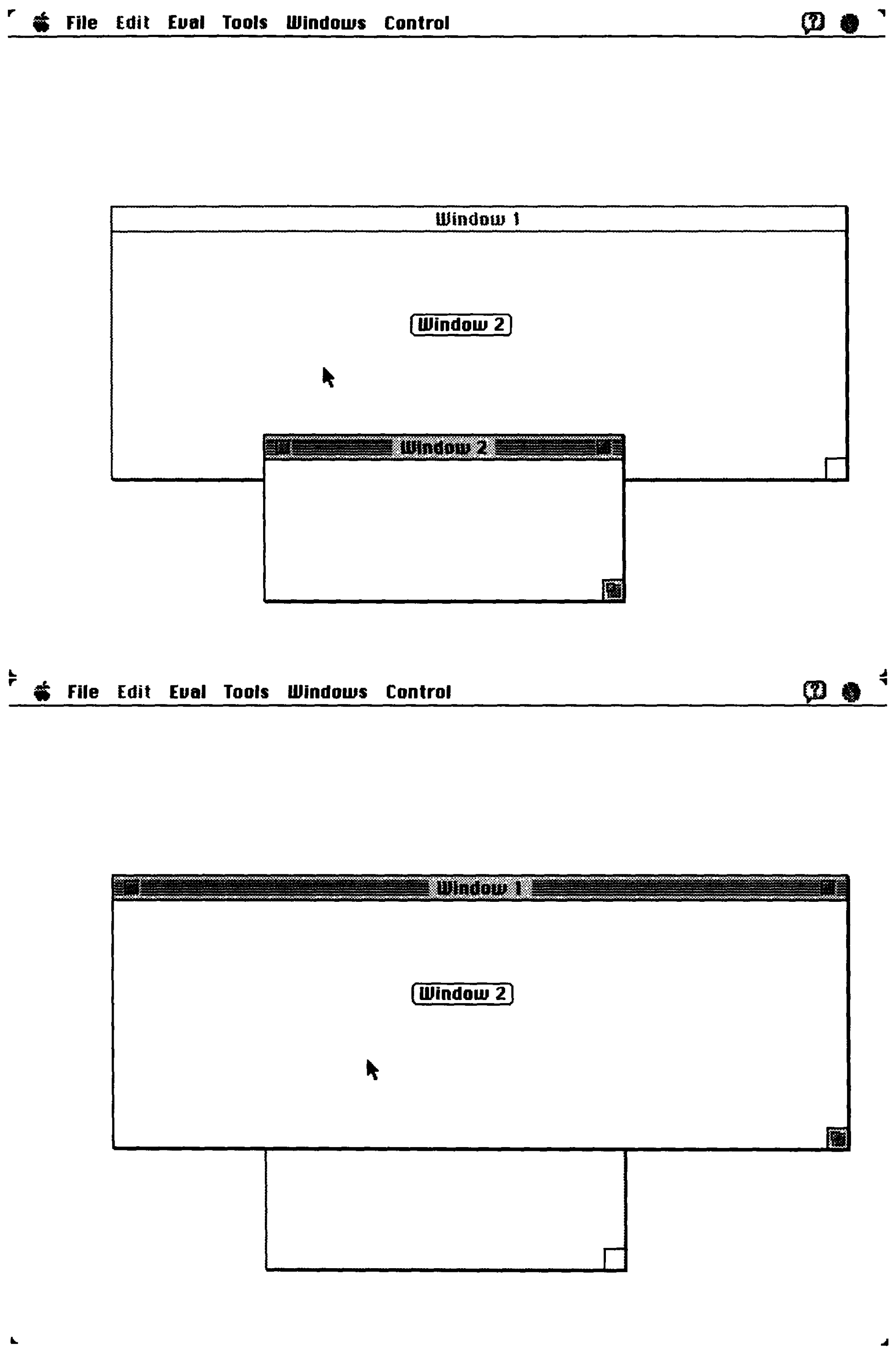

File Edit Eval Tools Windows Control
Window 1
Window 2
Window 2
File Edit Eval Tools Windows Control
Window 1
Window 2

Figure 2-2: Macintosh screen situations with overlapping windows

X the X-axis relationship of two objects

$$X(o_1, o_2) \in \{1122, 1212, 1221, 2211, 2112, 2121, 123, 132, 231, 213, 33\}$$

These symbols show the ordering of the endpoints of the two objects as we encounter them along the X (horizontal) axis moving from left to right. A "1" refers to an endpoint of the first argument (o_1), a "2" refers to an endpoint of the second argument (o_2), and a "3" refers to an endpoint of both objects (o_1 and o_2) simultaneously. For example, in Figure 2-2 *Window 1* starts to the left of *Window 2* and extends past it on the right. Moving left to right we encounter an endpoint of *Window 1* (denoted as "1"), then an endpoint of *Window 2* (denoted as "2"). Next we find another endpoint of *Window 2* and then an endpoint of *Window 1* — giving the string "1221" for the X relation. Thus

$$X(\textit{Window 1},\ \textit{Window 2}) = 1221$$

and similarly

$$X(\textit{Window 2},\ \textit{Window 1}) = 2112.$$

A value containing a "3" occurs when the two arguments have mutual endpoints. For example, the title-bar of *Window 1* in Figure 2-2 starts and ends exactly where *Window 1* starts and ends. Therefore the X relation of *Window 1* with its title bar is

$$X(\textit{Window 1},\ \textit{Window 1}\ \text{TB}) = 33.$$

Y the Y-axis relationship of two objects. Similarly to the X relation

$$Y(o_1, o_2) \in \{1122, 1212, 1221, 2211, 2112, 2121, 123, 132, 231, 213, 33\}$$

where the Y-axis extends from top to bottom (as it is defined by the Macintosh operating system.)

For example in the top screen situation of Figure 2-2

$$Y(\textit{Window 1, Window 2}) = 1212$$

$$Y(\textit{Window 2, Window 1}) = 2121.$$

In the bottom situation of Figure 2-2 *Window 2* is perceived to begin where *Window 1* ends, so their Y relation is $Y(\textit{Window 1, Window 2}) = 132$.

Notice that there is some ambiguity in the perceptions. In the bottom situation of Figure 2-2 the agent perceives the Y relation of the two windows as 132. In the true environment the Y relation of the windows may be 132 (the windows are as they are perceived), 1212 (the windows have 4 different end-points), or 312 (*Window 2* has the same start-point as *Window 1*). The definition of the X and Y relations use the perceived bounding boxes of the objects to assign a specific value. Thus $Y(\textit{Window 1, Window 2}) = 132$. It is impossible to determine, from perceptions alone, the Y relation of the windows in the true environment. This ambiguity is one of the situations where the Macintosh Environment is non-deterministic. As you recall from Chapter 1 non-determinism is acceptable in environments with manifest causal structure as long as one of the possible outcomes occurs often and the other outcomes are rare. In the example of Figure 2-2 the likely situation is that the windows have four different endpoints and the true Y relation is 1212.

There are other relations of interest in the Macintosh Environment. For example, the agent does not perceive the features of a window (close-box, zoom-box, title-bar, etc.) as *part-of* the window rectangle or even as *contained-in* the window rectangle. These two relations give more insight into the workings of the Macintosh Environment than the *overlap (OV)* relation above. Another concept of interest is the *active* window. It is obvious to those familiar with window interfaces whether a window is active or not. (In the Macintosh Environment the presence of lines in the title bar indicates an active window.) When a window is active the agent perceives that the title-bar rectangle has a different type (ATB rather than TB), but it has to learn that the type ATB means that the window is active. Chapters 4 and 5 address learning such concepts.

2.4 Summary

This chapter developed a general representation of perceptions as relations on objects. We also selected a representation for the Macintosh Environment in which the objects are rectangles and five relations ($EXIST$, $TYPE$, OV, X, and Y) on the rectangular objects describe all the relevant aspects of any screen situation. With this set of perceptions the Macintosh Environment remains complex and has manifest causal structure.

Chapter 3

Learning Rules

The goal of this thesis is to learn a causal world model in an environment with manifest causal structure. The approach is to learn specific facts first, then to use the specific knowledge to learn general concepts. Since the world model learned is a set of causal rules, the first step of learning is to find causal rules that describe effects in the environment directly from perceptions. A rule set that describes most aspects of the environment exists because the environment has manifest causal structure. This chapter presents a rule learning algorithm that converges to a set of reliable rules in environments with manifest causal structure.

The problem of learning causal rules has been addressed by several researchers in the past. Related research on rule learning, such as Drescher (1989) and Shen (1993), is discussed in Section 3.7. The difficulty of this problem stems from the abundance of perceptions, complexity of the rules, and noise in the environment or the perceptual interface. These characteristics together with the necessity to search the space of possible rules make learning hard or impossible.

The approach in this thesis overcomes these difficulties because the environment has manifest causal structure. An agent in an environment with manifest causal structure has many perceptions, but they are not low-level perceptions. As a result the learning algorithm can use its perceptions directly to form rules instead of creating higher-level perceptions or looking for hidden information. The structure of the rules is simple with this approach (see Section 3.1) because effects that are describable by the perceptions typically depend on few preconditions in the environment. Finally, although some noise is permitted in environments with manifest causal structure, the extent of the noise is restricted.

The rule-learning algorithm has the task of learning a set of reliable rules that describe characteristics in the environment. The algorithm has access to perceptions of the previous state of the screen, the agent's action, and perceptions of the new state of the screen. (This information is the algorithm's input.) The algorithm first isolates those effects in the new state that are unexpected (given the current rule base). To find a reliable rule that explains the unexpected effect, the algorithm searches for conditions in the previous state that are the causes of this effect. Section 3.3 presents the rule-learning algorithm and heuristics that reduce the time needed to search for preconditions.

The rule-learning algorithm in this chapter is proven to converge in environments with manifest causal structure (see the proof in Section 3.4). The algorithm successfully learns a set of rules that describes the Macintosh Environment. This chapter contains many examples of rules the algorithm learns, and Section 3.5 shows that the learned world model is useful for prediction and action selection.

3.1 The Structure of Rules

The world model is a set of causal rules. The number of rules in the world model is bounded by a preset parameter which is determined by the available memory and the computational speed of the computer on which the program runs. Each rule is a description of a cause and effect due to an action in the environment. A rule

$$\text{precondition} \rightarrow \text{action} \rightarrow \text{postcondition}$$

means that if the precondition is true in the current state and the action is taken then the postcondition will be true in the next state. The precondition and postcondition are a boolean combination of perceptions in the environment. (The pre- and post-conditions can be general, but this chapter considers only ground perceptual conditions.) The action can be any of the agent's actions, a general action, or

NOACTION. A NOACTION rule means that whenever the precondition is true in the environment the postcondition is also true in the environment.

A rule, r, **applies** in state S and action a (denoted $applies(r, S, a)$) when its preconditions are true in state S and its action is a. A rule, r, **predicts correctly** from state S_1 and action a to state S_2 if it $applies(r, S_1, a)$ and its postcondition is true in state S_2. Since environments with manifest causal structure are not necessarily deterministic, a reliability measure is associated with every rule. The **reliability** of a rule is its empirical probability of predicting correctly.

One of the difficulties in learning rules is the generality of preconditions and postconditions. Theoretical machine learning shows that general boolean combinations are not efficiently learnable (Kearns & Vazirani 1994), which indicates that if the precondition and postcondition may be any boolean combination of the perceptions the rules are hard to learn. We restrict the descriptive power of rules, intending that this restriction result in fast learning. The restrictions on the structure of the rules are derived from the logical structure of preconditions and postconditions, and from the assumption that the environment has manifest causal structure.

First consider the logical restrictions on postconditions. We are interested in predictive rules. In other words, when a rule applies in the current state and action we want it to give a definitive condition to predict. Therefore we do not want postconditions to contain negated conditions, such as *"the rectangle is not a close-box"* where the rectangle may still be one of many types. Similarly we do not want disjunctive postconditions, such as "either *Window 1* is visible or *Window 2* is visible", because we cannot know which condition is true from such rules. Although such rules can be valid and testable, using them to predict means solving a GRE style logic problem — a difficult task even for people. Thus the postcondition is restricted to a conjunction of positive perceptions. Notice also that a rule

$$P \rightarrow A \rightarrow C_1 \wedge C_2 \wedge C_3$$

is equivalent to the rules

$$P \rightarrow A \rightarrow C_1$$

$$P \rightarrow A \rightarrow C_2$$

$$P \rightarrow A \rightarrow C_3.$$

Whenever the former rule applies, all three of the latter rules apply, and both the former and the latter predict the same conditions (C_1, C_2, and C_3). The reverse is also true: whenever any of the bottom rules apply, all the bottom rules apply as does the top rule. So the postcondition can consist of one positive condition without loss of generality.

Although it may seem wasteful to create multiple rules when one rule suffices, the restriction to postconditions of one condition allows for a simple learning algorithm (see Section 3.3). Chapter 4 describes a way to find conditions that are correlated in the environment, much like the correlation of C_1, C_2, and C_3, which enables the agent to collapse the three bottom rules above to one rule while remaining within the one postcondition requirement.

With regard to the precondition structure, since the rules should be expressive enough to describe complex environments, we want the precondition of rules to be as general as possible. We can make some simplifications without losing descriptive power. Consider a disjunctive precondition

$$P_1 \vee P_2 \rightarrow A \rightarrow C.$$

This rule can be replaced by the rules

$$P_1 \rightarrow A \rightarrow C$$

$$P_2 \rightarrow A \rightarrow C$$

without loss of generality. When a negated condition appears in a precondition it can be replaced by exhaustive enumeration of the alternatives (since we assume that the set of possible values for any relation is finite). So the rules are restricted to a precondition which is a conjunction of positive conditions and a postcondition which is one condition.

Finally, the assumption that the agent-environment interface has manifest causal structure implies that the agent perceives a fairly complete description of the state including many details. When an event happens in the environment we expect the conditions relevant to causing this event to be local to the observed event. Thus the number of relevant conditions is probably small compared with the total number of perceptual inputs. In fact for most rules in most environments I believe the number of relevant conditions to any effect is very small indeed. (For example, in the Macintosh Environment two preconditions suffice.) Although the precondition is not restricted to any predetermined number of conditions, the rule-learning algorithm will use this observation and try to explain events with the simplest possible rules.

To summarize the above discussion, a ground rule has the form

$$\text{precondition} \rightarrow \text{action} \rightarrow \text{postcondition}$$

The precondition is a conjunction of perceptions and the postcondition is a single perception. The perceptions have the value of the input relations or the value NP which stands for "not-perceptible". (The not-perceptible value, NP, is the value of any relation on objects that do not appear on the screen, or an unknown value for a relation.) The action, for the purposes of this chapter, is any specific action or NOACTION.

Let us consider some example rules. In the Macintosh Environment the algorithm should learn rules such as

$$\textit{Window 2} \text{ covers } \textit{Window 1} \rightarrow \text{click-in } \textit{Window 1} \rightarrow \textit{Window 1} \text{ is fully visible}$$

for the transition of the screen situations in Figure 3-1. The description of this rule is high level, and uses term such as *covers* and *visible* that are not part of the agent's perceptions. The rule that the agent learns will be expressed in terms of its perceptions as

$$OV\ (\textit{Window 2},\ \textit{Window 1}) = T$$
$$\rightarrow \text{click-in } \textit{Window 1} \rightarrow OV\ (\textit{Window 1},\ \textit{Window 2}) = T.$$

For this situation there will be a dual rule

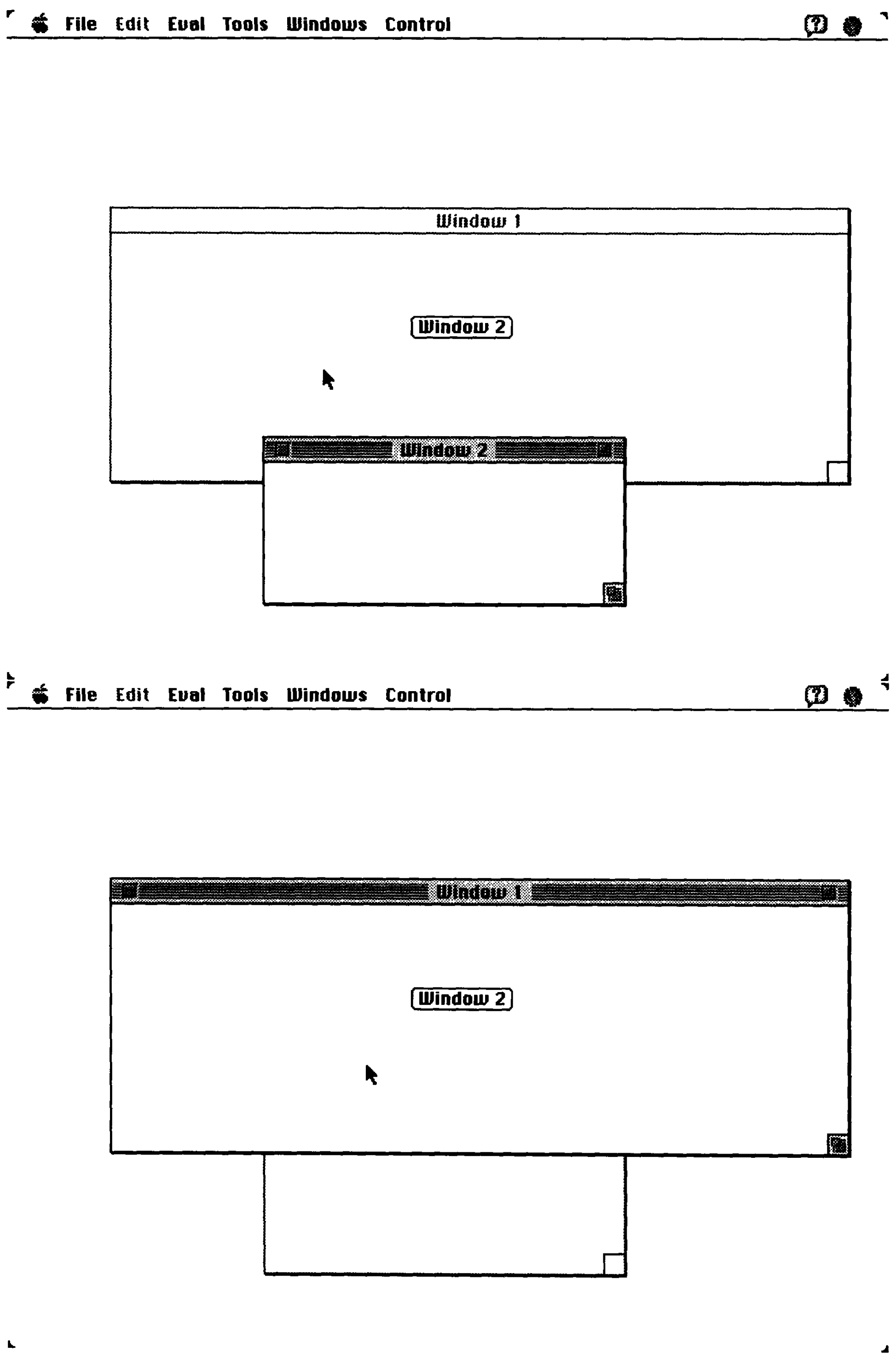

File Edit Eval Tools Windows Control
Window 1
Window 2
Window 2
File Edit Eval Tools Windows Control
Window 1
Window 2

Figure 3-1: Macintosh screen situations with overlapping windows

$$OV\ (\mathit{Window\ 2},\ \mathit{Window\ 1}) = T$$
$$\rightarrow \text{click-in } \mathit{Window\ 1} \rightarrow OV\ (\mathit{Window\ 2},\ \mathit{Window\ 1}) = F$$

and some rules to explain the disappearance of the parts of the window (the close-box, zoom-box, and title-bar.) These rules have the form

$$() \rightarrow \text{click-in } \mathit{Window\ 1} \rightarrow \mathit{EXIST}\ (\mathit{Window\ 2}\ \text{CB}) = \mathit{NP}$$

No precondition is needed for this rule because the close box disappears whenever a window is not active.

The remainder of this section discusses the algorithm to learn such rules.

3.2 World Model Assumptions

In this section we introduce two assumptions about the environment. These assumptions affect the world model that the agent constructs thereby influencing the learning algorithm as well as algorithms that use the learned model, such as the prediction algorithm. The world model uses the following assumptions about the environment.

1. Perceptions persist unless there is some rule that states otherwise. (Objects also persist because perceptions of the EXIST relation persist.) The persistence assumption means that the agent does not have to learn and store rules for situations where nothing changes, such as clicking in the active window.

2. If an object is not perceptible, all the relations on it are not perceptible.

Thus, for example, if *Window 2* title-bar is not perceptible, as in the bottom of Figure 3-1, the type of this title-bar is not perceptible. The first assumption implies, for example, that starting in the situation in the top of Figure 3-1 following a click in *Window 2* there will be no events to explain because there is no change to the environment.

Now let us turn to the rule-learning algorithm.

3.3 The Rule-Learning Algorithm

The goal of the rule-learning algorithm is to learn a set of specific rules that are valid in the environment. The algorithm uses a generate and test methodology to find valid rules. It begins with an empty set of rules (no a priori knowledge) and uses its observations to generate rules and to test if they predict correctly. The evolving nature of the rule set is reminiscent of classifier systems and the genetic algorithm (Holland 1976).

The rule-learning algorithm executes at every trial, namely after each action the agent takes. (Recall from Chapter 2 that this algorithm assumes that the learner is the only actor in the environment.) After every action the agent takes, the Macintosh screen changes. The learning algorithm uses the before and after screen situations to learn the effects of the action. The perceptions of the screen before the action are stored as the **previous-perceptions** and the perceptions of the screen following the action are saved as the **current-perceptions**. The action is saved as the **current-action**. These variables are inputs to the rule-learning algorithm.

The world model is the output of the learning algorithm. It is also an input to the learning algorithm which checks to see if an effect of the action is explained by the current world model. Naturally, we do not want the learning algorithm to spend time explaining effects that are already understood.

Figure 3-2 contains the outline of the rule-learning algorithm. The algorithm has two main parts: (1) removing and reinforcing rules and (2) creating new rules.

The rule-learning algorithm reinforces every rule at every trial. Section 3.3.3 discusses the evaluation of rules. To create new rules the algorithm finds perceptions that are different in the current-perceptions from the previous-perceptions and that are not explained by any existing rule. If an existing rule already explains an effect in the environment, there is no need for further explanation. The agent also does not explain perceptions that do not change following the action because the world model assumes that perceptions persist. Once the agent finds a perception to explain it creates a new rule. The method for creating rules is given in Section 3.3.2. The

Algorithm 2 Learn()

```
[remove and reinforce rules]
for each rule r
    Probabilistic-Rule-Reinforce(r).

[create new rules]
let different-perceptions = perceptions that are different in current-perceptions
    from previous-perceptions.
    for each different-perception in different-perceptions
    if the different-perception is not explained by some rule
        if current-trial $\leq$ MakeNOACTIONRulesThreshhold then
            make a new rule to explain different-perception with NOACTION
        else if the different-perception is explained by some NOACTION rule whose
            precondition is in different-perceptions then
            remove the different-perceptions from different-perceptions
            else make a new rule to explain different-perception
            with action current-action
```

Figure 3-2: Outline of the **Learn** Algorithm

Learn algorithm also learns and uses NOACTION rules to learn the world model. The following section examines NOACTION rules and their effect on the learning algorithm.

3.3.1 NOACTION Rules

Recall that the purpose of NOACTION rules is to express a correlation among perceptions. These rules indicate that a perception is always true when another perception is true. For example, consider Figure 3-3 where *Window 1* disappears following a click-in *Window 1*'s close-box. This action causes many changes. *Window 1* disappears, as does the close-box, zoom-box, interior, title-bar, etc. All these changes must be explained and they are not independent. If the algorithm succeeds in correlating the existence of the window parts, then one rule that explains the disappearance of the window due to clicking the close-box suffices. NOACTION rules provide a means of learning the correlated perceptions. Some examples of valid NOACTION rules in the Macintosh Environment are

$$EXIST\ (Window\ 1) = T \rightarrow \text{NOACTION} \rightarrow EXIST\ (Window\ 1\ \text{INTERIOR}) = T$$

$$EXIST\ (\mathit{Window\ 1}\ \mathrm{ATB}) = T \rightarrow \textsc{noaction} \rightarrow EXIST\ (\mathit{Window\ 1}\ \mathrm{CB}) = T$$

and

$$EXIST\ (\mathit{Window\ 1}\ \mathrm{ATB}) = T$$
$$\rightarrow \textsc{noaction} \rightarrow OV\ (\mathit{Window\ 1}\ \mathrm{ATB},\ \mathit{Window\ 1}) = T.$$

These rules describe that when *Window 1* is present so is its interior, when the active-title-bar of *Window 1* exists so does the close-box, and whenever the active-title-bar of *Window 1* exists it overlaps *Window 1.*

As a result of learning NOACTION rules the number of rules in the world model is reduced. Consider an environment in which perception C_2 is true whenever perception C_1 is true, and the following rules are true

$$P_1 \rightarrow A_1 \rightarrow C_1$$

$$P_1 \rightarrow A_1 \rightarrow C_2$$

$$P_2 \rightarrow A_2 \rightarrow C_1$$

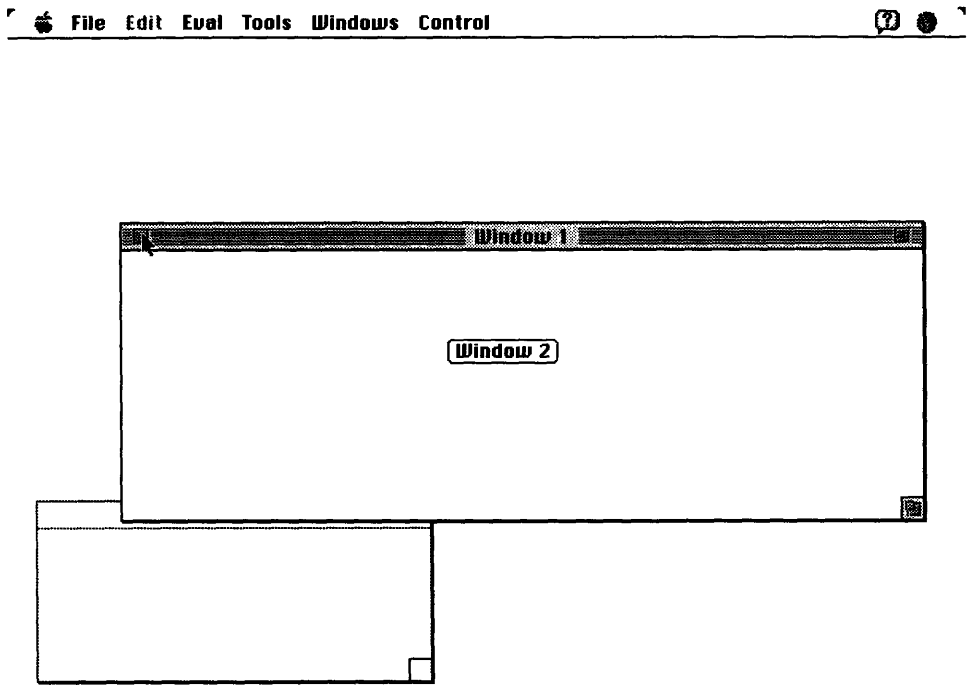
$$P_2 \rightarrow A_2 \rightarrow C_2.$$

Because C_2 is true whenever C_1 is true, the NOACTION rule

$$C_1 \rightarrow \textsc{noaction} \rightarrow C_2$$

is true in the environment. This rule makes the second and the fourth rules above redundant because the effects that they predict are predictable from the first rules with the NOACTION rule and from the third rule with the NOACTION rule respectively. The revised set of rules is more concise in capturing the characteristics of the environment.

The rule-learning algorithm in Figure 3-2 tries to learn the shorter world model which uses NOACTION rules. In the second part of the algorithm, it finds perceptions that change in the environment due to the current-action and are not explained by any rule. The algorithm first tries to create NOACTION rules to explain the perceptions.



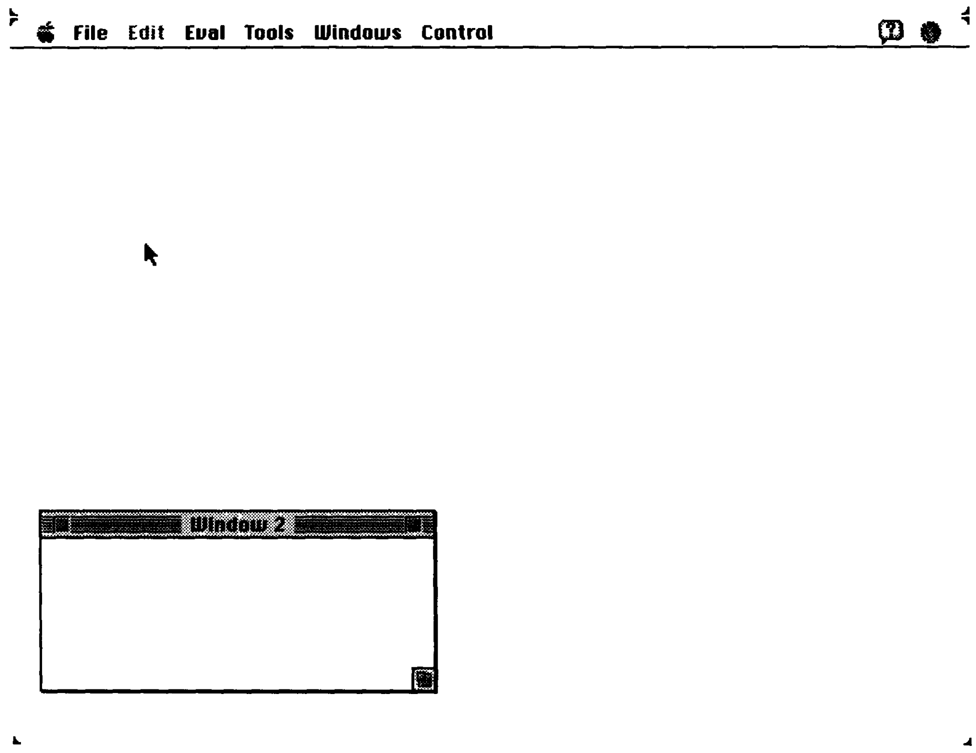


Figure 3-3: Macintosh screen before and after a click in *Window 1* close-box

(The length of time that the algorithm spends creating NOACTION rules is determined by the parameter MakeNOACTIONRulesThreshhold. The algorithm must use this control strategy because the agent has no way of knowing when it has learned all the correct NOACTION rules.)

After the algorithm has spent some time learning NOACTION rules it creates rules that describe the effects of actions. The algorithm uses the NOACTION rules it learned to reduce the number of perceptions for which it creates rules. Let us return to our example above. Suppose that the algorithm has already learned the rule

$$C_1 \rightarrow \text{NOACTION} \rightarrow C_2$$

and now finds that perceptions C_1 and C_2 change due to action A_1. Since the perception C_2 is explained by the NOACTION rule the algorithm only creates rules for the perception C_1 with action A_1. Now the algorithm must find the right set of preconditions to make a correct rule which is the topic of the next section.

3.3.2 Creating New Rules

The task that the algorithm to create rules faces is: given a postcondition and an action, to find a precondition (conjunction of perceptions in the previous-perceptions) such that the resulting rule is valid. To create NOACTION rules the algorithm uses the current-perceptions instead of previous-perceptions, but the algorithm to find preconditions is otherwise unchanged. Since the agent does not have an oracle or teacher to help it find a good precondition it can either enumerate all the possible preconditions or guess at the right preconditions.

Enumerating the possible preconditions from the list of perceptions is straightforward if the agent has enough space to create all possible rules and enough time to check if these rules are reliable. Since the set of possible preconditions is the power set of the previous-perceptions, its size is exponential in the number of perceptions. Therefore, the algorithm cannot create all the possible rules. For example, in the Macintosh Environment with two windows the number of perceptions can be as high as 420. The size of the power set is 2^{420} which is clearly too large to enumerate. Even if the number of preconditions is bounded by a constant (as I believe it is for most environments), the number of possible rules is too large to create all the rules. In the Macintosh Environment, if the number of preconditions is restricted to two, then the set of possible preconditions has size 88411 — which includes the possible rules for one changed perception out of 420 possible perceptions.

Therefore, the algorithm creates a few rules to attempt to explain one situation at a time (typically between 1 and 10 new rules). If these rules are not reliable the algorithm will have the opportunity to create additional rules when this effect occurs again. The advantage of this approach is that the size of the rule set remains manageable. The disadvantage is that unreliable rules may be created repeatedly, but such rules are removed quickly. Once the algorithm finds reliable rules it does not create additional rules.

As a baseline strategy for finding preconditions, the algorithm selects at random from the list of previous-perceptions. A few strategies improve the algorithm's chances of picking good preconditions. These heuristics are described in the next four sections.

Learn Simple Rules First

When creating a rule to explain the postcondition, the algorithm does not know the necessary number of preconditions to make a reliable rule. Rather than create rules with a large number of preconditions, the algorithm tries simple rules first. It creates rules with no preconditions first. When it it has spent some time creating rules for an effect with no preconditions and has not been able to explain the effect it creates rules with one precondition, then two preconditions, and so on. The length of time (number of trials) that the algorithm spends creating rules with zero preconditions, one precondition, two preconditions, etc. depends on parameters but contains a random element. Thus early on the algorithm creates rules with zero preconditions only. Later it creates rules with more and more preconditions but occasionally it makes rules with fewer preconditions. This strategy of enumeration is commonly used in computer science algorithms and saves this algorithm both time and space.

Since there is a smaller number of rules with few preconditions the algorithm creates fewer rules and spends less time creating and evaluating rules. For example, in Figure 3-1 following a click-in *Window 1* action the postcondition $EXIST$ (*Window 1* CB) $= T$ is explained by the rule

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow EXIST\ (\textit{Window 1}\ \text{CB}) = T$$

with no preconditions. The learning algorithm finds this rule immediately when using the strategy outlined above. It creates no additional rules and does not spend any extra time explaining the postcondition.

Use the Same Relation

Most of the time the cause of a change is local. For example, if following a click in *Window 1* the condition OV (*Window 1*, *Window 2*) changes from F to T, then the fact that *Window 1* was under *Window 2* is more relevant than the fact that *Window 2* has a close-box. The algorithm, therefore, has a higher probability of creating rules with preconditions that have the same relation and objects as the postcondition with the value of the relation on these objects in the previous perceptions. The probability of picking the same relation is determined by a parameter. In this example, the algorithm tries the conditions OV (*Window 1*, *Window 2*) and OV (*Window 2*, *Window 1*) with higher probability than other perceptions.

Focus Attention

The algorithm also keeps the size of the rule set small by trying to learn one relation at a time. In the Macintosh Environment it concentrates on learning the $EXIST$ relation first, then the $TYPE$ relation, the OV relation, the X relation, and the Y relation. (This ordering of relations is imposed because an understanding of the $EXIST$ relation is instrumental for predicting the other relations. For example, whenever a close-box is present it has type close-box. To use this simple rule to predict the type of a close-box the agent must be able to predict that the close-box is present. The order of learning the remaining relations is arbitrary.) Naturally, as the algorithm learns each additional relation the number of rules increases. The number of rules describing effects on a relation, however, is typically smaller when the effects are understood than the number of rules maintained as hypotheses during learning.

Mysteries

A mystery is an effect the agent sees which it finds surprising enough to spend extra effort to explain. When Newton's apple (allegedly) fell from the tree, Newton found this event both surprising and interesting. He spent much effort to think and re-think this event until he understood why the apple fell. The mystery heuristic mimics the process of learning by re-playing events in the learner's mind. When the agent encounters a surprising effect it saves the relevant data (the previous state, action, and postcondition). Later, the agent repeatedly creates rules to explain this event, until a reliable rule explains the event.

One of the difficulties with using mysteries is that the agent does not know if an event is rare — a mystery — when it observes the event. The agent must decide if the event is sufficiently important to save based on some tangible measure. The learning algorithm uses a measure of surprise which depends on the rules explaining this event, or lack thereof.

The definition of a surprising event depends on the environment. When the environment is easy to learn and all situations are equally likely, an event must be very surprising to become a mystery. The algorithm may require that there are no *potential* rules explaining an event to make a mystery. In other cases, the requirement may be that there are no *reliable* rules to explain the event.

This criterion for mysteries does not ensure that the saved events are very rare. Some saved events may not be rare, especially early in the learning process when all events are surprising. As the world model improves, however, many events will be explained. Thus the surprising events found later on are likely to be true mysteries. Any frequently occuring event that was saved as a mystery will also be explained quickly leaving the agent with a set of mysteries.

The agent tries to explain the mysteries periodically. The interval between successive explanations depends on a preset parameter — typically every 100 trials. At this time, the agent checks if the mysteries are explained, removes the explained mysteries, and ranks the remaining mysteries according to their measure of surprise. The agent then re-plays the most surprising of these events, i.e., it sets the previous perceptions to the mystery's previous perceptions and the current action to the mystery's action and then creates rules to explain the mystery's postcondition. Again, the number of replayed mysteries depends on a parameter. For the Macintosh Environment the algorithm replays 10 mysteries.

Mysteries are particularly useful in environments with a few rare events. That is, environments where most situations are encountered often, but a few situations occur infrequently. The Macintosh Environment is not one that has many rare events so the following example is concocted but possible. Consider a screen situation with ten windows where each window has a button which pops the next window up, and this button is the only way to bring up the windows. If the agent takes random actions, then window 10 will rarely be present. The agent will learn more quickly by re-playing this situation as a mystery than by waiting for the situation to occur again. When there are no rare events in the environment, mysteries still appear to speed the learning of rules somewhat.

In summary, the rule-creation algorithm uses some effective heuristics to guess the preconditions for a rule. The algorithm does not guarantee that the rules it creates are valid; they are only guesses. Since there are few valid rules compared with the total number of rules, most rules that the algorithm creates are not valid. For this reason new rules are put on **probation** initially and are not considered part of the world model until they are taken off probation by the evaluation algorithm.

3.3.3 Reinforcing Good Rules and Removing Bad Rules

The objective of the evaluation algorithm is to determine if rules are valid. A rule is **valid** if its true probability of predicting correctly[1] is above a predetermined threshold. The difficulty of determining if a rule is valid is that the learner does not know the true probability of predicting correctly. Instead the algorithm must use the empirical reliability of a rule to determine if it is valid. The process of evaluating rules uses statistical methods that take into consideration the error from using the empirical rather than the true measure of reliability.

To determine the reliability of a rule the agent tests rules in the environment. Recall that rules are predictive. A rule

$$\text{precondition} \rightarrow \text{action} \rightarrow \text{postcondition}$$

means that if the preconditions are true in one state and the action is taken, then the postcondition will be true in the next state. The algorithm evaluates the reliability of rules based on their predictive ability.

Consider the rule

$$() \rightarrow \text{click-in } \textit{Window 1} \text{ CB} \rightarrow \textit{EXIST}\ (\textit{Window 1}) = \textit{NP}.$$

This rule applies whenever the agent's last action was a click in *Window 1*'s close-box. The above rule predicts correctly if *Window 1* disappears in the next state. This rule is a *perfect predictive rule* since a window always goes away following a click in its close-box.

Prediction is a way of estimating a rule's probability of predicting correctly. We define a valid rule as having probability of predicting correctly above threshold Θ. Rules with probability of predicting correctly above this threshold are considered valid and rules with lower probability of predicting correctly are not valid. The value of the threshold Θ depends on the environment. In deterministic environments the threshold

[1]Note that a rule's probability of predicting correctly may depend on the sequence of actions the agent takes (e.g., the agent may not explore some states of the environment). Thus this measure is not always well-defined. This learning algorithm, however, selects actions at random with equal probability of taking any action from any state. Thus the probability of predicting correctly for any rule is well-defined (see Section 3.4 for a detailed discussion of this issue).

is 1, because all the reliable rules should be perfect predictors. In environments with manifest causal structure, the threshold depends on the degree of non-determinism of the environment.

Consider first the simpler case of deterministic environments. Since reliable rules for such environments are perfect predictors, these rules never make a prediction error. A rule can therefore be removed as soon as it predicts incorrectly. This strategy is implemented in algorithm **Deterministic-Rule-Reinforce** below. (The algorithm stores the number of times each rule, r, applies in *apply(r)* and the number of correct predictions in *success(r)*. The reliability of a rule $reliability(r) = success(r)/apply(r)$. The functions *precondition(r)*, *action(r)*, and *postcondition(r)* refer to the precondition, action, and postcondition of r respectively.) This rule-reinforcement algorithm executes for every rule at every trial, i.e., after every action the agent takes. The previous-perceptions are the perceptions prior to taking the action and the current-perceptions are the perceptions following the action.

Algorithm 3 Deterministic-Rule-Reinforce*(r)*

```
if r is a NOACTION rule then
    let prev-perceptions = current-perceptions
    else let prev-perceptions = previous-perceptions
if applies(r, prev-perceptions, current-action) then
    increment apply(r)
    if postcondition(r) is true in the current-perceptions
        then increment success(r)
        else remove r.
```

If the environment is non-deterministic the algorithm estimates the reliability of rules, but the estimated reliability is not necessarily equal to the rule's true probability of predicting correctly. The estimated reliability does not guarantee that the rule's true probability of predicting correctly is above or below the threshold. To determine if the rule is above or below the threshold, with high probability, the algorithm uses the sequential ratio test (Wald 1947). Algorithm **Probabilistic-Rule-Reinforce** describes the rule evaluation algorithm for non-deterministic environments. The se-

Algorithm 4 Probabilistic-Rule-Reinforce*(r)*

```
if r is a NOACTION rule then
    let prev-perceptions = current-perceptions
    else let prev-perceptions = previous-perceptions
if applies(r, prev-perceptions, current-action) then
    increment apply(r)
    if postcondition(r) is true in the current-perceptions then
        increment success(r)
    let test = SequentialTest(apply(r), success(r), $p_0$, $p_1$, $\alpha$(current-trial), $\beta$(current-trial))
        if test = accept then
            remove r from probation
            reset apply(r) and success(r) to 0.
        if test = reject then
            remove r.
```

quential ratio test is described in the next section.

Notice that algorithm **Probabilistic-Rule-Reinforce** repeatedly tests rules, rather than testing a rule once and either accepting or rejecting. Rules must be tested repeatedly because there is a small probability that the sequential ratio test will accept a bad rule (as we will see in the following section). Re-testing rules is necessary for convergence to a good world model (see Section 3.4).

The Sequential Ratio Test

The sequential ratio test determines, with high probability, if the estimated error probability of a rule is above or below a threshold. In algorithm **Probabilistic-Rule-Reinforce**, let p_1 be $1-$ the value of the threshold, $(1 - \Theta)$, and let p_0 be a smaller value (e.g., $p_1 = 0.1$ and $p_0 = 0.05$). The parameters α and β determine the probability of misclassifying a rule as reliable or not. In algorithm **Probabilistic-Rule-Reinforce** α and β become smaller with time, specifically $\alpha(t) = \beta(t) = \frac{1}{2^{2\lceil \log t \rceil}}$. Note that $\alpha(t)$ and $\beta(t)$ are not recomputed after every trial, only after increasing intervals, and the probability of making mistakes goes to zero with time.

The details of the sequential ratio test as given by Wald (1947) are as follows.

The Problem Given a coin with unknown probability of failure p.

Test if $p \leq p_0$ vs. $p \geq p_1$, accept if $p \leq p_0$, reject if $p \geq p_1$.

Requirements The probability of rejecting a coin does not exceed α whenever $p \leq p_0$, and the probability of accepting a coin does not exceed β whenever $p \geq p_1$.

The Test Let m be the number of samples $(apply(r))$, and f_m be the number of failures in m samples $(apply(r) - success(r))$.

if

$$f_m \geq \frac{\log \frac{1-\beta}{\alpha}}{\log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0}} + m \frac{\log \frac{1-p_0}{1-p_1}}{\log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0}}$$

reject.

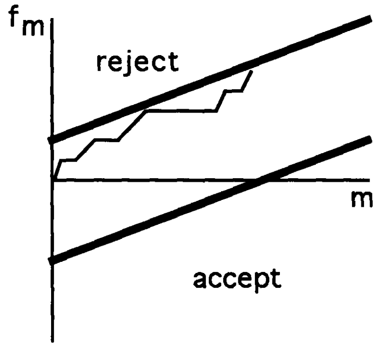
if

$$\frac{\log \frac{\beta}{1-\alpha}}{\log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0}} + m \frac{\log \frac{1-p_0}{1-p_1}}{\log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0}} \leq f_m$$

accept.

Otherwise, draw another sample.

This test defines two lines with different intercepts and the same slope, where the area above the first line is a reject region and the area below the second line is the accept region (as shown in the figure below). The test is a random walk which terminates when it reaches the reject or accept regions.



Testing the rules using the sequential ratio test is efficient since the accept and reject regions are defined once the parameters p_0, p_1, α, and β are set. The algorithm pre-computes a table which determines acceptance (and rejection), given the values of $success(r)$ and $apply(r)$, in one step.

Observe that the **Probabilistic-Rule-Reinforce** algorithm repeatedly tests rules (even if they are off probation). Rules must be tested repeatedly because the sequential ratio test has a small probability of accepting an invalid rule. The test likewise has a small probability of rejecting a valid rule and when such rules are removed we hope that the rule-creation algorithm will re-create the rule or a similar rule to explain the same effect. The probability of making mistakes of this kind decreases with time (because the parameters α and β depend on the current-trial.) The importance of this issue will become clear as we go through the convergence result in the next section.

3.4 Rule Learning Converges

This section proves that the rule-learning algorithm converges to a good model of the environment. Before proceeding with the proof, the notion of an environment with manifest causal structure is defined as well as the model of the environment which the learning algorithm is aiming toward. For simplicity, we prove that the learning algorithm converges in deterministic environments with manifest causal structure first (in Section 3.4.1). Section 3.4.2 proves the general convergence result for any environment with manifest causal structure.

3.4.1 Convergence in Deterministic Environments

A deterministic environment is essentially a finite automaton. There are known algorithms for learning finite automata (see, e.g., Rivest & Schapire (1989) and Rivest & Schapire (1990)). This section presents the convergence proof in deterministic environments to prepare the reader for the more complex proof of convergence in probabilistic environments. We first define terms such as deterministic environments with manifest causal structure and the goal world model. Then we prove convergences to the goal model in deterministic environments with manifest causal structure.

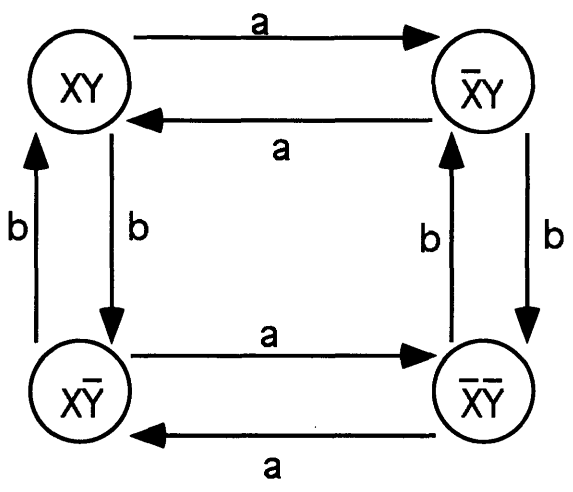


Figure 3-4: A deterministic environment

Definition of Deterministic Environments with Manifest Causal Structure

Consider the deterministic environment in Figure 3-4. There are two binary relations of 0 arguments, X and Y. The perception $X() = T$ is abbreviated as X, and $X() = F$ is abbreviated $\overline{X}$. Similarly, $Y() = T$ is Y, and $Y() = F$ is $\overline{Y}$. The agent in the environment of Figure 3-4 has two actions, a and b, where a toggles X and b toggles Y. The states of the graph are subsets of the perceptions X, Y, $\overline{X}$, and $\overline{Y}$, and there is one transition from every state on every action, a and b. These two observations define a deterministic environment with manifest causal structure.

Definition 1 *A* **deterministic environment with manifest causal structure** *is a connected graph where the nodes are states (subsets of perceptions) and there is exactly one directed arc for each action from every state.*

Definition of a Goal World Model

We have discussed the structure of the world model extensively. Recall that the world model is a set of rules. In this section we discuss a model of a particular action and postcondition pair $\xrightarrow{A} C$. The world model of the whole environment is a collection of models of $\xrightarrow{A} C$ for every action A and postcondition C. The following definition of a model holds for any environment (not only deterministic environments).

Definition 2 *A* **model** *of an (action, postcondition) pair ($\xrightarrow{A}C$) in an environment is a set of rules, R. Each rule in R has the form $r_i = p_i \xrightarrow{A} C$, where p_i is a conjunction of perceptions of the environment.*

The learning algorithm aims toward learning a complete-model of its environment.

Definition 3 *A* **complete-model** *of $\xrightarrow{A}C$ in a deterministic environment contains all the rules with action A and postcondition C that are true for the environment. (I.e., in any state where the rule's preconditions are true the arc on action A leads to a state where condition C is true.)*

In the environment in Figure 3-4, the complete-model for $\xrightarrow{a}X$ contains the rule

$$\overline{X} \rightarrow a \rightarrow X.$$

Similarly, the complete-model for $\xrightarrow{b}\overline{Y}$ contains the rule

$$Y \rightarrow b \rightarrow \overline{Y}.$$

The complete-model for $\xrightarrow{a}X$ also contains rules such as

$$\overline{X}Y \rightarrow a \rightarrow X$$

that are extraneous. Such rules do not add new information; rather, they are more specific than some other valid rule. To avoid learning such specific rules and other rules that do not add new information the learning algorithm's true aim is to learn a model that is predictively-equivalent to the complete-model, not the complete-model itself. The following definition of predictively-equivalent models holds for all environments.

Definition 4 *One model of $\xrightarrow{A}C$, R_1,* **predictively-implies** *model R_2 of $\xrightarrow{A}C$ ($R_1 \Longrightarrow R_2$) if $\forall r_i^1 \in R_1$, $r_i^1 = p_i^1 \xrightarrow{A} C$ and in every state where p_i^1 is true at least one of the p_j^2's is true (denoted $p_i^1 \Rightarrow p_1^2 \vee p_2^2 \vee \ldots$) where p_j^2 are the preconditions of rules in R_2.*

Definition 5 *Two models of $\xrightarrow{A}C$, R_1 and R_2, are* **predictively-equivalent** *($R_1 \equiv R_2$)* **iff** *$R_1 \Longrightarrow R_2$ and $R_2 \Longrightarrow R_1$.*

Proof of Convergence in Deterministic Environments

This section proves Theorem 1 which states that the rule-learning algorithm converges in deterministic environments with manifest causal structure. We assume the learning algorithm uses the **Deterministic-Rule-Reinforce** algorithm to evaluate rules. The proof shows that the algorithm converges to a model that is predictively-equivalent to the complete-model.

The theorem requires that every rule in the complete-model is exercised infinitely often and every rule not in the complete-model is violated infinitely often. This requirement guarantees that the agent has the opportunity to explore enough of its environment so that it can learn the world model. An agent obviously cannot learn about a five room house if it never goes outside of the bathroom. One way to satisfy this condition is to select random actions.

Theorem 1 *In a deterministic environment with manifest causal structure, if every rule in the complete-model of* $\xrightarrow{A} C$ *is exercised infinitely often and every rule not in the complete-model of* $\xrightarrow{A} C$ *is violated infinitely often, the learning algorithm with the* **Deterministic-Rule-Reinforce** *procedure will converge to a model of* $\xrightarrow{A} C$ *that is predictively-equivalent to the complete-model of* $\xrightarrow{A} C$*.*

Proof: (1) The learning algorithm converges.

Since the environment has manifest causal structure, there is a finite number of perfect predicting rules for $\xrightarrow{A} C$.

At any time that the set of rules the algorithm learned does not explain some situation there is a non-zero probability that the algorithm will create one of the perfect predicting rules.

This process will stop when the algorithm has all the perfect rules, or all situations are explained. At this time no new rules are created. All the perfect rules will be kept indefinitely, and the imperfect ones will be removed eventually.

(2) The complete-model of $\xrightarrow{A} C$, RC, and the learned model, RL, are predictively-equivalent.

Show that $RC \Longrightarrow RL$.

For any rule $r^c = p^c \xrightarrow{A} C$ in RC

case 1 $r^c \in RL$. Then clearly $p^c \Rightarrow \bigcup_i p_i^l$ (the union of all preconditions in RL), because p^c is equal to one of the p_i^l's.

case 2 $r^c \notin RL$. If r^c was created it would never be removed since it would never make a mistake. So r^c was not created. There are two reasons for not creating a rule:

- $\xrightarrow{A} C$ was explained by some other rule every time r^c applied. In this case $p^c \Rightarrow \bigcup_i p_i^l$.
- The rule creation algorithm did not select the preconditions p^c. But with infinitely many repetitions of $p^c \xrightarrow{A} C$, and random precondition selection invoked infinitely often, p^c would be selected eventually with probability 1.

So $RC \Longrightarrow RL$.
Now show that $RL \Longrightarrow RC$.
We need to show that for any rule $r^l = p^l \xrightarrow{A} C$ in RL it must be the case that $r^l \in RC$.
Assume to the contrary that $r^l \notin RC$. To create r^l the learner must see examples where $p^l \xrightarrow{A} C$ is true, and where no rules explain these situations. Although situations where $p^l \xrightarrow{A} C$ is true are seen infinitely often, as more and more perfect predictors are found these situations will be explained.
So an imperfect predictor like r^l may be created early on, but it will be removed because it will be violated infinitely often. When the learning algorithm converges $\xrightarrow{A} C$ will be explained and r^l will not be created again. This argument shows that r^l cannot be an imperfect predictor, so $r^l \in RC$, and clearly $p^l \Rightarrow \bigcup_i p_i^c$.
So $RL \Longrightarrow RC$.

Now consider a learning algorithm that attempts to learn only the rules about conditions that change in the environment from one state to the next. This learning algorithm would not try to learn the rules

$$X \rightarrow b \rightarrow X$$

and

$$\overline{Y} \rightarrow a \rightarrow \overline{Y}$$

for the environment in Figure 3-4. The goal of the learner, in this case, is to learn a model that is equivalent to the complete-model excluding the rules that do not describe a changed postcondition. We define this model to be the Δ-complete-model.

Definition 6 *A* **Δ-complete-model** *of $\xrightarrow{A} C$ in a deterministic environment contains all the rules with action A and postcondition C that are true in the environment and that explain situations where C is changed from the previous state.*

The convergence theorem holds with minor modifications to the proof.

Theorem 2 *In a deterministic environment with manifest causal structure when every rule in the Δ-complete-model of $\xrightarrow{A} C$ is exercised infinitely often and every rule not in the Δ-complete-model of $\xrightarrow{A} C$ is violated infinitely often, the learning algorithm with the* **Deterministic-Rule-Reinforce** *procedure will converge to a model of $\xrightarrow{A} C$ that is predictively-equivalent to the Δ-complete-model of $\xrightarrow{A} C$.*

3.4.2 Convergence in Probabilistic Environments

This section extends the proof of Theorem 1 to include probabilistic environments. Recall from Chapter 1 that if the underlying environment is non-deterministic, if there is hidden state in the environment, or if the learner's perceptions of the environment are incomplete, then the agent's perceived environment is probabilistic. This section extends each step of the previous section to probabilistic environments. We define probabilistic environments with manifest causal structure and prove that the learning algorithm converges to the desired model. Probabilistic environments present several complications that must be resolved prior to attempting the convergence proof.

Randomized Action Selection

The main complication in probabilistic environments is that the probabilities of changing state in the environment are not always well-defined and may depend on the learner's action sequence. Thus, before we can define manifest causal structure in probabilistic environments we must make these probabilities well-defined by making an assumption about the learner's action selection mechanism.

Assumption 1 *The learner uses action selection such that in any perceptual state there is a probability vector on actions. I.e., for each perceptual state, PS there is a vector of action probabilities $(q_1^{PS}, q_2^{PS}, \ldots, q_n^{PS})$ such that the probability of taking action j in perceptual state PS is q_j^{PS}, where n is the number of actions and $q_j^{PS} > 0$ for each j. Call this action selection policy* **randomized action selection**.

Assuming that the agent uses a randomized action selection policy does not imply that the agent selects actions at random. Rather the agent can use any method to select actions, as long as the distribution of actions in each state defines a probability vector with the above requirements.

Definition of Environments with Manifest Causal Structure

This section defines the notion of manifest causal structure in probabilistic environments.

Definition 7 *An* **environment with manifest-causal-structure(Θ)** *is a connected graph where the nodes are states (subsets of perceptions) and there is at least one directed arc for each action from every state. The arcs are labeled with a probability. The sum of the probabilities for each action from any state is 1, and there is an arc from each state for each action with probability $\geq \Theta$. (We assume that the agent in the environment uses a randomized action selection policy.)*

We need to show that the probabilities on the arcs in the environment are well-defined.

Lemma 1 *The probabilities on arcs in an environment graph are well-defined when the agent uses a randomized action selection policy.*

Proof: Let the underlying environment have states $S_1, \ldots, S_k$, actions $A_1, \ldots, A_n$, and a probability on transitions $P(S_l \xrightarrow{A} S_m)$ for every pair of states S_l and S_m, and action A. (Recall that the underlying environment may be non-deterministic. If the underlying environment is deterministic then $P(S_l \xrightarrow{A} S_m) = 1$ for every transition.) The (perceived) environment has perceptual states $PS_1, \ldots, PS_{k'}$, actions $A_1, \ldots, A_n$, and a probability vector $(q^{PS}_{A_1}, q^{PS}_{A_2}, \ldots, q^{PS}_{A_n})$ for each perceptual state PS ($q^{PS}_{A_j}$ is the probability of taking action A_j in perceptual state PS). Lastly, there is a mapping H from states in the underlying environment to perceptual states where $H(S_l) = PS_i$ if state S_l maps to perceptual state PS_i.

For each state in the underlying environment, the probability of taking action A in state S is $P(S \xrightarrow{A}) = q^{H(S)}_A$. These probabilities define a Markov chain in the underlying environment.

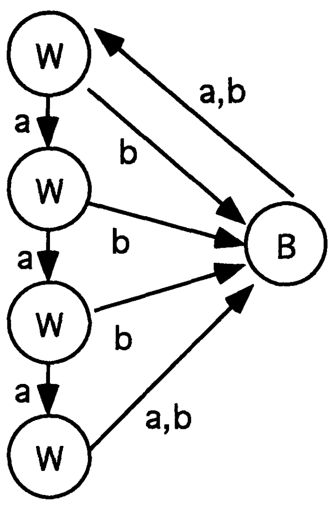
Since the underlying environment is a Markov chain, we can compute the probability of being in any state in the underlying environment. For each state S_m

$$P(S_m) = \sum_{S_l, A} P(S_l) P(S_l \xrightarrow{A}) P(S_l \xrightarrow{A} S_m)$$

where exists transition $S_l \xrightarrow{A} S_m$. To find the probabilities solve the linear equations.

The probability of being in a state together with the probabilities of taking any action define the probability of any arc in the perceptual environment. To compute the probability of an arc we need the probability of perceiving PS_i when starting in a state where PS_j is perceived and taking action A with probability $P(PS_j \xrightarrow{A})$. We can compute this arc probability as follows:

$$P(PS_i | PS_j \xrightarrow{A}) = \frac{P(PS_j \xrightarrow{A} PS_i)}{P(PS_j \xrightarrow{A})}$$



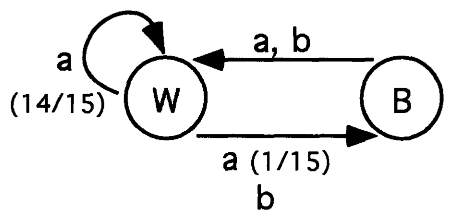


Figure 3-5: An example of a deterministic underlying environment and the corresponding non-deterministic perceived environment

$$= \frac{\sum_{S_l,S_m} P(S_l)P(S_l \xrightarrow{A})P(S_l \xrightarrow{A} S_m)}{\sum_{S_l} P(S_l)P(S_l \xrightarrow{A})}$$

where $H(S_l) = PS_j$, $H(S_m) = PS_i$, and exists transition $S_l \xrightarrow{A} S_m$. Thus the arc probabilities are well-defined.

To clarify the proof above let us compute the arc probabilities for a simple example. Consider the deterministic environment on the left-hand side of Figure 3-5. This environment collapses to the probabilistic perceived environment on the right, since the agent perceives only the condition W in all of the states on the left. Suppose that the probability of taking action a or b is 0.5 in every state.

We can compute the probability of being in any state as follows. Let x be the probability of being in state B of the underlying environment. Then there is x probability of being in the first W state, $x/2$ probability of being in the second W state, $x/4$ in the third, and $x/8$ in the last W state. Since the probability of being in some state is 1, we find that $x = 8/23$. Now to compute the transition probabilities in the perceived environment. The probability of going from B to W on action a or b is 1 as it is in the underlying environment. The probability of going from W to B on b is also 1 since all W states in the underlying environment go to B on b. The

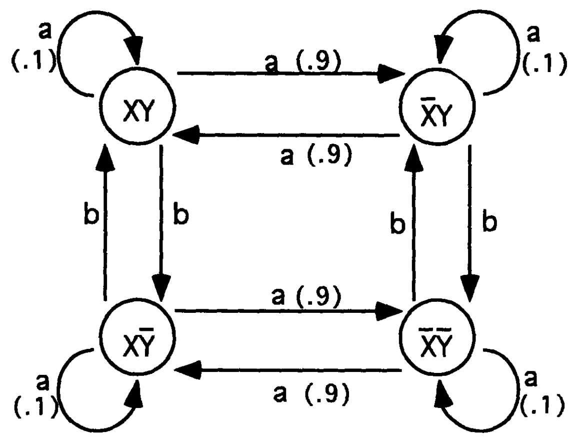


Figure 3-6: A probabilistic environment

probability of going from W to B on a is $P(B|W\xrightarrow{a}) = \dfrac{P(W \xrightarrow{a} B)}{P(W \xrightarrow{a})}$. The probability of transition from a W state on a to a B state is $P(W\xrightarrow{a}B) = (x/8) \cdot (1/2)$, and $P(W\xrightarrow{a}) = 15x/16$. So $P(B|W\xrightarrow{a}) = 1/15$ and the probability of going from W to W on a is $14/15$.

Note that when the perceptual environment has hidden state, the trials may not be independent, which violates the conditions under which the sequential ratio test is known to achieve its acceptance requirements. But because the action selection is randomized and rules are tested repeatedly with longer and longer tests, the tests are likely to include examples from unrelated states of the environment. Thus we assume the trials are sufficiently independent.

The perceived environment in Figure 3-6 is a probabilistic environment with manifest causal structure. This environment is similar to the environment of Figure 3-4, but b toggles Y deterministically while a toggles X with high probability (0.9). As a result this environment has manifest-causal-structure(.9) with probability vector $(.5, .5)$ on actions a and b in every state.

In an environment with manifest-causal-structure(Θ) there may be rules that are not perfect predictors. For example the rule

$$X \rightarrow a \rightarrow \overline{X}$$

has reliability .9 in the environment of Figure 3-6. Therefore, we define a complete-model(Θ). The goal of the learner is to learn a model that is predictively-equivalent to the complete-model(Θ) of the environment.

Definition 8 *The* **complete-model(Θ)** *of $\xrightarrow{A} C$ in an environment with a randomized action selection policy is the set of all rules with action A and postcondition C that have defined reliability $\geq \Theta$ in the environment.*

Proof of Convergence in Probabilistic Environments

This section presents the main (theoretical) result of this chapter in Theorem 3 which states that learning converges to the complete-model(Θ) in environments with manifest causal structure. Before proceeding with the main convergence theorem we need the following lemma which states that the learning algorithm for probabilistic environments makes a finite number of mistakes when it evaluates rules.

Lemma 2 *The number of erroneous acceptances and rejections the* **Probabilistic-Rule-Reinforce** *procedure makes makes with parameters $\alpha(t) = \beta(t) = \frac{1}{2^{2\lceil \log t \rceil}}$ is finite.*

Proof: The number of mistakes the **Probabilistic-Rule-Reinforce** procedure makes is bounded by the probability of making mistakes at each trial ($\alpha(t) + \beta(t)$) multiplied by the number of rules at each trial (which is bounded by a constant). So it remains to show that the probabilities of making mistakes for all time have a finite sum.

Let $k = \lceil \log t \rceil$.

$$\sum_{t=1}^{\infty} \alpha(t) = \sum_{t=1}^{\infty} \frac{1}{2^{2\lceil \log t \rceil}} = \sum_{k=0}^{\infty} \frac{1}{2^{2k}} 2^k = \sum_{k=0}^{\infty} \frac{1}{2^k} = 2.$$

Similarly $\sum_{t=1}^{\infty} \beta(t)$ is finite. So the total number of mistakes the algorithm makes is finite with probability 1 (see the Borel-Cantelli lemmas (Grimmett & Stirzaker 1982)).

The convergence result for probabilistic environments shows that the learning algorithm converges to a model that is predictively equivalent to the complete-model(Θ). It assumes that the learner uses a randomized action selection policy. Like Theorem 1, rules in the complete-model(Θ) must be exercised infinitely often and rules not in the complete-model(Θ) must be violated infinitely often. If the underlying environment is finite then randomized action selection is sufficient to guarantee that rules in the complete model are exercised infinitely often and rules not in the complete model are violated infinitely often. In Theorem 3 we do not assume that the environment is finite. Rather the theorem assumes that rules are exercised infinitely often directly.

The proof of the theorem follows the outline of the proof for Theorem 1, with modification in the details.

Theorem 3 *In an environment with manifest causal structure where the learner uses a randomized action selection policy and every rule in the complete-model(Θ) of $\xrightarrow{A} C$ is exercised infinitely often and every rule not in the complete-model(Θ) of $\xrightarrow{A} C$ is violated infinitely often, the learning algorithm with the* **Probabilistic-Rule-Reinforce** *procedure will converge to a model of $\xrightarrow{A} C$ that is predictively-equivalent to the complete-model(Θ) of $\xrightarrow{A} C$.*

Proof: Consider the set of rules accepted by the sequential ratio test with threshold Θ. Call this rule set RL.

(1) RL converges.

Since the environment has manifest causal structure, there is a finite number of rules for $\xrightarrow{A} C$ with reliability $\geq \Theta$.

At any time that the set of rules the algorithm learned does not explain some situation there is a non-zero probability that the algorithm will create a rule with reliability $\geq \Theta$. Since the algorithm makes a finite number of mistakes, when it is not making any more mistakes it is guaranteed never to reject these rules. Eventually either all rules with reliability $\geq \Theta$ will be in RL, or all situations that can be explained by rules with reliability $\geq \Theta$ will be explained by some rule in RL.

New rules to explain situations that are not explained by any rule with reliability

$\geq \Theta$ will be created continually, but these rules have reliability $< \Theta$. When the **Probabilistic-Rule-Reinforce** algorithm does not accept erroneously these rules will not be accepted.

(2) The complete-model(Θ) of $\xrightarrow{A} C$, RC, and the learned model, RL, are predictively-equivalent.

Show that $RC \Longrightarrow RL$.

For any rule $r^c = p^c \xrightarrow{A} C$ in RC

case 1 $r^c \in RL$. Then clearly $p^c \Rightarrow \bigcup_i p_i^l$ (the union of all preconditions in RL), because it is equal to one of them.

case 2 $r^c \notin RL$. r^c could be created and then removed with probability $\alpha(t)$, but there are infinitely many opportunities to create r^c and the number of mistakes the algorithm makes is finite so the probability that the learner is continually creating and removing r^c is 0. Thus the learner is not creating r^c for one of two reasons:

- $\xrightarrow{A} C$ was explained by some other accepted rule every time r^c applied. In this case $p^c \Rightarrow \bigcup_i p_i^l$.
- The rule creation algorithm didn't select the preconditions p^c. But with infinitely many repetitions of $p^c \xrightarrow{A} C$, and random precondition selection, p^c would be selected eventually with probability 1.

So $RC \Longrightarrow RL$.

Now show that $RL \Longrightarrow RC$.

For any rule $r^l = p^l \xrightarrow{A} C$ in RL it must be the case that $r^l \in RC$.

Assume to the contrary that $r^l \notin RC$, then r^l has reliability $< \Theta$. r^l may be created repeatedly (in an attempt to explain a situation not explained by any rule with reliability $\geq \Theta$). But the probability of accepting a rule with reliability $< \Theta$ is 0 after the algorithm has made the (finite) number of mistakes it will make. So $RL \Longrightarrow RC$.

We can again define the model for probabilistic environments that only contains rules for changed postconditions.

Definition 9 *A* **Δ-complete-model(Θ)** *of $\xrightarrow{A} C$ in an environment with manifest-causal-structure(Θ) contains all the rules with action A and postcondition C that are true in the environment and that explain situations where C is changed from the previous state.*

The convergence theorem for probabilistic environments holds and proves that the rule-learning algorithm, which only learns about changed conditions, converges to a model that is predictively-equivalent to the Δ-complete-model(Θ) of the environment.

Theorem 4 *In an environment with manifest causal structure where the learner uses a randomized action selection policy and every rule in the Δ-complete-model(Θ) of $\xrightarrow{A} C$ is exercised infinitely often and every rule not in the Δ-complete-model(Θ) of $\xrightarrow{A} C$ is violated infinitely often, the learning algorithm with the* **Probabilistic-Rule-Reinforce** *procedure will converge to a model of $\xrightarrow{A} C$ that is predictively-equivalent to the Δ-complete-model(Θ) of $\xrightarrow{A} C$.*

To summarize, this section proves that the rule-learning algorithm from Figure 3-2 converges to a good predictive model of environments with manifest causal structure.

3.5 Learning Rules in the Macintosh Environment

This section shows that the rule-learning algorithm is powerful enough to learn the complex Macintosh Environment. The experiments reported in this section ran on a Quadra 610 Macintosh computer. It is of great interest to this research that the rule-learning algorithm succeeds in learning a complex environment on a relatively slow computer such as the Quadra. This computer is slow compared with typical computers used for world model learning research, such as a Connection Machine or a Sparc workstation. The learning phases are time and space intensive on the Macintosh, but learning the rules for one relation function typically requires a few hours (e.g., 150000 trial to learn the $EXIST$ relation take six hours of actual time — as opposed to CPU time). The resource use shows that the rule-learning algorithm is indeed efficient, although we cannot compare this algorithm directly with previous results with different environments.

Evaluation of the empirical success of the rule-learning algorithm includes

- examining the learned world model,
- using the world model to predict, and
- using the world model to achieve a goal.

The following sections present empirical results for each method of evaluation.

3.5.1 The Learned World Model

Due to the complexity of the Macintosh Environment and the low level of the perceptions there are thousands of valid rules. Later chapters develop concept learning which should reduce the number of rules in the world model. In this section only rules that are comprised of direct perceptions are considered.

Since the Macintosh Environment is not deterministic the rule-learning algorithm never stops creating new rules. We know from Theorem 4 that the set of accepted rules converges so eventually no new rules are accepted. The rule-learning algorithm is, however, continually creating new rules to explain those aspects of the environment that are not manifest. Therefore, the total number of rules remains large after the world model predicts effectively. Many of these rules are on probation as we see in the prediction trace of Figure 3-9.

The learning algorithm can use a maximum of 3000 rules for each relation it learns. To explain the $EXIST$ relation the learner requires about 500 rules of which about 350 are off probation (see the prediction trace in Figure 3-9). The $TYPE$ relation is explained primarily with NOACTION rules and the learner uses fewer than 100 rules to explain this relation. The OV, X, and Y relations are binary and thus have many more perceptions to explain. As a result the number of rules needed to explain these relations is much higher than the number of rules needed to explain either the *EXIST* or the *TYPE* relations. Experiments show that the learning algorithm uses nearly all 3000 possible rules to learn these relations. Of the 3000 rules close to 2000 are valid.

The number of rules is too large to list the entire model. Rather, Figure 3-7 lists a number of interesting learned rules. Section 3.5.2 evaluates the model as a whole through prediction. Each rule in Figure 3-7 is presented together with the number of times it predicted successfully, its status (i.e., on or off probation), and its estimated reliability. Each rule demonstrate some correlations the agent learned. For example, the first rule indicates that whenever *Window 2* is visible so is *Window 2*'s grow-box and the fourth rule states that a click in *Window 1* makes *Window 1*'s active-title-bar present.

3.5.2 Predicting with the Learned World Model

The predictive nature of the rules permits the agent to predict the next state of the world. Each rule that applies (i.e., its preconditions are true in the current state and its action is the current action) predicts that its postcondition will be true in the next state. The prediction has a **prediction-strength** which is equal to the reliability of the rule that makes the prediction. If more than one rule predicts a condition then the largest rule reliability becomes the prediction-strength of this condition. The agent also assumes that any relation that is not given a value by some rule retains its current value. This prediction algorithm is shown in Figure 3-8.

For each relation the prediction algorithm can predict

- the correct value,
- the correct value and other values,
- some number of incorrect values, or
- no predicted value.

1. (success 10, probation NIL, reliability 1.0)
   EXIST (Window 2) = T → NOACTION → EXIST (Window 2 GB) = T
2. (success 22, probation NIL, reliability 1.0)
   EXIST (Window 1 ATB) = T
   → NOACTION → EXIST (Window 1 INTERIOR) = T
3. (success 20, probation NIL, reliability 1.0)
   NIL → click-in Window 1 CB → EXIST (Window 1) = NP
4. (success 39, probation NIL, reliability 1.0)
   X (Window 2, Window 1) = 2121
   → click-in Window 1 → EXIST (Window 2 TB) = T
5. (success 13, probation NIL, reliability 1.0)
   X (Window 2 ATB, Window 1 TB) = 1212
   → click-in Window 1 BUTTON-DIALOG-ITEM Window 2
   → EXIST (Window 2 TB) = T
6. (success 1, probation NIL, reliability 1.0)
   EXIST (Window 1) = T
   → click-in Window 2 ZB → EXIST (Window 1 INTERIOR) = NP
7. (success 5, probation NIL, reliability 1.0)
   EXIST (Window 1 ATB) = T → NOACTION → TYPE (Window 1 ATB) = ATB
8. (success 4, probation NIL, reliability 1.0)
   EXIST (Window 1 ATB) = T
   → NOACTION → OV (Window 1 CB, Window 1 ATB) = T
9. (success 63, probation NIL, reliability 0.955)
   OV (Window 1, Window 2) = T
   → click-in Window 2 INTERIOR → OV (Window 1 Window 2) = F
10. (success 5, probation NIL, reliability 1.0)
    Y (Window 1, Window 2 GB) = 1122
    → click-in Window 1 TB → OV (Window 1, Window 2) = T
11. (success 36, probation NIL, reliability 1.0)
    NIL → click-in Window 1 TB
    → Y (Window 1 CB, Window 1 INTERIOR) = 1122
12. (success 7, probation NIL, reliability 1.0)
    X (Window 2 ATB, Window 1) = 1212
    → click-in Window 1 TB → X (Window 1 ATB, Window 2) = 2121

Figure 3-7: Rules learned in the Macintosh Environment. Notice that all the rules are valid, but not all of them are the rules we expect or want to find. For example, rule 6 would be correct with no preconditions.

Algorithm 5 Predict*()*

```
Let predict-perceptions = $\emptyset$.
For each rule r
    if applies(r, current-perceptions, current-action) then
        if postcondition(r) is in predict-perceptions
            then old-strength = strength of prediction of postcondition(r)
            else old-strength = 0.
        add postcondition(r) to predict-perceptions with strength
        MAX(reliability(r),old-strength).
Repeat until no new perceptions are added
    For each NOACTION rule r
        if applies(r, predicted-perceptions, NOACTION) then
            if postcondition(r) is in predict-perceptions
                then old-strength = strength of prediction of postcondition(r)
                else old-strength = 0.
            add postcondition(r) to predict-perceptions with strength
            MAX(reliability(r),old-strength).
For each relation rel in the current-perceptions
    if there is no value for rel in predict-perceptions
        add current value of rel to predict-perceptions with strength 1.
```

Figure 3-8: The **Predict** Algorithm

```
trial 6240 rule count = 469 (on probation 134) mysteries 0
     Prediction: found relations 8, mistakes  0.0, missed 0, Total 8 :
     Smoothed Error 0.81

trial 6241 rule count = 469 (on probation 134) mysteries 0
     Prediction: found relations 12, mistakes  0.0, missed 0, Total 12 :
     Smoothed Error 0.81

trial 6242 rule count = 469 (on probation 134) mysteries 0
     Prediction: found relations 7, mistakes  0.0, missed 0, Total 7 :
     Smoothed Error 0.81

trial 6243 rule count = 469 (on probation 134) mysteries 0
     Prediction: found relations 7, mistakes  0.0, missed 0, Total 7 :
     Smoothed Error 0.74
```

Figure 3-9: A trace of a few trials in the Macintosh Environment.

To evaluate the prediction at every trial the algorithm compares the predicted perceptions with the new perceptions. When the algorithm predicts the correct value for a relation, we say the value is *found.* If any incorrect values are predicted the corresponding prediction strengths are added (over all incorrect values). These are prediction *mistakes.* If a relation in the new perceptions has no predicted value, we say it was *missed.* The total prediction error for each trial is the number of missed perceptions plus the sum of the strengths of the prediction mistakes.

The total prediction error of one trial is not a good measure of the world model's predictive ability. The model can predict perfectly for twenty trials and be surprised by a rare event that it cannot predict on the twenty-first trial. For example, if *Window 1* completely hides *Window 2* and the agent clicks parts of *Window 1* or the *background* for 20 trials the agent may predict perfectly. Suppose that on the twenty-first trial the agent clicks the close-box of *Window 1.* The agent cannot predict the appearance of *Window 2* so the prediction error in this trial is high. Therefore, to give a quantitative value to the world model's predictive ability we look at the average error of a window of 100 prediction trials. Call this the smoothed error.

Figure 3-9 shows a trace of a small number of trials. The agent is learning rules

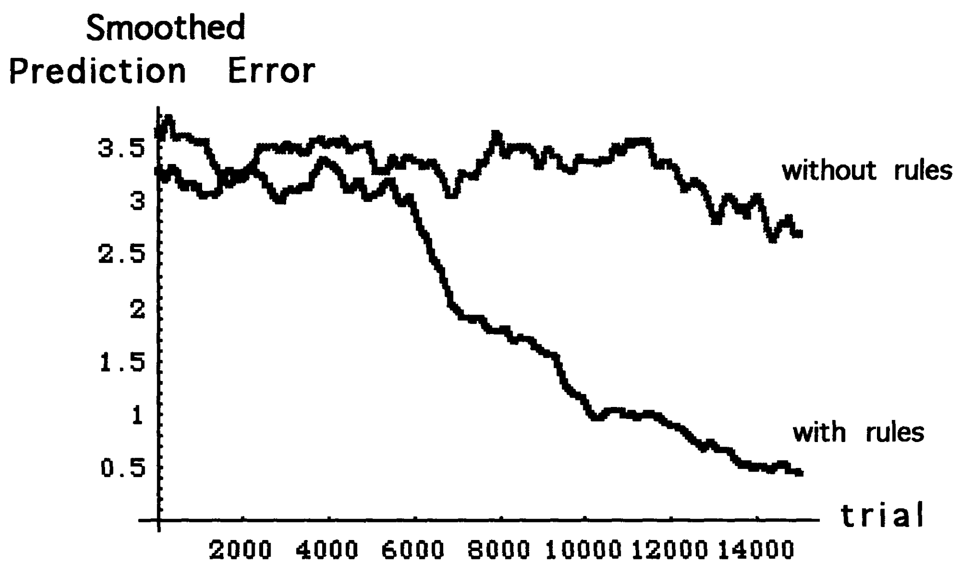


Figure 3-10: A graph of the smoothed error values as the agent learns the *EXIST* relation (the black line) compared with the smoothed error for an empty model (the gray line). To make this graph the prediction errors that the agent makes are further smoothed by averaging a window of 1000 trials. The agent is learning NOACTION rules only in the first 6000 trials.

to explain the *EXIST* relation only. It therefore predicts relations for the EXIST relation only. Recall that to save space the agent attempts to explain one relation function at a time. The trace in Figure 3-9 is late in the learning phase for the *EXIST* relation. For each trial the total number of rules, the number of rules on probation, and the number of mysteries are shown as well as the prediction values. The agent has a large number of valid rules to explain the *EXIST* relation (specifically $470 - 135 = 335$ valid rules). As you can see the agent makes few prediction mistakes and the smoothed prediction error is low (near .8 averaged errors per trial compared with near 3.5 averaged errors per trial with no learned rules).

Figure 3-10 shows a graph of the smoothed error values as the agent learns about the *EXIST* relation. The figure compares the error while learning with the smoothed error when the agent has no rules (i.e., it always predicts no change). After an initial learning phase the learner predicts better and continues to improve. It finally reaches an error rate so low that it can be attributed to non-determinism of the environment. Since the agent only learns NOACTION rules in the first 6000 trials there is an obvious

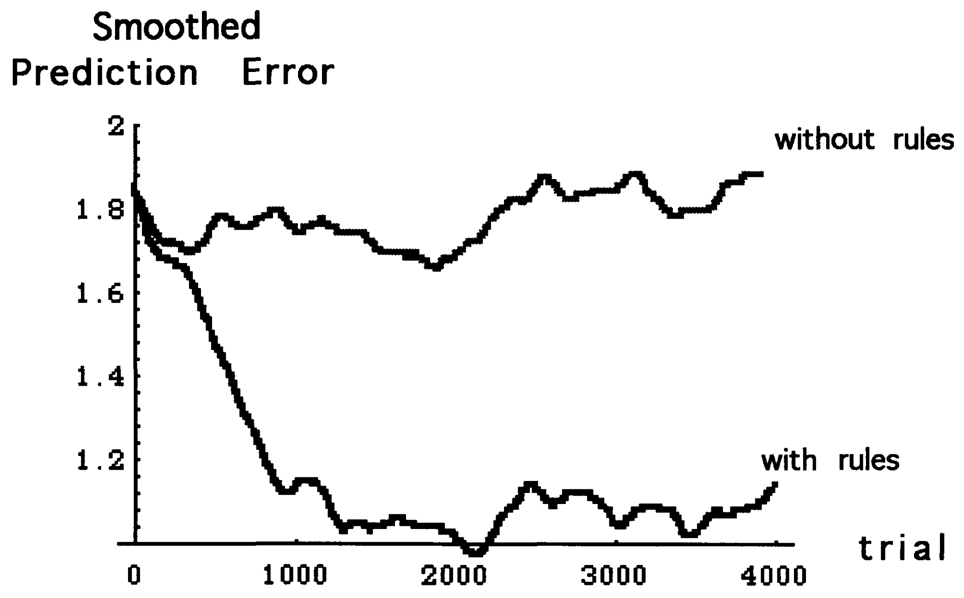


Figure 3-11: A graph of the smoothed error values as the agent learns the *TYPE* relation (the black line) compared with the smoothed error for an empty model (the gray line). To make this graph the prediction errors that the agent makes are further smoothed by averaging a window of 1000 trials. The agent is learning NOACTION rules only.

change in the graph at that time. Until trial 6000 there is no apparent improvement in prediction because NOACTION rules alone cannot be used to predict. Later prediction exhibits a more typical learning curve.

Similar prediction results are shown for the *TYPE* and *OV* relations in Figures 3-11 and 3-12 respectively. The agent uses the model it has already learned for the *EXIST* relation when learning both the *TYPE* and the *OV* relations. The agent learns the *TYPE* relation very quickly since once the *EXIST* relation is explained NOACTION rules are sufficient to explain the *TYPE* relation. The *OV* relation is more difficult to learn than either the *TYPE* or the *EXIST* relations because it has two arguments and thus there are many perceptions of the *OV* relations and, as we can see in Figure 3-12, a high prediction error rate before learning. The agent learns very quickly at first (using NOACTION rules). Further progress is slower. Traces of prediction for the *X* and *Y* relations are similar to the prediction graph for learning the *OV* relation. The prediction error when learning the *X* and *Y* relations is show in Figures 3-13 and 3-14 respectively.

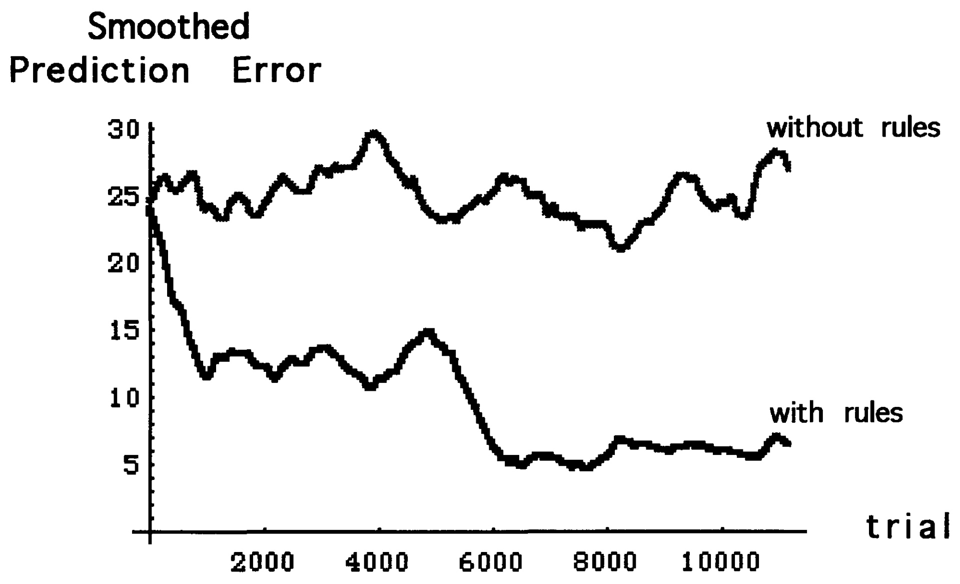


Figure 3-12: A graph of the smoothed error values as the agent learns the OV relation (the black line) compared with the smoothed error for an empty model (the gray line). To make this graph the prediction errors that the agent makes are further smoothed by averaging a window of 1000 trials. The agent is learning NOACTION rules only in the first 5000 trials.

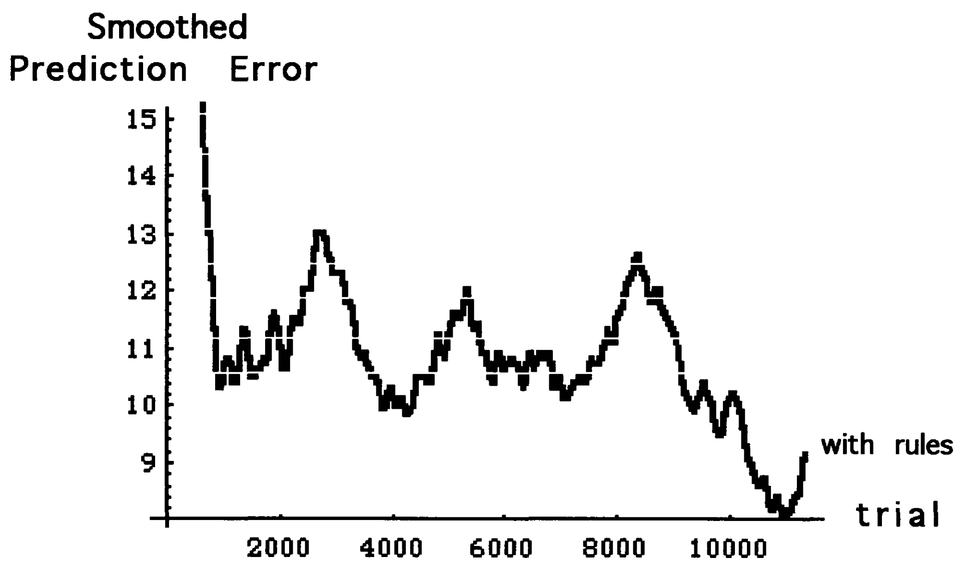


Figure 3-13: A graph of the smoothed error values as the agent learns the X relation. To make this graph the prediction errors that the agent makes are further smoothed by averaging a window of 1000 trials. The agent is learning NOACTION rules only in the first 5000 trials.

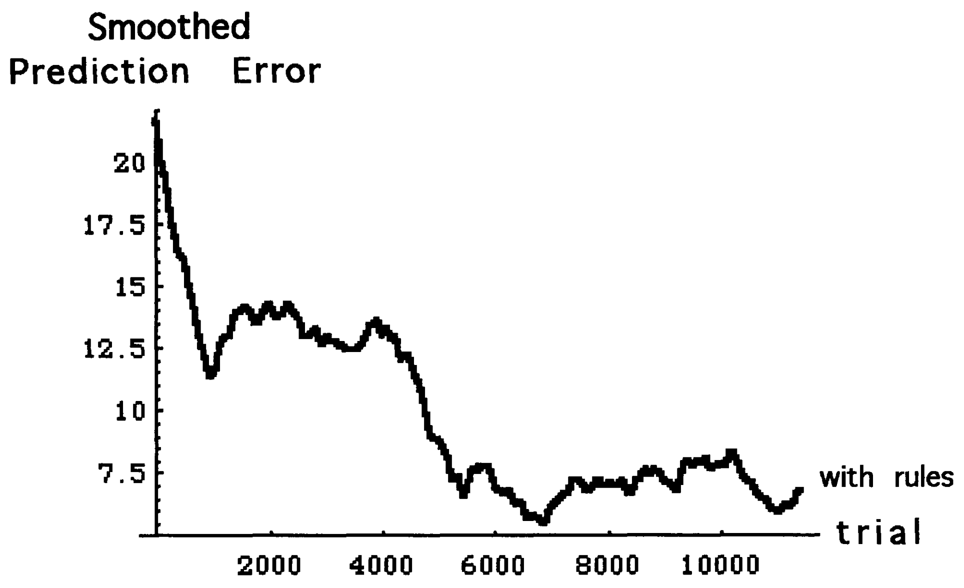


Figure 3-14: A graph of the smoothed error values as the agent learns the Y relation. To make this graph the prediction errors that the agent makes are further smoothed by averaging a window of 1000 trials. The agent is learning NOACTION rules only in the first 5000 trials.

3.5.3 Achieving a Goal

In this section we describe how the agent uses the learned world model to achieve a goal. The goal-oriented action selection implemented for this thesis is a simple procedure which does not take advantage of all the known techniques for goal-oriented action selection. We use it merely to show that the learned model can be used to achieve goals. This action-selection algorithm performs standard backward chaining using the rules in the model. Starting from the goal it finds a rule that has the goal as postcondition. It makes the preconditions of the rule sub-goals and recursively tries to achieve these sub-goals until its goal is true in the current environment. Then the agent takes the resulting series of actions and re-plans if necessary (e.g., when an incorrect rule is used and the resulting situation is not the expected situation).

When the agent — beginning in the screen situation of Figure 3-15 — tries to achieve the goal $OV(\text{Window 1}, \text{Window 2}) = T$ it performs the action leading to the screen situations in Figure 3-16 (i.e., a click in the button for *Window 2*). Then it clicks in *Window 1*'s interior and reaches the desired goal situation in Figure 3-17. The agent used the following rules to achieve its goal:

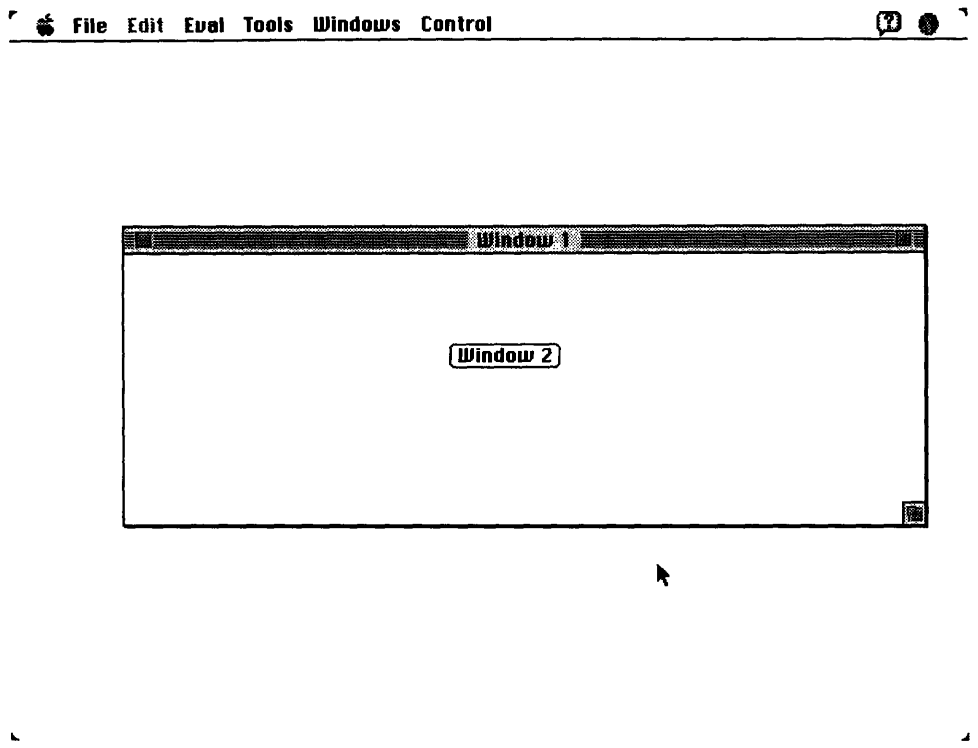


Figure 3-15: Starting situation for an agent with goal $OV(\textit{Window 1}, \textit{Window 2}) = T$

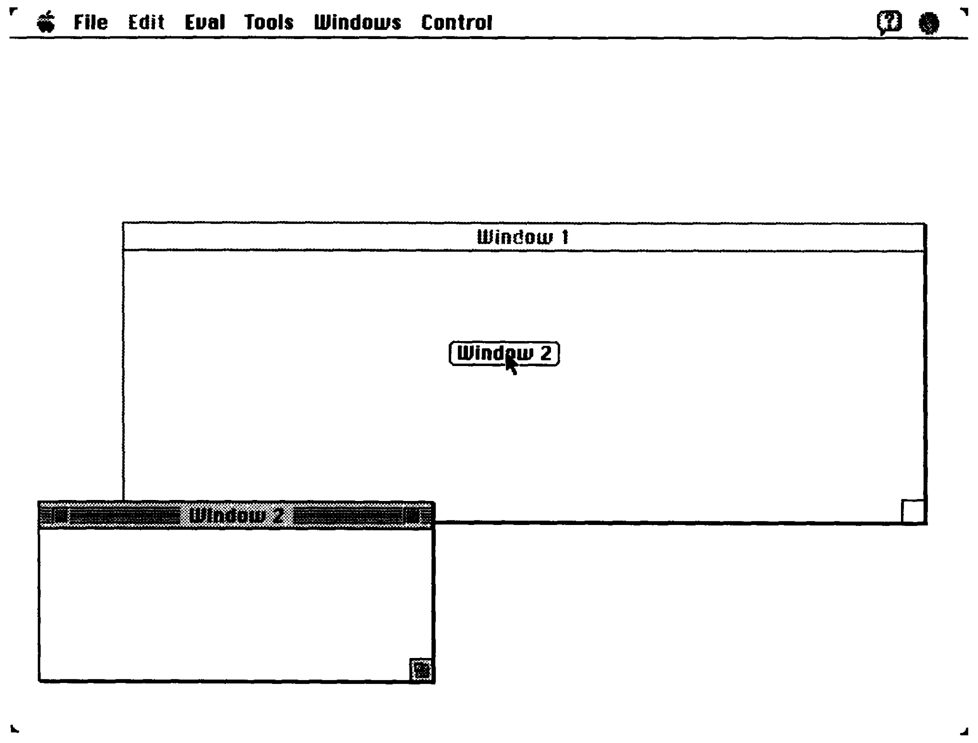


Figure 3-16: Intermediate situation agent with goal $OV(\textit{Window 1}, \textit{Window 2}) = T$

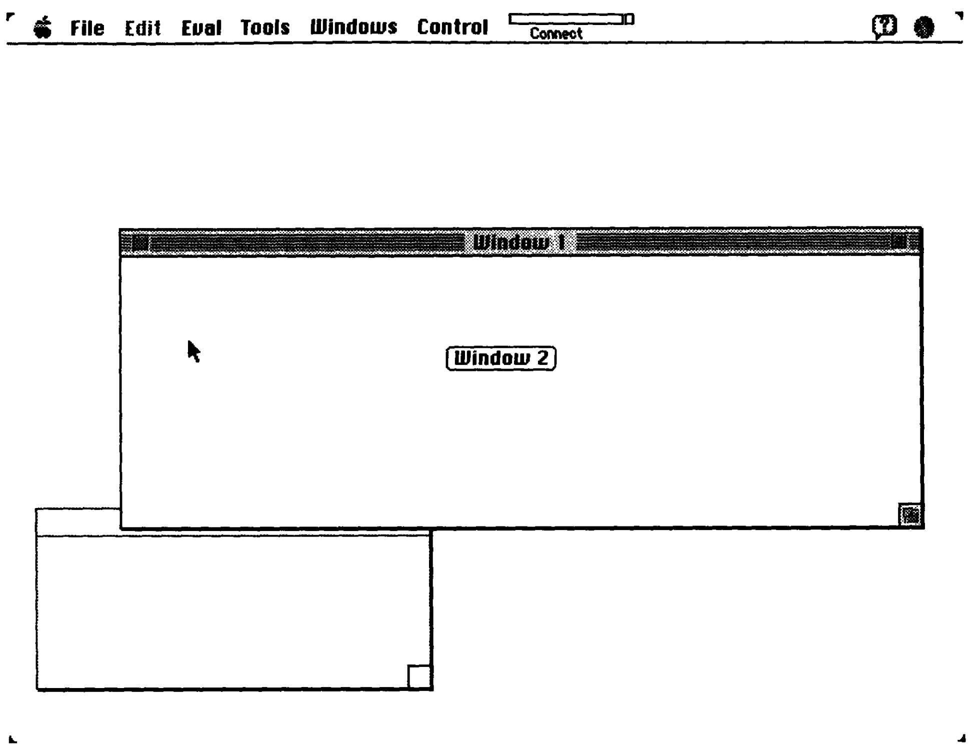


Figure 3-17: Final situation for an agent with goal *OV*(*Window 1*, *Window 2*) = *T* — goal achieved!

```
(success 35, probation NIL, reliability 0.972)
    EXIST (Window 1 ATB) = T -->
    click-in Window 1 BUTTON-DIALOG-ITEM Window 2 -->
    OV (Window 1 INTERIOR, Window 2) = F


(success 3, probation NIL, reliability  1.0)
    OV (Window 1 INTERIOR, Window 2) = F -->
    click-in Window 1 INTERIOR --> OV (Window 1, Window 2) = T
```

3.6 Discussion

The previous section showed that the rule-learning algorithm learns a world model that captures knowledge of the environment well enough to predict and plan. Several questions of interest arise about the learning algorithm: how does the algorithm cope with a new environment or a changing environment, how much time does the learning algorithm spend creating invalid rules, and how do mysteries affect the speed of learning? These questions are discussed in the following sections.

3.6.1 Learning in New or Changing Environments

The experiments in Section 3.5 showed that the rule-learning algorithm learns a good world model of an environment with *Window 1* and *Window 2* since the experiments were all conducted in environments with exactly those two windows. Suppose now that an agent has a world model which it learned in such an environment (with *Window 1* and *Window 2*). When the agent encounters a somewhat different environment containing three windows (*Window 1*, *Window 2*, and *Window 3*), how useful will its world knowledge be and how will the learning algorithm react?

Obviously a world model that was learned with a two window environment is incomplete and sometimes incorrect in a three window environment. For example, in a two window environment, a rule stating that if one window is active and the other is present then closing the active window will make the second window active, is valid. In a three window environment this rule is not valid since either the second or the third window can become active. Thus, in changed environments, the world model contains some rules that are still valid, but contains some rules that are no longer valid and is missing some valid rules.

The rule-learning algorithm can cope with such a change well because this algorithm continues learning indefinitely. The rules that remain valid will not be removed and the algorithm will continue to use them for prediction. Thus the world model starts with more knowledge in a changing environment than in a completely new environment. The rules in the model that are no longer valid will fail frequently in the changed environment and the algorithm will remove them. Most importantly, since the learning algorithm will once again encounter unexplained events, it will create new rules to explain these events. The world model will adjust after a learning phase to an accurate model of the changed environment. This learning algorithm, therefore, is adaptive.

A somewhat different question is how the agent reacts to a new environment, e.g., an environment containing two windows — *Window 4* and *Window 5* — that are unfamiliar to the agent. In this case the specific rules learned by the rule-learning algorithm in this section are not useful. These rules state facts about the specific windows, *Window 1* and *Window 2*, that it encountered during learning. This knowledge cannot be transferred to other windows, so the rule-learning will have to learn about the new environment with no prior knowledge. Chapter 5 presents a rule-generalization algorithm that address the problem of rule specialization.

3.6.2 Time Spent Creating Rules

We know that the rule-creation algorithm merely guesses rules and rule-evaluation determines if these rules are valid. Therefore many of the rules created are bad. We are interested in determining how much time the algorithm spends creating bad rules early and late in the learning process.

We do not know if a rule is good or bad when it is created. Therefore, we must use the number of rules removed for evidence of how many bad rules are created. Consider the following experiment. The rule-learning algorithm is learning in a two window environment. It is focusing on the *EXIST* relation only. It learns for 16000 trials of which the first 5000 are spend learning NOACTION rules.

We count the number of rules created and removed in an early interval (trials 5000 - 6000) and a late interval (trials 15000 - 16000). In the early interval 512 rules are created and 276 rules are removed. In the late interval 177 rules are created and 159 rules are removed. We can see the improvement to the world model from the reduced number of rules created. Furthermore, the absolute number of bad rules created is smaller later from this evidence (159 rules removed later compared with 276 rules removed early). On the other hand the probability that a rule is bad is higher later in learning ($159/177 \approx .9$ compared with $276/512 \approx .54$) which is reasonable since a higher percentage of the surprising situations cannot be explained when the world model is fairly good.

This experiment shows that, as we expect, as the world model improves it has fewer situations to explain and therefore it creates fewer rules. Likewise, it creates

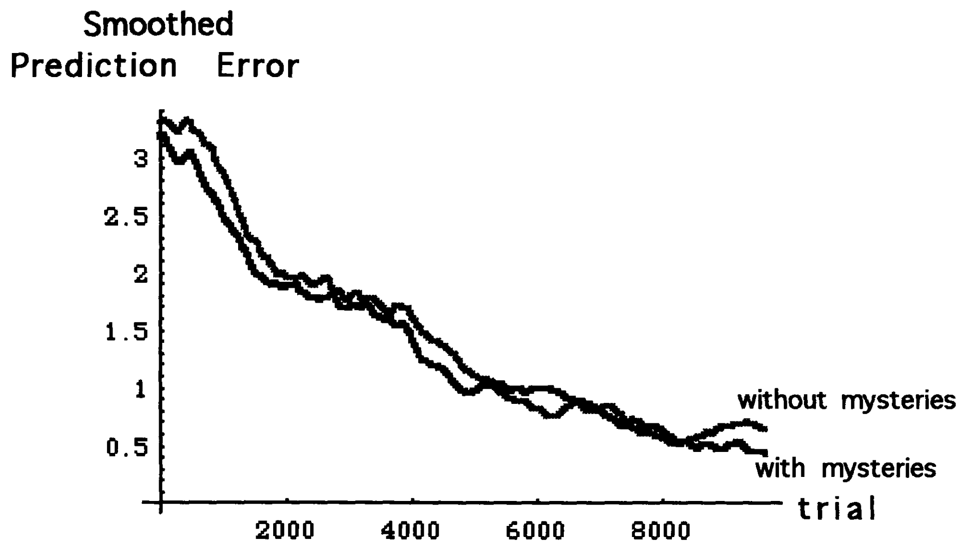


Figure 3-18: A graph of the smoothed error values as the agent learns the *EXIST* relation with mysteries (the black line) compared with the smoothed error when the error learns without mysteries (the gray line). To make this graph the prediction errors that the agent makes are further smoothed by averaging a window of 1000 trials.

fewer bad rules with time, although a higher percentage of the rules created are bad.

3.6.3 Learning with Mysteries

We discussed the use of mysteries to learn about rare events in Section 3.3.2. Mysteries improve learning because re-playing rare events increases the probability of creating rules that explain the mysteries. Thus the main difference between learning with and without mysteries is that a few specific rules, which explain the mysteries are created faster when mysteries are used.

This improvement is easy to measure if the complete set of rules making up the world model is known and can be encoded. In this case the learned model can be compared with the "perfect" model and the number of correct rules can be measured directly. If we can count the number of correct rules we can compare the percentage of good rules learned with and without mysteries.

Experiments in a grid environment, which we discussed briefly in Chapter 2, showed that using mysteries the learning algorithm consistently finds more of the correct rules than it finds without mysteries. Unfortunately, in the Macintosh Envi-

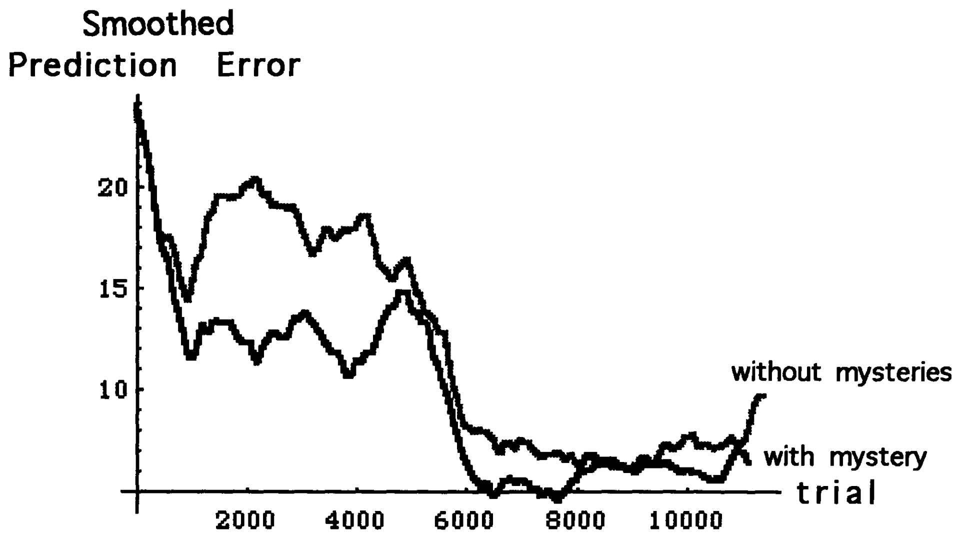


Figure 3-19: A graph of the smoothed error values as the agent learns the *OV* relation with mysteries (the black line) compared with the smoothed error as the agent learns without mysteries (the gray line). To make this graph the prediction errors that the agent makes are further smoothed by averaging a window of 1000 trials. The agent is learning NOACTION rules only in the first 5000 trials.

ronment, listing the perfect model is not realistic or even possible. Furthermore, the Macintosh Environment, as we discussed in Section 3.3.2, does not have rare events. Therefore the benefit of using mysteries is not as clearly evident in this environment. To support the claim that mysteries speed up learning we can examine graphs that compare prediction with and without mysteries. Figures 3-18 and 3-19 compare prediction of the *EXIST* and *OV* relations with and without mysteries. Both these graphs demonstrate a small speedup early in the learning process. The apparent prediction improvement in any one of these graphs is small and not convincing on its own. Since the graphs are consistent this evidence indicates that mysteries improve learning even in the Macintosh Environment.

3.7 Related Approaches to Rule Learning

This section describes a select number of related works that are directly related to learning rule based causal world models. The approaches to learning as well as the complexity of the learned environments are compared.

Early work in theoretical machine learning showed that finite automata are learnable from queries and counter-examples (Angluin 1987). In other words there is an algorithm that learns a world model in any deterministic finite automaton environment. The structure of this model is not a set of rules, but a finite automaton is easily translated to a set of rules. For every transition from state s_1 to state s_2 on action a make the rule $s_1 \rightarrow a \rightarrow s_2$. The limitations of this research compared with the issues in this thesis are the severe restriction of environment types and the need for counterexamples which are not available to the autonomous agent in this research.

Dean et al. (1992) address autonomous learning of deterministic finite automata environments with noise. Like Angluin (1987), the algorithms are not as general as the rule-learning algorithm in this chapter because the learning algorithm assumes that the underlying environment is a deterministic finite automaton.

Rivest & Schapire (1990) explore autonomous learners in a finite automata environment with hidden state. In other words the underlying environment is deterministic, but the agent has only partial perceptions of the state. Rivest & Schapire (1990) give an algorithm that learns the finite automaton (modulo states that cannot be distinguished) from experimentation. This research address the specific problem of hidden information — a problem that this thesis avoids in favor of exploring more complex environments.

Drescher (1989) develops the schema mechanism which is most closely related to the work in this thesis. The structure of the rules in the world model is based on Drescher's schemas but simplified somewhat. The main difference between the schema mechanism and this thesis is that Drescher's work focuses on learning hidden state, whereas the work in this thesis concentrates its effort on learning the part of the environment that is easy to learn.

There are illuminating differences between the schema generating algorithm and the rule-learning algorithm in this thesis. The primary difference is that schemas are never removed. Once a schema is generated it can improve upon itself by creating (spinning-off) new schemas, but it is not removed even if it has proved to be invalid. This strategy relies upon clever schema generation procedures which collect a great deal of statistics about the relevancy of potential preconditions and postconditions. It also requires a great deal of memory and computational resources. The rule-learning algorithm in this thesis, on the other hand, guesses potential rules on demand, when unexplained situations occur. The "thinking" goes into choosing which rules to remove.

Shen (1993) presents a different approach to learning rule-based world models of deterministic environments. This work concentrates on finding general explanations for perceived effects. We might say that the learning algorithm generalizes first and asks questions later. Rules are typically over-generalized when they are created and the algorithm makes them more specific with experience. The rule-learning algorithm of this thesis, on the contrary, learns as much specific information as possible first. It generalizes when it has enough specific knowledge to make a good guess at the general concept, which mimics what people typically do.

3.8 Summary

This chapter presented a rule-learning algorithm that uses simple rule-creation strategies coupled with reliable statistical methods of separating good and bad rules. The rule-learning algorithm is proven to converges to a good predictive model in environments with manifest causal structure. An agent uses this algorithm to learn the Macintosh Environment and the empirical results of these experiments show effective learning of a realistic and complex environment.

Chapter 4

Correlated Perceptions as New Concepts

The previous chapter described an algorithm that excels at finding correlations in the environment. For any environment conditions and action the rule-learning algorithm makes rules with every consistent resulting condition in the environment as a postcondition. In many environments there is some perceptual redundancy such as co-occurring perceptions or a perception that is always true when another perception is true. The algorithm in this chapter learns new concepts by finding redundant or correlated perceptions. It changes the world model to use the newly learned concepts resulting in a world model with fewer rules that are closer both to the "natural" causes of effects in the environment (rather than perceptions that are correlated with the effect) and to the way people think about the environment.

Consider the event of bringing *Window 1* to the foreground in Figure 4-1. In the previous state *Window 1* is behind *Window 2* and is not active. The action is a click-in *Window 1*'s interior. The result is that *Window 1* is active, i.e., *Window 1* rectangle exists, *Window 1*'s interior exists, *Window 1*'s active-title-bar exists, *Window 1*'s close-box exists, etc. For each of these resulting perceptions the rule-learning algorithm creates a rule with the same preconditions and action. These rules are all valid. They express true correlations in the environment but not necessarily the most relevant cause for the effect. The most concise description of all the above effects is that the perceived rectangles are related to each other by being parts of a window. The fact that the window exists and is active is the "true" cause for the perceived parts of the window.

Once the agent learns the correlation in the perceptions one rule suffices to describe the correlated effects, such as the effects of bringing *Window 1* to the foreground. In addition to expressing the causes, rather than correlations, the world model is more concise. It contains fewer rules, and these rules express effects in a way that is easier for people to understand because it is more similar to people's descriptions of the environment.

This chapter presents an algorithm that learns new relations from correlated perceptions. The algorithm uses NOACTION rules learned by the rule-learning algorithm and converts NOACTION rules to a directed graph of correlated perceptions. The sets of correlated perceptions are strongly connected components in the digraph. Each strongly connected component collapses to a single node which becomes a new relation or new object (or both). Links between the collapsed nodes guide the creation of perceptions with new relations and new objects. Section 4.3 describes in detail the algorithm to collapse correlations with examples and results from the Macintosh Environment.

New relations and objects replace the underlying correlated perceptions in rules. Like any other rules the algorithm evaluates rules with new relations and objects and uses them to predict. Section 4.4 describes algorithms to create and use rules with new relations and objects.

4.1 Completely Correlated Perceptions are Important

The concept-learning algorithm in this chapter relies on a single crucial observation — perceptions that always occur together indicate a deeper structure in the environment. The agent should be aware of any correlation among perceptions. For example, when one perceptions is correlated with another such that the perception is true whenever

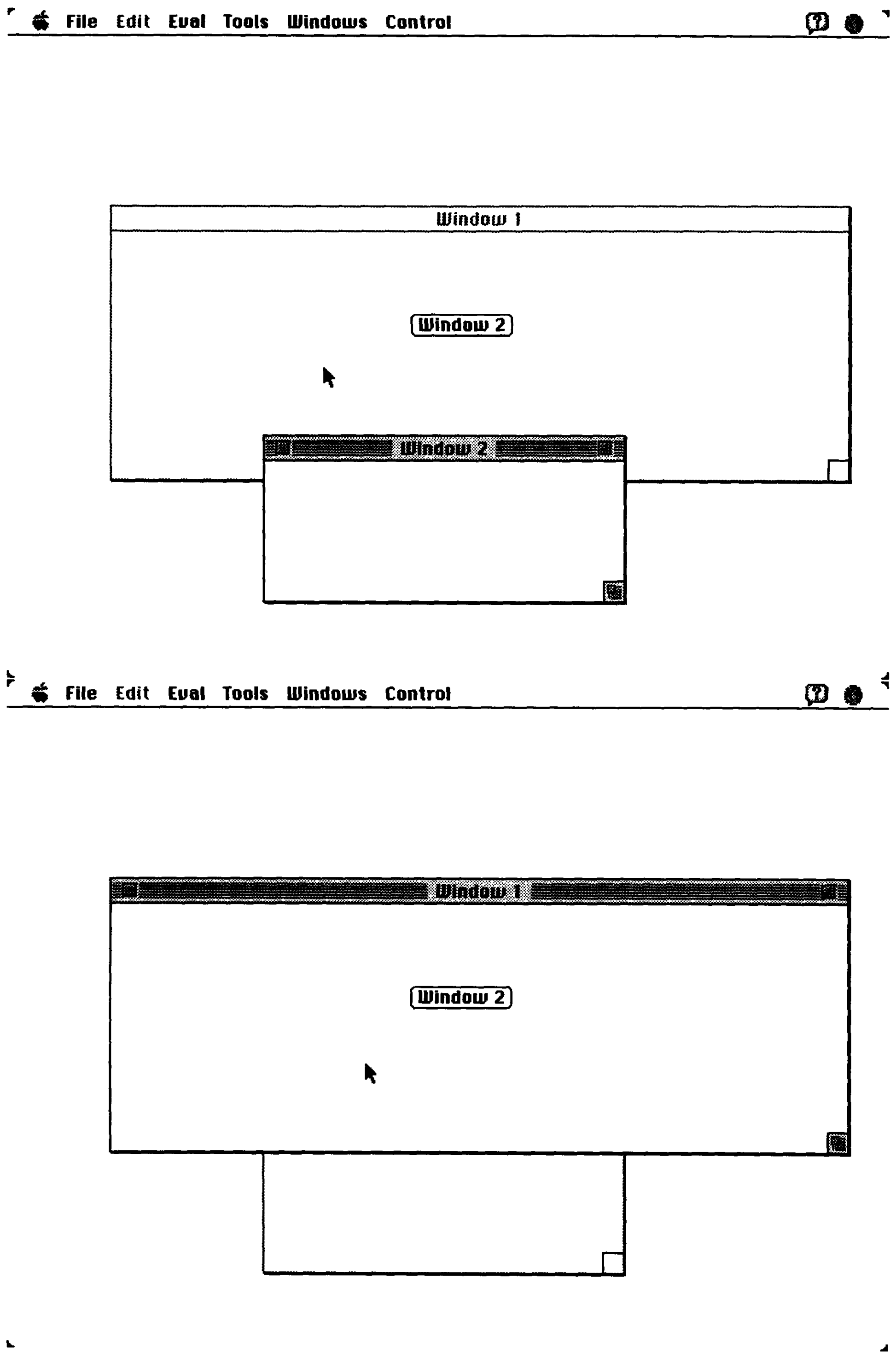


Figure 4-1: Macintosh screen situations with overlapping windows

the other perception is true, the agent gains a great deal of predictive power from knowing this correlation. If the agent knows that some perceptions are completely correlated (i.e., they always occur together) it can, of course, use this knowledge to predict. We observe, however, that the agent can extract much more information from completely correlated perceptions.

Usually when perceptions are completely correlated in an environment, there is a reason for the correlation. There is some underlying cause for all of the correlated perceptions which the agent may not yet grasp. The completely correlated perceptions occur because of the underlying cause of the events that follow the agent's action.

For example, in the Macintosh Environment, when the agent clicks in a window several perceptions appear together. Among them are the perceptions that the active-title-bar, close-box, and zoom-box are visible. Viewing the effects of the action as causing these perceptions is superficial. A more accurate account of the events is that clicking in the window makes the window active and when a window is active the active-title-bar, close-box, and zoom-box are visible. The revised explanation of the effects of the action uses the concept of an active window.

In this chapter the agent finds concepts, such as the concept of an active window, from completely correlated perceptions. We assume that whenever there are perceptions that always occur together there is an underlying cause for the correlation. The agent defines the underlying cause to be a new concept that expresses the cause of the completely correlated perceptions.

In the terminology of the world model, a new concept can be either a new object or a new relation on a new object. For example, a window rectangle and its interior rectangle always occur together. The perceptions that these two rectangles are visible define a new object "window". On the other hand, the concept that a window is active is a new relation, "active," on a new object, "window."

Section 4.3 presents an algorithm to find new relations and new objects. First an algorithm to find completely correlated perceptions is described. The completely correlated perceptions indicate that new concepts should be defined. Next we decide which concepts become new objects and which become new relations on new objects.

Before we look at the algorithm in detail, let us discuss the representation for new concepts in the world model.

4.2 Representing New Relations and Objects

The generality of the relation representation allows new relations to have the same form that perceptual relations have. When the learning algorithm creates a new relation it makes a unique symbol for this relation. Like the unique symbol $EXIST$, a new relation is represented by a symbol $NEWRELATIONxxx$ where xxx is a unique number assigned by the algorithm. The learning program allows a human observer to name the new relation. This name has no meaning for the learner but is useful for people trying to decipher the learned world model. Similarly when the learning algorithm creates a new object it makes a unique symbol $NEWOBJxxx$ which the user can name.

A new relation (such as $ACTIVE1$) on a new object (such as NEW-WINDOW1) is similar to a perception

$$ACTIVE1(\text{NEW-WINDOW1}) = T.$$

Like any other object the learner can have the pseudo-perception of the existence of a new object, such as $EXIST$(NEW-WINDOW1) $= T$. The rule-learning algorithm as well as prediction and action selection routines can use new relations just as it would use perceptions as part of rules.

The next section describes when and how the learning algorithm creates new relations and new objects.

4.3 Algorithm to Collapse Correlated Perceptions into New Relations and Objects

The algorithm to collapse correlations has two parts. The first finds the correlated perceptions from the NOACTION rules and the second uses the correlations to create new relations and new objects.

To find the completely correlated perceptions the algorithm relies on the NOACTION rules, since these rules state correlations of the form "if the precondition is true then the postcondition is true." Ideally the set of NOACTION rules would be complete, i.e., it would contain every valid NOACTION rule prior to learning new concepts. In this case the concept learning algorithm would be able to find all the correlated perceptions. More realistically, given the nature of the rule-learning algorithm, the set of NOACTION rules will be almost complete when the algorithm collapses correlations. Therefore it is important for the agent to wait until the set of NOACTION rules learned contains as many valid rules as possible prior to collapsing correlations. Since the agent cannot know what percentage of the valid rules it has learned, it waits until a specified number of trial has passed. This number of trials is predetermined by a parameter.

It is best for the algorithm to learn NOACTION rules exclusively for a fixed number of trials and to collapse correlations immediately after learning the NOACTION rules. Rules with NOACTION must be learned first to best achieve their function — preventing the creation of rules with correlated effects. Furthermore, the algorithm uses its time more efficiently if it collapses correlated perceptions before creating any rules (with actions) because after learning NOACTION rules it will have fewer perceptions to explain.

The agent can execute the algorithm for collapsing correlations repeatedly. Repeating this procedure may be useful if the algorithm finds additional valid NOACTION rules. The structure of the resulting new relations and new objects may be hierarchical (containing previously created new relations and objects as well as basic perceptions). This structure can be difficult to understand and can introduce redundancy instead of removing redundancy.

The two parts of the concept-learning algorithm that collapses correlation — finding completely correlated perceptions and creating new relations and new objects — are presented in the two following sections.

4.3.1 Finding Correlated Perceptions From NOACTION Rules

The algorithm to find correlated perceptions has as input a set of NOACTION rules. A NOACTION rule

$$\text{precondition} \rightarrow \text{NOACTION} \rightarrow \text{postcondition}$$

means that whenever the precondition is true the postcondition is also true. Therefore the NOACTION rules can be converted to a directed graph where the nodes correspond to perceptions or sets of perceptions and for each NOACTION rule there is a link in the graph from the precondition to the postcondition.

For example, consider the set of noaction rules in Figure 4-2. These rules are a subset of the NOACTION rules learned for the *EXIST* relation that deal with *Window 1*. The rules in Figure 4-2 are converted to the directed graph in Figure 4-3. Note, for example, a link from $EXIST(\textit{Window 1}\ \text{CB}) = T$ to $EXIST(\textit{Window 1}\ \text{ATB}) = T$ corresponding to the first rule

$$EXIST(\textit{Window 1}\ \text{CB}) = T \rightarrow \text{NOACTION} \rightarrow EXIST(\textit{Window 1}\ \text{ATB}) = T.$$

To complete the algorithm for finding correlations note that if we have the rules

$$EXIST(\textit{Window 1}\ \text{ATB}) = T \rightarrow \text{NOACTION} \rightarrow EXIST(\textit{Window 1}\ \text{CB}) = T$$

and

$$EXIST(\textit{Window 1}\ \text{CB}) = T \rightarrow \text{NOACTION} \rightarrow EXIST(\textit{Window 1}\ \text{ATB}) = T$$

then the perceptions $EXIST(\textit{Window 1}\ \text{ATB}) = T$ and $EXIST(\textit{Window 1}\ \text{CB}) = T$ are completely correlated. That is, they always occur together. In the corresponding graph the above rules make a cycle of two nodes. The graph can also have longer

```
 1. (success 28, probation NIL, reliability 1.0)
    EXIST (Window 1 CB) = T → NOACTION → EXIST (Window 1 ATB) = T

 2. (success 45, probation NIL, reliability 1.0)
    EXIST (Window 1) = T → NOACTION → EXIST (Window 1 INTERIOR) = T

 3. (success 28, probation NIL, reliability 1.0)
    EXIST (Window 1 ATB) = T
    → NOACTION → EXIST (Window 1 INTERIOR) = T

 4. (success 28, probation NIL, reliability 1.0)
    EXIST (Window 1 ATB) = T → NOACTION → EXIST (Window 1) = T

 5. (success 28, probation NIL, reliability 1.0)
    EXIST (Window 1 ATB) = T → NOACTION → EXIST (Window 1 ZB) = T

 6. (success 28, probation NIL, reliability 1.0)
    EXIST (Window 1 ATB) = T → NOACTION → EXIST (Window 1 CB) = T

 7. (success 28, probation NIL, reliability 1.0)
    EXIST (Window 1 ATB) = T → NOACTION → EXIST (Window 1 GB) = T

 8. (success 4, probation NIL, reliability 1.0)
    EXIST (Window 1 GB) = T → NOACTION → EXIST (Window 1) = T

 9. (success 45, probation NIL, reliability 1.0)
    EXIST (Window 1 INTERIOR) = T
    → NOACTION → EXIST (Window 1) = T

10. (success 16, probation NIL, reliability 1.0)
    EXIST (Window 1 TB) = T → NOACTION → EXIST (Window 1) = T
```

Figure 4-2: Some NOACTION rules for the EXIST relation on *Window 1*

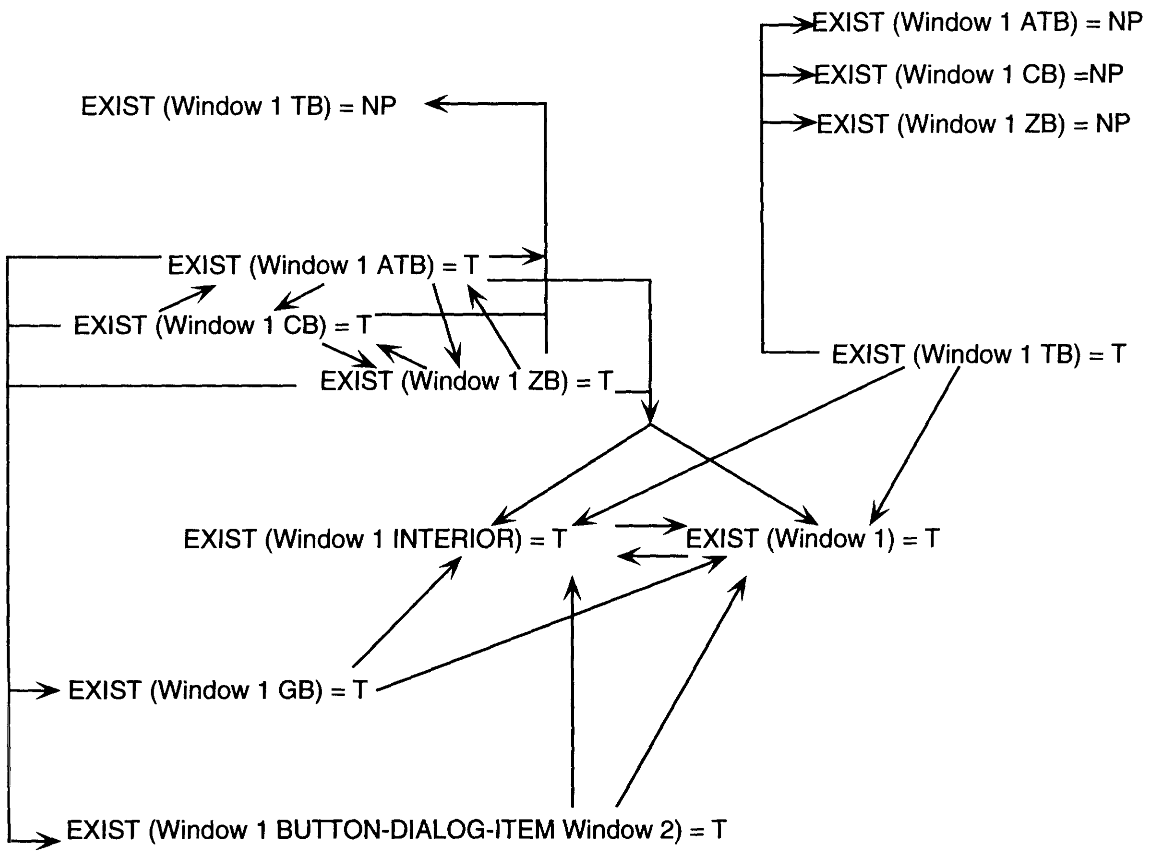


Figure 4-3: The correlation graph for the NOACTION rules for the *EXIST* relation on *Window 1*

cycles and structures of multiple cycles of correlated perceptions. In short, completely correlated perceptions show up in the correlation graph as strongly connected components. There are known algorithms to find strongly connected components of a graph efficiently (linear time in the number of nodes of the graph) (Baase 1988).

Figure 4-4 outlines the algorithm for finding correlated perceptions. Figure 4-5 shows the strongly connected components for the correlation graph. Note that *component 1* contains perceptions that are true when *Window 1* is active and *component 3* contains perceptions that are true whenever *Window 1* is present.

Algorithm 6 Find-Correlations*()*

```
[Make correlation graph from NOACTION rules]
For each NOACTION rule r
    make a directed link from precondition(r) to postcondition(r)
Find strongly connected components in the correlation graph
```

Figure 4-4: Algorithm to find correlated perceptions

4.3.2 Creating New Relations and New Objects

The algorithm to collapse correlated perceptions is shown in Figure 4-6. The algorithm looks for a link between two components, such as the link between *component 1* and *component 3* in Figure 4-5. Both components must contain at least two correlated perceptions so that the algorithm will not create redundant new objects and new relations.

It is natural to think of the meaning of links in the component graph as attribute-of links. Consider a link $a \rightarrow b$. Whenever a is present so is b, but when b is present a is not necessarily present. Therefore, a is one of several possible attributes of b. An attribute of an object becomes a relation on the object in the representation in this thesis. For example, when a is present, the relation $a(b) = T$ shows that the attribute a of b is true.

In Figure 4-5, *component 1* is an attribute of *component 3*, and *component 2* is another attribute of *component 3*. The algorithm, therefore, collapses *component 3*

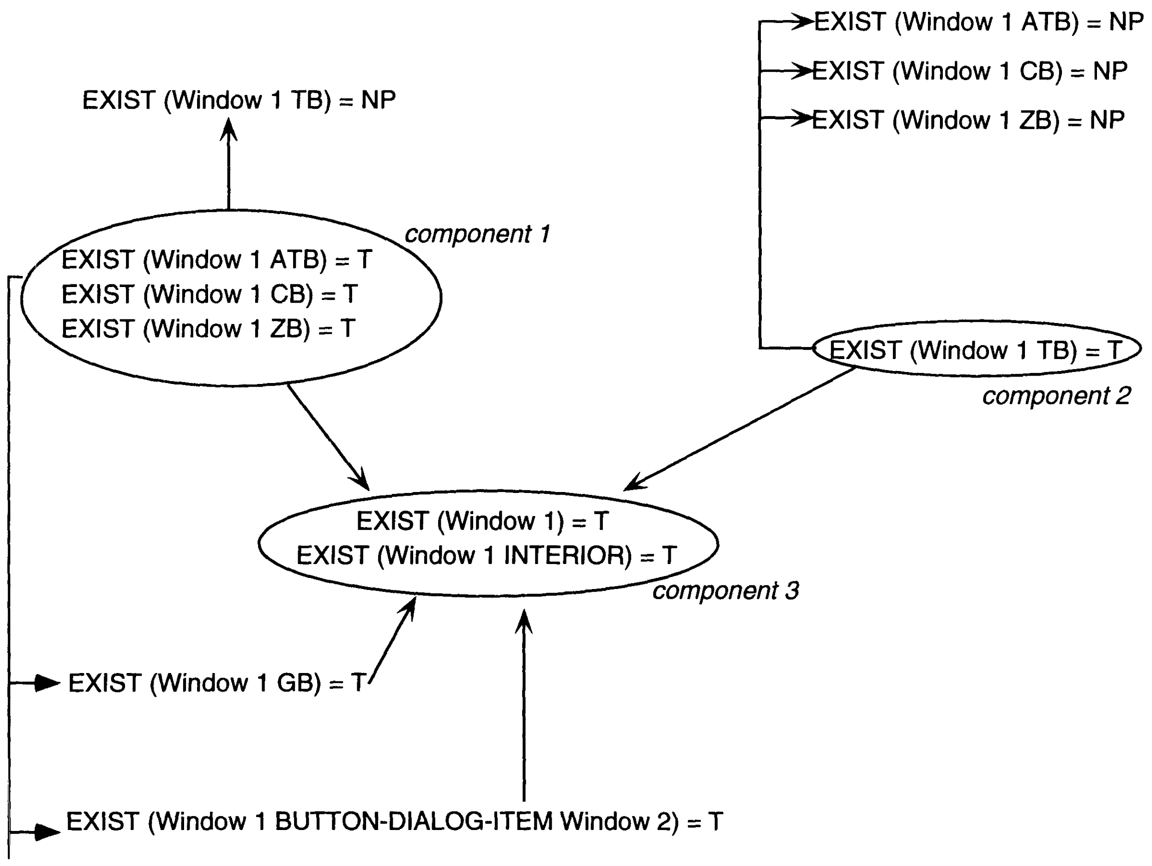


Figure 4-5: The component graph for the *EXIST* relation on *Window 1*

Algorithm 7 Make-New-Relations*()*

```
For each strongly connected component, c, in the correlation graph
    if c contains more than 1 perception then
        [make a new relation]
        Make a symbol for the new relation function — new-rel.
        Prompt the user for a name for the new relation.
        For each component c2 such that the component graph
            contains a link $c \longrightarrow c2$
            if c2 contains more than 1 perception then
                [make a new object]
                Make a symbol for the new object — new-obj.
                Prompt the user for a name for the new object.
                Make the defining NOACTION rule r for the new object
                Set $precondition(r) = perceptions\ in\ c2$
                Set $postcondition(r) = EXIST(new\text{-}obj) = T$
                Insert r into the rule set

                [Define the parts of the new object]
                For every precondition $p = exist(o)t$ in c2
                    Make NOACTION rule r with $precondition(r) = p$ and
                    $postcondition(r) = part\text{-}of(o, new\text{-}obj) = T$
                    Insert r into rule set

                Make the defining NOACTION rule r for the new relation
                Set $precondition(r) = perceptions\ in\ c$
                Set $postcondition(r) = new\text{-}rel(new\text{-}obj) = T$
                Insert r into the rule set
```

Figure 4-6: Algorithm to collapse correlated perceptions to new relations and new objects.

```
? (rule-table-count)
366
? (find-correlations)

(EXIST (Window~1 ZB) = T, EXIST (Window~1 CB) = T, EXIST (Window~1 ATB) = T)
"Enter name for relation: " active1

(EXIST (Window~1) = T, EXIST (Window~1 INTERIOR) = T)
"Enter name for object: " NEW-WINDOW1

(EXIST (Window~2 ATB) = T, EXIST (Window~2 CB) = T, EXIST (Window~2 ZB) = T)
"Enter name for relation: " active2

(EXIST (Window~2 INTERIOR) = T, EXIST (Window~2) = T, EXIST (Window~2 GB) = T)
"Enter name for object: " NEW-WINDOW2
NIL
? (replace-perceptions-in-rules)
NIL
? (rule-table-count)
325
```

Figure 4-7: A trace of an execution of the **Find-Correlations** and **Make-New-Relations** algorithms in the Macintosh Environment

to a new object, which I name NEW-WINDOW1 for clarity when examining the world model. *Component 1* becomes a new relation named $ACTIVE1$. Figure 4-7 shows a trace of the **Find-Correlations** and **Make-New-Relations** algorithms with the NOACTION rules for the $EXIST$ relation as input. The algorithm creates the new relation $ACTIVE1$ and the new object NEW-WINDOW1 as well as the corresponding relation $ACTIVE2$ and object NEW-WINDOW2.

We say that the new object NEW-WINDOW1 exists whenever the perceptions of *component 3* are true. To express this definition the algorithm makes a NOACTION rule

$$EXIST(\mathit{Window\ 1}) = T \wedge EXIST(\mathit{Window\ 1}\ \text{INTERIOR}) = T$$
$$\rightarrow \text{NOACTION} \rightarrow EXIST(\text{NEW-WINDOW1}) = T.$$

Similarly we say that the attribute $ACTIVE1$ of NEW-WINDOW1 is true whenever the perceptions of *component 1* are true. Note that the structure of the graph implies that whenever the perceptions of *component 1* are true so are the perceptions of *component 3*. Thus the rule

$$EXIST(\textit{Window 1}\ \text{ATB}) = T \wedge EXIST(\textit{Window 1}\ \text{CB} = T$$
$$\wedge EXIST(\textit{Window 1}\ \text{ZB}) = T)$$
$$\rightarrow \text{NOACTION} \rightarrow ACTIVE1(\text{NEW-WINDOW1}) = T$$

is sufficient to define the new relation $ACTIVE1$(NEW-WINDOW1).

The final aspect of creating new objects is recognizing parts of the new objects. When the algorithm creates a new object it recognizes that any perception in the component that states that an object exists indicates that the object is part of the new object. For example, the perceptions in *component 3* indicate that *Window 1* and *Window 1 interior* exist. These objects are recognized as parts of the new object NEW-WINDOW1. These relationships are captured in the rules

$$EXIST(\textit{Window 1}) = T \rightarrow \text{NOACTION} \rightarrow$$
$$\textit{PART-OF}(\textit{Window 1}, \text{NEW-WINDOW1}) = T$$

and

$$EXIST(\textit{Window 1}\ \text{INTERIOR}) = T \rightarrow \text{NOACTION} \rightarrow$$
$$\textit{PART-OF}(\textit{Window 1}\ \text{INTERIOR}, \text{NEW-WINDOW1}) = T.$$

The next section describes how new relations and new objects are incorporated into the rule-learning algorithm and as part of the world model.

4.4 Rules with New Relations and Objects

The advantage of a general perceptual representation is that when the agent adds new relations and new objects it does not need to change algorithms that deal with perceptions and rules. New relations on new objects, such as $ACTIVE1$(NEW-WINDOW1) = T, and $EXIST$ relations on new objects, such as $EXIST$(NEW-WINDOW1) = T, are true in the current state whenever the preconditions of the rules that define the new relations are true. For example, $ACTIVE1$(NEW-WINDOW1) = T is true whenever $EXIST(\textit{Window 1}\ \text{ATB}) = T$, $EXIST(\textit{Window 1}\ \text{CB}) = T$, and $EXIST(\textit{Window 1}\ \text{ZB}) = T$ are all true. Like external perceptions the new relations are in the list of current perceptions. These relations are always added to the current perceptions immediately after the agent perceives the environment.

Also like any external perception, a new relation or an $EXIST$ relation on a new object can be part of a rule's precondition or postcondition. For example, in the Macintosh Environment

$$ACTIVE1(\text{NEW-WINDOW1}) = T \rightarrow \text{NOACTION} \rightarrow$$
$$EXIST(\textit{Window 1}\ \text{ATB}) = T$$

and

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow ACTIVE1(\textit{NEW-WINDOW1}) = T$$

are valid rules with new relations.

The next three sections describe the algorithm that creates such rules, how it evaluates these rules and how it predicts from these rules.

4.4.1 Creating Rules with New Relations and Objects

When the learning algorithm creates new relations it first replaces collapsed perceptions in existing rules by the new relation. For example, the rule

$$EXIST(\textit{Window 1}\ \text{ATB}) = T \rightarrow \text{NOACTION} \rightarrow EXIST(\textit{Window 1}) = T$$

becomes

$$ACTIVE1(\text{NEW-WINDOW1}) = T \rightarrow \text{NOACTION} \rightarrow$$
$$EXIST(\text{NEW-WINDOW1}) = T.$$

The algorithm replaces collapsed perceptions in all the existing rules, not only the NOACTION rules.

When creating additional rules the algorithm does not use collapsed perceptions. Rather it uses the new relations. A number of learned rules with new relations are

```
1. (success 8, probation NIL, reliability 1.0)
   ACTIVE2 (NEW-WINDOW2) = T → NOACTION → EXIST (Window 1 ZB) = NP

2. (success 8, probation NIL, reliability 1.0)
   ACTIVE2 (NEW-WINDOW2) = T → NOACTION → EXIST (NEW-WINDOW2) = T

3. (success 24, probation NIL, reliability 1.0)
   NIL → click-in Window 1 INTERIOR → EXIST (NEW-WINDOW1) = T

4. (success 9, probation NIL, reliability 1.0)
   NIL → click-in Window 1 INTERIOR → ACTIVE1 (NEW-WINDOW1) = T

5. (success 36, probation NIL, reliability 1.0)
   NIL → click-in Window 1 CB → EXIST (NEW-WINDOW1) = NP

6. (success 11, probation NIL, reliability 1.0)
   X (Window 1, Window 2) = 1212
   → click-in Window 1 CB → ACTIVE2 (NEW-WINDOW2) = T

7. (success 21, probation NIL, reliability 1.0)
   NIL → click-in Window 1 TB → ACTIVE1 (NEW-WINDOW1) = T

8. (success 39, probation NIL, reliability 1.0)
   NIL → click-in Window 2 → ACTIVE2 (NEW-WINDOW2) = T

9. (success 11, probation NIL, reliability 1.0)
   ACTIVE1 (NEW-WINDOW1) = T
   → click-in Window 2 INTERIOR → EXIST (Window 1 TB) = T
```

Figure 4-8: Examples of a few learned rules with new relations and objects

shown in Figure 4-8. Recall that in addition to the example of learning the concepts *ACTIVE*1 and NEW-WINDOW1 used in this chapter, the algorithm learns the similar *ACTIVE*2 and NEW-WINDOW2 concepts, which appear in some of the rules in Figure 4-8.

The set of rules that results from replacing collapsed perceptions is smaller than the original set. For example, one rule

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow \textit{ACTIVE}1 \text{ (NEW-WINDOW1)} = T$$

replaces the three rules

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow \textit{EXIST}(\textit{Window 1} \text{ ATB}) = T$$

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow EXIST(\textit{Window 1}\ \text{CB}) = T$$

and

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow EXIST(\textit{Window 1}\ \text{ZB}) = T.$$

In Figure 4-7 we can see that the number of rules in the world model after replacing the collapsed perceptions with the new relations is smaller than the original number of rules. Furthermore, the rule above describes the concept of an active window — a key concept in the Macintosh Environment.

4.4.2 Evaluating Rules with New Relations and Objects

There is no difference between evaluating rules with new relations or objects and evaluating rules with only external perceptions. Recall that new relations that are true in a given state are added to the list of perceptions for that state. Specifically the new relations that are true in the current state are in current-perceptions and the new relations that were true in the previous state are in previous-perceptions. To evaluate the rules the learning algorithm uses the **Probabilistic-Rule-Reinforce** procedure from Chapter 3. This procedure checks if a rule's preconditions and postcondition are true in the previous and current state respectively, which is straightforward for both perceptions and new relations.

4.4.3 Predicting Using Rules with New Relations and Objects

Prediction using rules with new relations is unchanged from the prediction algorithm in Figure 3-8. The algorithm determines if a rule applies for prediction as usual. A rule that applies may have a new relation as a postcondition. For example, the rule

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow ACTIVE1(\textit{NEW-WINDOW1}) = T$$

applies following a click-in *Window 1* action. The prediction algorithm then predicts that the new relation $ACTIVE1(\text{NEW-WINDOW1}) = T$ will be true in the next

```
trial 758 rule count = 354 (on probation 157) mysteries 0
     Prediction: found relations 12, mistakes  0.0, missed 0, Total 12 :
     Smoothed Error 0.83

trial 759 rule count = 354 (on probation 157) mysteries 0
     Prediction: found relations 12, mistakes  0.0, missed 0, Total 12 :
     Smoothed Error 0.83

trial 760 rule count = 354 (on probation 157) mysteries 0
     Prediction: found relations 12, mistakes  0.0, missed 0, Total 12 :
     Smoothed Error 0.78

trial 761 rule count = 354 (on probation 157) mysteries 0
     Prediction: found relations 5, mistakes  0.0, missed 4, Total 9 :
     Smoothed Error 0.82
```

Figure 4-9: A trace of a few predictive trials for the $EXIST$ relation in the Macintosh Environment. The world model contains some new relations and objects.

state. It also predicts that the collapsed perceptions that define the new relation will be true (i.e., $EXIST(\textit{Window 1}\ \mathrm{ATB}) = T$, $EXIST(\textit{Window 1}\ \mathrm{CB}) = T$, and $EXIST(\textit{Window 1}\ \mathrm{ZB}) = T$).

Figure 4-9 shows a trace of a few prediction trials with a world model that includes new relations. The algorithm is learning and predicting the $EXIST$ relation only. The smoothed total error is close to the smoothed total error without new relations (in Figure 3-9).

4.5 Summary

This chapter presented an algorithm that learns concepts by collapsing correlated or redundant perceptions. It makes new relations and new objects that describe the environment more concisely than the underlying perceptions. In the Macintosh Environment we saw examples of learning important concepts that are similar to concepts people develop when using the Macintosh, e.g., "window" and "active."

Chapter 5

General Rules as New Concepts

A limitation of the world model learned by the rule-learning algorithm is its specificity. A rule generalizes over states of the environment, because its preconditions may apply in many states. The knowledge a rule contains, however, is true for specific objects in the environment. The set of specific rules does not group objects that behave similarly nor does it recognize that there are similar objects. For example, in an environment with two light switches the rule-learning algorithm learns that light switch 1 turns electricity on or off and that light switch 2 turns electricity on or off. This chapter presents an algorithm that creates rules for general objects, such as a light switch, rather than specific objects, such as light switch 1. Such general rules describe high-level characteristics of the environment.

As usual, the Macintosh Environment serves as an example. In this environment the concept of a window is a key concept for understanding the environment. We have examined many example situations with two windows (*Window 1* and *Window 2*). *Window 1* and *Window 2* share many characteristics that are true of any window. For example, a click in a window causes that window to be active. Similarly, objects such as a close-box or active-title-bar have characteristics that are true for any object of that type.

The algorithm in this chapter looks for rules that "match," i.e., they are the same except for the specific objects in the rules. For example, the two rules that indicate that a click in *Window 1* causes *Window 1* to be active, and that a click in *Window 2* causes *Window 2* to be active, match. To generalize these rules the algorithm replaces the specific objects with general objects.

The final step of the algorithm finds attributes of the general objects (i.e., relations on the general objects) by searching for perceptions of the specific objects in the original rule. In this process, the agent uses its perception as well as its current knowledge to drive the construction of the world model. This process is a natural cognitive process in people; it corresponds to observing the environment to find the reasons for an event. For example, the event of a rolling ball is explained by the observation that the ball is round. The generalizing algorithm adds as many attributes of the general objects as possible to the general rule to avoid over-generalization and nonsensical rules.

The algorithm to generalize rules is presented in Section 5.1. Like specific rules, general rules may or may not be valid so they must be evaluated. The procedure for evaluating general rules as well as for using general rules for prediction or action selection requires some change from these procedures for specific rules. Section 5.4 describes these procedures. Section 5.3 contains examples of general rules learned for the Macintosh Environment.

Rule generalization is exciting because learning research to date has not been successful at learning general concepts in a form people can understand. Neural networks (Rumelhart & McClelland 1986) are capable of generalizing gracefully from limited examples (such as the generalization from the characteristics of *Window 1* and *Window 2* to the characteristics of any window). The resulting representation of the learned knowledge (the network) is typically incomprehensible, except for small problems. The general rules that the algorithm in this chapter learns are similar to those a person might give to explain his knowledge about the Macintosh Environment.

The process of creating general rules by finding rules that match is similar to the generalization in (Berwick 1985) in the context of finding grammar rules for language from example sentences. Another related area of research involves learning first order predicates, such as $grandfather(x,y) \leftarrow father(x,z) \wedge father(z,y)$, (see, e.g., (Richards & Mooney 1992, Pazzani, Brunk & Silverstein 1991, Winston 1992)). The problem in both Berwick (1985) and learning first order predicates differs from the problem this chapter faces in that examples in this problem (rules) are noisy, additional examples are not available, and the database of attributes may be incomplete. The examples from which the algorithm in this chapter learns are the previously learned rules. Thus the number of examples is limited, and some of the example rules may be incorrect and may appear to match when they should not. Furthermore, attributes of objects are available from observations which are continually changing. Thus, relevant attributes may not be present.

5.1 An Algorithm to Learn General Rules

Before the agent can learn general rules, it must already have a good understanding of how specific objects behave. Since the rule generalization algorithm finds regularities in the environment by looking for similar specific rules, the set of specific rules must contain the rules to be generalized. The set of specific rules learned together with the current and previous perceptions are inputs to the rule-generalization algorithm — the specific rules are the examples of the general concept and the current and previous perceptions are used to find attributes of the objects in the rules, such as the *TYPE* of the objects.

Figure 5-1 shows the rule-generalization algorithm. Additional subroutines and utility functions are given in Figures 5-3 and 5-4. In the remainder of this section we will step through the **Generalize-Rules** algorithm with an example. As an example let us use the rules

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow \textit{EXIST}\ (\text{Window 1 } \textit{ATB}) = T$$

$$() \rightarrow \text{click-in } \textit{Window 2} \rightarrow \textit{EXIST}\ (\text{Window 2 } \textit{ATB}) = T.$$

It is obvious that these rules describe the same observation in two windows. Structurally they have the same template, modulo the specific object. A generalization would result in a valid and important observation about the environment. The current and previous perceptions at the time of generalization are shown in Figure 5-2.

Algorithm 8 Generalize-Rules*()*

```
For each rule r
    if r is not on probation and was not already matched
        let gr = make a general rule where each specific object in r
            is replaced by a general object.
        bind each general object in gr to the corresponding
            specific object in r.
        let attributes = Find-Attributes of general objects
                    due to binding with specific objects in r.
        For each rule r1
            if r1 is not on probation and was not already matched
                if r1 and r match
                    bind each general object in gr to the corresponding
                        specific object in r1
                    let more-attributes = Find-Attributes of general objects
                        due to binding with specific objects in r1.
                    if no attribute in more-attributes contradicts some attribute
                    in attributes
                        set attributes = attributes $\bigcup$ more-attributes.
        If at least one matching rule was found
            set preconditions(gr) = preconditions(gr) $\bigcup$ attributes.
            add gr to the rule set.
```

Figure 5-1: The **Generalize-Rules** Algorithm

Previous Perceptions	**Current Perceptions**
EXIST (Window 1) = T	EXIST (Window 1) = T
EXIST (Window 1 T = ATB	EXIST (Window 1 ATB) = T
TYPE (Window 1) = REC	TYPE (Window 1) = REC
TYPE (Window 1 ATB) = ATB	TYPE (Window 1 ATB) = ATB
OV (Window 1 ATB, Window 1) = T	OV (Window 1 ATB, Window 1) = T
OV (Window 1, Window 1 ATB) = F	OV (Window 1, Window 1 ATB) = F
X (Window 1, Window 1 ATB) = 33	X (Window 1, Window 1 ATB) = 33
(Y Window 1, Window 1 ATB) = 321	Y (Window 1, Window 1 ATB) = 321
EXIST (Window 2) = REC	
TYPE (Window 2) = REC	
OV (Window 1, Window 2) = T	

Figure 5-2: A subset of current and previous perceptions for a Macintosh screen situation where in the current screen situation *Window 1* is active and covers the entire screen and in the previous screen situation *Window 1* was active and *Window 2* was inactive.

(Figure 5-2 shows only a subset of the current and previous perceptions due to the large number of perceptions.)

The **Generalize-Rules** algorithm searches through all the rules and encounters the rule

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow EXIST\ (\text{Window } 1\ ATB) = T$$

which it refers to as r. It makes a general rule, called gr, in which specific objects are replaced with general objects.

$$() \rightarrow \text{click-in } x \rightarrow EXIST(y) = T.$$

The rule-generalization algorithm assigns general objects symbol names such as "genob53324". Throughout this chapter u, v, w, x, y, and z stand for general objects in order to make the rules more readable.

In the next step, the rule-generalization algorithm searches for attributes of the objects x and y by looking at the environment for perceptions of the corresponding specific objects *Window 1* and *Window 1* ATB. Figure 5-3 contains the algorithm to find attributes. In our example, it finds the perceptions

$$TYPE(\textit{Window 1}) = REC$$

Algorithm 9 Find-Attributes*(r,gr)*

```
Let the objects in r be ${o_1, \ldots, o_n}$.
Let the corresponding general objects in gr be ${go_1, \ldots, go_n}$.
Let attributes = $\emptyset$.
For i = min number of arguments of a relation
    to max number of arguments of a relation
    For every ordered sequence, S, of the objects ${o_1, \ldots, o_n}$,
        where S has length i
        For each relation, rel, that has i arguments
            if rel(S) has value v in current-perceptions then
                add rel(S) = v to attributes.
            elseif rel(S) has value v in previous-perceptions then
                add rel(S) = v to attributes.
Replace the specific objects in every attribute in attributes by the
corresponding general object
Return attributes
```

Figure 5-3: An algorithm to find attributes of general objects from perceptions

$$TYPE(\textit{Window 1}\ \text{ATB}) = ATB$$

$$OV(\textit{Window 1}\ \text{ATB}, \textit{Window 1}) = T$$

$$OV(\textit{Window 1}, \textit{Window 1}\ \text{ATB}) = F$$

$$X(\textit{Window 1}, \textit{Window 1}\ \text{ATB}) = 33$$

$$Y(\textit{Window 1}, \textit{Window 1}\ \text{ATB}) = 321$$

etc. To get attributes of the general objects x and y, the algorithm replaces the specific objects in each of the perceptions above, giving

$$TYPE(x) = REC$$

$$TYPE(y) = ATB$$

$$OV(y, x) = T$$

$$OV(x, y) = F$$

Algorithm 10 Match(*r*1,*r*2)

```
action(r1) and action(r2) are the same action or
    action(r1) and action(r2) can be bound to each other and
Match-Perception(postcondition(r1), postcondition(r2)) and
For each perception perc1 in precondition(r1)
    there is some perception perc2 in precondition(r2) such that
    Match-Perception(perc1, perc2).
```

Algorithm 11 Match-Perception(*p*1,*p*2)

```
perception-function(p1) = perception-function(p2) and
perception-value(p1) = perception-value(p2) and
For each pair of objects o1 and o2
    in perception-arguments(p1) and perception-arguments(p2) respectively
    If o1 or o2 is bound but not to each other
        then False
    If neither o1 nor o2 is bound
        then bind o1 and o2 to each other; True
        else False.
```

Algorithm 12 Perception-Contradicts(*p*1,*p*2)

```
perception-function(p1) = perception-function(p2) and
perception-arguments(p1) = perception-arguments(p2) and
perception-value(p1) ≠ perception-value(p2)
```

Figure 5-4: Utility functions for the rule-generalization algorithm

$$X(x, y) = 33$$

$$Y(x, y) = 321$$

etc. The algorithm saves these attributes in the set *attributes*. Note that the algorithm finds the redundant attribute $OV(x, y) = F$ as well as $OV(y, x) = T$. Similarly, it finds attributes for any ordering of the objects in any relation with more than one argument. For clarity, in this chapter we write the rules without the redundant attributes.

The inner loop of the rule-generalization algorithm searches through the set of rules for a rule that matches r. When it reaches

$$r1 = \; () \rightarrow \text{click-in } \mathit{Window\ 2} \rightarrow EXIST(\text{Window 2 } ATB) = T$$

it finds that $r1$ matches r. The **Match** function in Figure 5-4 successfully binds *Window 2* to *Window 1* and *Window 2* ATB to *Window 1* ATB, and returns true.

Next the algorithm looks for additional attributes for the general objects by binding x to *Window 2* and y to *Window 2* ATB. It finds the attribute

$$TYPE(x) = REC$$

only, because *Window 2* is not active in either the current or the previous states. This attribute does not contradict any of the known attributes (see the **Perception-Contradicts** function in Figure 5-4). The new set of attributes is the union of the set of attributes with the additional attributes. In our example the set of attributes

$$TYPE(x) = REC$$

$$TYPE(y) = ATB$$

$$OV(y, x) = T$$

$$X(x, y) = 33$$

$$Y(x, y) = 321$$

is unchanged.

If the additional attributes contradict the known attributes then the rule $r1$ would not be considered a match. It could be matched with a more appropriate rule. The inner loop, which matches rules, does not stop when it finds one matching rule. Rather it finds as many matches as possible, thereby finding more and more attributes for the general objects.

To finish the example, once the algorithm completes the inner loop it checks if a match was found. Since there was a match, the algorithm adds the attributes to the preconditions of the general rule and adds the resulting rule

$$TYPE(x) = REC \wedge TYPE(y) = ATB \wedge OV(y,x) = T \wedge X(x,y) = 33 \wedge Y(x,y) = 321$$
$$\rightarrow \text{click-in } x \rightarrow EXIST(y) = T$$

to the rule set. This rule states that a click in a rectangle object causes an active-title-bar to be present if the active-title-bar overlaps the rectangle and has the specified X and Y relations with the rectangle. This rule is valid and useful.

The above example brings an important issue to light. The algorithm does not find any attributes of the general object y due to $r1$. In the above example, the algorithm finds attributes of y due only to the rule r. As a worst case example, consider executing the rule generalization algorithm in a state where both the current and previous perceptions contain no windows. In this case the algorithm would not find any attributes of either x or y and would generate the rule

$$() \rightarrow \text{click-in } x \rightarrow EXIST(y) = T$$

from the two specific rules in the above example. This general rule would match to any two objects in the environment and would be wrong most of the time. For example, bind x to *Window 1* CB and y to *Window 1*. Because of such possible bindings this rule is invalid, and the evaluation algorithm will quickly remove it (see Section 5.4). The problem is that, as we saw previously, there is a valid rule resulting from generalizing these two specific rules. The algorithm missed this rule because at this time its perceptions are insufficient. Therefore, the rule-generalization algorithm must execute repeatedly in different environment states.

5.2 Generalizing Rules with New Relations

The example in the previous section generated a valid rule that predicts the presence of an active-title-bar after a click in a window. Recall that Chapter 4 describes an algorithm that learns that when the agent perceives a window's active-title-bar then the new relation — that the window is active — is true. A rule stating that a click in a window makes that window active describes a deeper understanding of the environment than the rule in the previous section. The prediction that the corresponding active title bar exists naturally follows from the definition of the new relation.

In general, not only in the Macintosh Environment, the rule generalization algorithm should use the knowledge captured by new relations rather than disregarding these concepts. This section describes the changes to the **Generalize-Rules** algorithm that allow it to deal with new relations using the example from the previous section. In this section, however, new relations replace the collapsed perceptions. The matching example rules are

$$() \rightarrow \text{click-in } \textit{Window 1} \rightarrow \mathit{ACTIVE}1\ \textit{(NEW-WINDOW1)} = T$$

$$() \rightarrow \text{click-in } \textit{Window 2} \rightarrow \mathit{ACTIVE}2\ \textit{(NEW-WINDOW2)} = T.$$

The previous and current relations are shown in Figure 5-5. They contains pseudo-perceptions of the new relations and new objects.

Adapting the **Generalize-Rules** algorithm to handle new relations requires no change to the main function; only some subroutines are changed. The modified subroutines are **New-Find-Attributes** in Figure 5-6 and **New-Match-Perception** in Figure 5-7.

Again, let us step through the **Generalize-Rules** algorithm. The algorithm finds the rule

$$r = \ () \rightarrow \text{click-in } \textit{Window 1} \rightarrow \mathit{ACTIVE}1\ (\text{NEW-WINDOW1}) = T$$

and makes the general rule

$$gr = \ () \rightarrow \text{click-in } x \rightarrow \mathit{ACTIVE}1(y) = T.$$

Next the algorithm looks for attributes of x and y due to r using **New-Find-Attributes**. It finds the attribute $\mathit{TYPE}(\textit{Window 1}) = \mathit{REC}$ and $\textit{PART-OF}\,(\textit{Window 1}, \text{NEW-WINDOW1}) = T$. There are no additional attributes so the general attributes are

$$\mathit{TYPE}(x) = \mathit{REC}$$

Previous Perceptions
EXIST (Window 1) = T
EXIST (NEW-WINDOW1) = T
TYPE (Window 1) = REC
PART-OF (Window 1 INTERIOR, NEW-WINDOW1) = T
PART-OF (Window 1, NEW-WINDOW1) = T
EXIST (Window 2) = T
EXIST (NEW-WINDOW2) = T
TYPE (Window 2) = REC
PART-OF (Window 2, NEW-WINDOW2) = T
PART-OF (Window 2 INTERIOR, NEW-WINDOW2) = T
PART-OF (Window 2 GB, NEW-WINDOW2) = T
OV (Window 1, Window 2) = T
Current Perceptions
EXIST (Window 1) = T
EXIST (NEW-WINDOW1) = T
TYPE (Window 1) = REC
PART-OF (Window 1 INTERIOR, NEW-WINDOW1) = T
PART-OF (Window 1, NEW-WINDOW1) = T
EXIST (Window 2) = T
EXIST (NEW-WINDOW2) = T

Figure 5-5: A subset of current and previous perceptions for a Macintosh screen situation, including new relations and new objects. In this screen situation *Window 1* is active and covers the entire screen and in the previous screen situation *Window 1* was active and *Window 2* was inactive.

Algorithm 13 New-Find-Attributes*(r,gr)*

```
Let the objects in r be objects = $\{o_1, \ldots, o_n\}$.
Let the corresponding general objects in gr be $\{go_1, \ldots, go_n\}$.
Let attributes = $\emptyset$.
For $i$ = min number of arguments of a relation
    to max number of arguments of a relation
    For every ordered sequence, $S$, of the objects $\{o_1, \ldots, o_n\}$,
        where $S$ has length $i$
        For each relation, rel, that has $i$ arguments
            if rel($S$) has value $v$ in current-perceptions then
                add rel($S$) = $v$ to attributes.
            elseif rel($S$) has value $v$ in previous-perceptions then
                add rel($S$) = $v$ to attributes.
For every object $o \in$ objects
    if the object $o$ is a new object and there is no attribute with relation
    PART-OF where $o$ is one of the arguments then
        find all perceptions with relation function PART-OF such that the
        object $o$ is one of the arguments
        add one of the PART-OF perceptions selected at random to attributes
        if there are any objects in the PART-OF perception that are not in
            $\{o_1, \ldots, o_n\}$ then
            make a general object for the new objects
            add the additional objects to the set objects.
Find attributes of the additional objects.
Replace the specific objects in every attribute in attributes by the
corresponding general object
Return attributes.
```

Figure 5-6: A modified algorithm to find attributes of objects and new objects.

and

$$\mathit{PART\text{-}OF}(x, y) = T.$$

In this example, there is an immediate connection between the two objects *Window 1* and NEW-WINDOW1 via a *PART-OF* relation. With other rules, such as

$$() \rightarrow \text{click-in } \mathit{Window\ 1} \text{ TB} \rightarrow \mathit{ACTIVE}\ (\text{NEW-WINDOW1}) = T,$$

the connection between the objects is not immediate. A third object must be added to establish the connection between this invented object and the ground objects. Therefore, for rules that contain new objects, the **Find-Attributes** algorithm uses a special procedure to find a connection through the *PART-OF* relation that the agent defines when it creates the new object. The algorithm looks for all the *PART-OF* relations it can perceive (in this case *PART-OF* (*Window 1*, NEW-WINDOW1) = T, and *PART-OF* (*Window 1* interior, NEW-WINDOW1) = T). It selects one of these perceptions at random. Recall that the rule generalizing algorithm executes repeatedly so eventually it will create rules with all possible *PART-OF* perceptions. Suppose the algorithm selects the perception

$$\mathit{PART\text{-}OF}(\mathit{Window\ 1}, \text{NEW-WINDOW1}) = T.$$

The algorithm then adds a new general object for the object *Window 1* to the objects in the general rule and looks for attributes of this object. Among other attributes it finds OV(*Window 1* TB, *Window 1*) = T which connects the title-bar with the new object NEW-WINDOW1.

Continuing with our example the rule-generalization algorithm next searches for a rule that matches r. When it encounters

$$r1 = \ () \rightarrow \text{click-in } \mathit{Window\ 2} \rightarrow \mathit{ACTIVE}2\ (\text{NEW-WINDOW2}) = T$$

it can match *Window 2* to *Window 1* and NEW-WINDOW2 to NEW-WINDOW1. The relations $ACTIVE1$ and $ACTIVE2$ are not equal but because these are new relations which themselves include specific objects they are different from other relation functions. The algorithm **New-Match-Perception** shows the situations in which

Algorithm 14 New-Match-Perception(p1,p2)

```
perception-value(p1) = perception-value(p2) and
(perception-function(p1) = perception-function(p2) or
perception-function(p2) is a general relation and
    perception-function(p1) fits the general relation perception-function(p2) or
perception-function(p1) and perception-function(p2) are new relations and
    perception-function(p1) and perception-function(p2) can be generalized) and
For each pair of objects $o1$ and $o2$
    in perception-arguments(p1) and perception-arguments(p2) respectively
    If $o1$ or $o2$ is bound but not to each other
        then False
    If neither $o1$ nor $o2$ is bound
        then bind $o1$ and $o2$ to each other; True
        else False.
```

Algorithm 15 Match-General-Relation(f1,f2)

```
If f1 is one of the relations that were generalized to create f2 then
return True
return False.
```

Figure 5-7: Matching perceptions with new relations and new objects

new relations generalize. Briefly, any set of new relations can generalize to form a new relation. The generalization is remembered so in the future the original relation can match the new relation or vise-versa in the matching procedure. In our example, the relation functions $ACTIVE1$ and $ACTIVE2$ generalize to a new relation, which I name $ACTIVE$. The general rule gr becomes

$$() \rightarrow \text{click-in } x \rightarrow ACTIVE(y) = T.$$

Having found a match the algorithm looks for attributes of x and y due to $r1$. Again it finds $TYPE(\textit{Window 2}) = REC$ and $PART\text{-}OF\ (\textit{Window 2},\ \text{NEW-WINDOW2}) = T$ which does not add to the set of attributes. The final set of attributes is, therefore,

$$TYPE(x) = REC$$

and

$$PART\text{-}OF(x, y) = T.$$

These attributes do not contradict the previous attributes so the resulting general rule is

$$TYPE(x) = REC \wedge PART\text{-}OF(x,y) = T \rightarrow \text{click-in } x \rightarrow ACTIVE(y) = T.$$

This rule is one of the rules Chapter 1 set as goals for the learning algorithm.

5.3 General Rules in the Macintosh Environment

This section describes some general rules learned for the Macintosh Environment. Because general rules are sometimes difficult to read, we look at each general rule with a trace of the specific rules that lead to the creation of the general rule. The following trace shows the creation of a simple general rule that states that a click in a close-box makes the close-box disappear.

$$NIL \rightarrow \text{click-in } \textit{Window 1}\text{ CB} \rightarrow EXIST(\textit{Window 1}\text{ CB}) = NP$$

$$NIL \rightarrow \text{click-in } \textit{Window 2}\text{ CB} \rightarrow EXIST(\textit{Window 2}\text{ CB}) = NP$$

$$TYPE(x) = CB \rightarrow \text{click-in } x \rightarrow EXIST(x) = NP$$

When the general rule is created it is placed on probation. Since this rule is valid, rule-evaluation accepts it and takes it off probation after some time (see Section 5.4 for a discussion regarding evaluating general rules). The above rule after evaluation has reliability 1.0 and is off probation.

(success 7, probation NIL, reliability 1.0)

$$TYPE(x) = CB \rightarrow \text{click-in } x \rightarrow EXIST(x) = NP$$

The following general rule describes that a click-in a close box makes the corresponding window not perceptible.

$$NIL \rightarrow \text{click-in } \textit{Window 1}\text{ CB} \rightarrow EXIST(\text{NEW-WINDOW1}) = NP$$

$$NIL \rightarrow \text{click-in } \textit{Window 2}\text{ CB} \rightarrow EXIST(\text{NEW-WINDOW2}) = NP$$

$$
\begin{gathered}
TYPE(z) = REC \wedge X(z,x) = 1221 \wedge Y(z,x) = 2211 \\
\wedge OV(zx) = F \wedge \mathit{PART\text{-}OF}(z,y) = T \wedge ACTIVE(y) = T \\
\wedge TYPE(x) = CB \wedge OV(x,z) = F \\
\rightarrow \text{click-in } x \rightarrow EXIST(y) = NP
\end{gathered}
$$

This rule uses some new objects and new relations. When creating this rule a third object (z) was added to connect the new object NEW-WINDOW1 with the *Window 1* CB. Recall that the algorithm finds the connection through the *PART-OF* relation. When creating the above rule it selected the perception *PART-OF*(*Window 1* INTERIOR, NEW-WINDOW1) $= T$, since the interior of a window is the only rectangle object with the attributes in the rule. Appendix A contains additional rules that explain the disappearance of the grow-box, zoom-box, and active-title-bar of a window due to a click in the window's close-box.

In this chapter we stepped through the generalization algorithm with example rules describing that a click in a window makes that window active. In fact the rule-generalization algorithm finds that the specific rules for a click in the interior of a window also match the two rules we used as an example. The following trace shows that four rules match to create the general rule that states that a click in a rectangle that is part of a window object makes that window object active. The algorithm creates the new general relation *ACTIVE* in the process of matching these rules.

$$NIL \rightarrow \text{click-in } \textit{Window 1} \text{ INTERIOR} \rightarrow ACTIVE1(\text{NEW-WINDOW1}) = T$$

$$NIL \rightarrow \text{click-in } \textit{Window 1} \rightarrow ACTIVE1(\text{NEW-WINDOW1}) = T$$

$$NIL \rightarrow \text{click-in } \textit{Window 2} \text{ INTERIOR} \rightarrow ACTIVE2(\text{NEW-WINDOW2}) = T$$

$$NIL \rightarrow \text{click-in } \textit{Window 2} \rightarrow ACTIVE2(\text{NEW-WINDOW2}) = T$$

$$TYPE(x) = REC \wedge PART - OF(x,y) = T \rightarrow \text{click-in } x \rightarrow ACTIVE(y) = T$$

The following general rule states the similar concept that a click in the title-bar of a window makes that window active. In this rule z is a window rectangle.

$$NIL \rightarrow \text{click-in } \textit{Window 1}\ \text{TB} \rightarrow ACTIVE1(\text{NEW-WINDOW1}) = T$$

$$NIL \rightarrow \text{click-in } \textit{Window 2}\ \text{TB} \rightarrow ACTIVE2(\text{NEW-WINDOW2}) = T$$

$$TYPE(z) = REC \wedge PART-OF(z, y) = T \wedge TYPE(x) = TB$$
$$\wedge X(x, z) = 33 \wedge Y(x, z) = 312 \wedge OV(x, z) = T$$
$$\rightarrow \text{click-in } x \rightarrow ACTIVE(y) = T$$

The remaining rules in this section describe effects of a click in one window on another window. The following rule describes that a click in a rectangle makes the active-title-bar of another window disappear.

$$NIL \rightarrow \text{click-in } \textit{Window 2}\ \text{INTERIOR} \rightarrow EXIST(\textit{Window 1}\ \text{ATB}) = NP$$

$$NIL \rightarrow \text{click-in } \textit{Window 2} \rightarrow EXIST(\text{Window 1 } \textit{ATB}) = NP$$

$$TYPE(y) = ATB \wedge OV(y, x) = F \wedge TYPE(x) = REC$$
$$\wedge X(x, y) = 1212 \wedge Y(x, y) = 2211 \wedge OV(x, y) = F$$
$$\rightarrow \text{click-in } x \rightarrow EXIST(y) = NP$$

Due to the OV, X, and Y relations in the above rule, it applies only in specific environment configurations, such as the configuration of Figure 5-8. Other similar rules describe the same effect in different configurations. For example the following rule applies when the active-title-bar overlaps the clicked rectangle in the bottom left hand corner.

$$NIL \rightarrow \text{click-in } \textit{Window 1}\ \text{INTERIOR} \rightarrow EXIST(\textit{Window 2}\ \text{ATB}) = NP$$

$$NIL \rightarrow \text{click-in } \textit{Window 1} \rightarrow EXIST(\text{Window 2 } \textit{ATB}) = NP$$

$$TYPE(y) = ATB \wedge X(y, x) = 1212 \wedge Y(y, x) = 2121 \wedge OV(y, x) = T$$
$$\wedge TYPE(x) = REC \rightarrow \text{click-in } x \rightarrow EXIST(y) = NP$$

The following rule describes an effect of a click in a zoom-box of a window on a rectangle object in another window — namely that the rectangle object disappears.

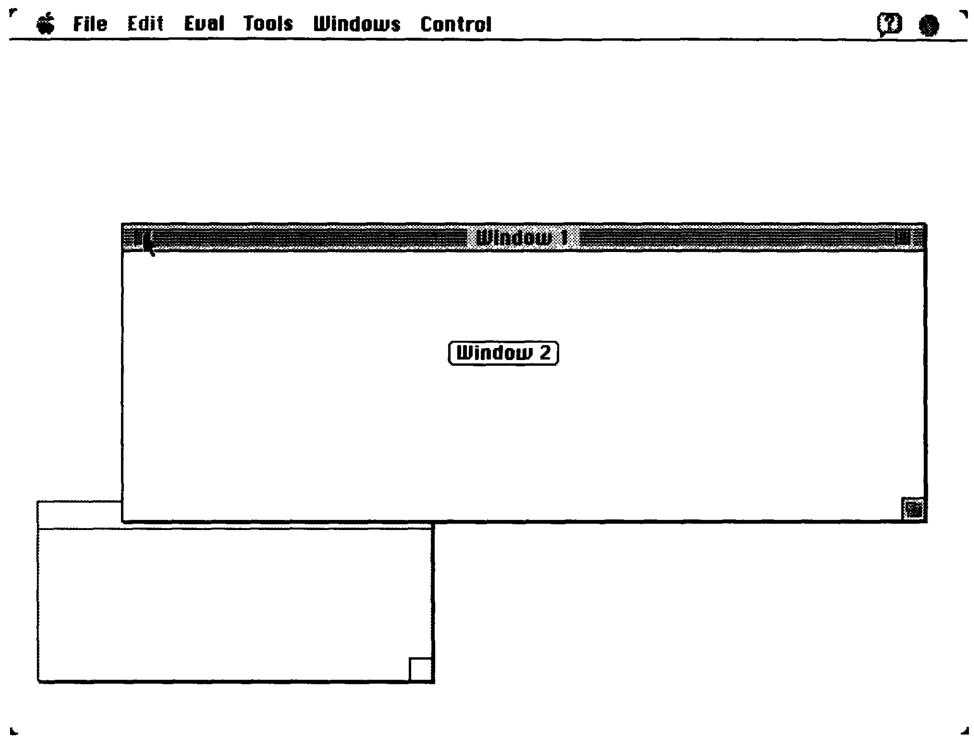


Figure 5-8: A screen situation where one window is below and to the left of another active window

$$EXIST(\text{NEW-WINDOW1}) = T \rightarrow \text{click-in } \textit{Window 2} \text{ ZB} \rightarrow$$
$$EXIST(\textit{Window 1}) = NP$$

$$EXIST(\text{NEW-WINDOW1}) = T \rightarrow \text{click-in } \textit{Window 2} \text{ ZB} \rightarrow$$
$$EXIST(\textit{Window 1} \text{ INTERIOR}) = NP$$

$$EXIST(z) = T \wedge TYPE(y) = REC \wedge PART\text{-}OF(y,z) = T$$
$$\wedge TYPE(x) = ZB \wedge X(x,y) = 2112 \wedge Y(x,y) = 2211 \wedge OV(x,y) = T$$
$$\rightarrow \text{click-in } x \rightarrow EXIST(y) = NP$$

All the above rules describe effects on the *EXIST* relation. Now let us examine a few general rules for the *OV* relation. The following rule states that if a window is partially covered by another rectangle then a click in the title-bar of the window makes the window rectangle overlap the other rectangle.

$$NIL \rightarrow \text{click-in } \textit{Window 2} \text{ TB} \rightarrow OV(\textit{Window 2}, \textit{Window 1} \text{ INTERIOR}) = T$$

$$NIL \rightarrow \text{click-in } \textit{Window 2} \text{ TB} \rightarrow OV(\textit{Window 2}, \textit{Window 1}) = T$$

$$TYPE(z) = REC \wedge OV(zy) = T \wedge TYPE(y) = REC \wedge TYPE(x) = TB$$
$$\wedge X(x,y) = 33 \wedge X(x,z) = 1212 \wedge Y(x,y) = 312 \wedge Y(x,z) = 2121$$
$$\wedge OV(x,y) = T \wedge OV(x,z) = F$$
$$\rightarrow \text{click-in } x \rightarrow OV(y,z) = T$$

The following rule states that after a click in a rectangle that is overlapped by a window the rectangle is not overlapped by the interior of the window.

$$EXIST(\textit{Window 2}) = T \rightarrow \text{click-in } \textit{Window 1} \text{ INTERIOR} \rightarrow$$
$$OV(\textit{Window 2} \text{ INTERIOR}, \textit{Window 1} \text{ INTERIOR}) = F$$

$$EXIST(\textit{Window 2}) = T \rightarrow \text{click-in } \textit{Window 1} \rightarrow$$
$$OV(\text{Window 2 } \textit{INTERIOR}, \text{Window 1}) = F$$

$$EXIST(z) = T \wedge TYPE(x) = REC \wedge X(x,z) = 2121 \wedge X(x,y) = 2121$$
$$\wedge Y(x,z) = 1212 \wedge Y(x,y) = 1122 \wedge OV(x,y) = F \wedge TYPE(y) = REC \wedge X(y,z) = 33$$
$$\wedge Y(y,z) = 213 \wedge OV(y,z) = T \wedge TYPE(z) = REC \wedge OV(z,x) = T$$
$$\rightarrow \text{click-in } x \rightarrow OV(y,x) = F$$

The last rule we will discuss is especially interesting since it is the second of the goal rules from Chapter 1. This rule states that if one rectangle is under another rectangle then a click in the bottom rectangle brings that rectangle to the front.

$$OV(\textit{Window 1} \text{ INTERIOR}, \textit{Window 2} \text{ INTERIOR}) = F$$
$$\rightarrow \text{click-in } \textit{Window 1} \text{ INTERIOR} \rightarrow$$
$$OV(\textit{Window 1} \text{ INTERIOR}, \textit{Window 2} \text{ INTERIOR}) = T$$

$$OV(\textit{Window 1}, \textit{Window 2} \text{ INTERIOR}) = F \rightarrow \text{click-in } \textit{Window 1} \rightarrow$$
$$OV(\text{Window 1, Window 2 } \textit{INTERIOR}) = T$$

$$TYPE(y) = REC \wedge TYPE(x) = REC \wedge X(x,y) = 2121$$
$$\wedge Y(x,y) = 1212 \wedge OV(x,y) = F \rightarrow \text{click-in } x \rightarrow OV(x,y) = T$$

Appendix A contains some more examples of general rules that the **Generalize-Rules** algorithm creates.

Algorithm 16 Test-All-Bindings(gr, test)

```
Let GeneralObjects = all the objects in the rule gr.
Let ObjectList = all the known specific objects.
For each go in GeneralObjects
    Let PossibleObjects(go) = the objects in ObjectList that can be bound to go
    s.t. gr applies in the previous state
For every possible binding of the objects in GeneralObjects
    test gr
```

Figure 5-9: An algorithm to test a general rule on every possible binding to specific objects in the environment

5.4 Evaluating and Using General Rules

General rules that the **Generalize-Rules** algorithm generates are not guaranteed to be valid. The rules are often overly general because the current and previous perceptions at the time of generalization did not contain enough attributes for the general objects. Therefore, new general rules, like specific rules, are on probation initially. The algorithm evaluates their validity with tests in the environment. Like the rule-evaluation algorithm in Chapter 3, a rule succeeds when its preconditions and actions apply in the previous state and its postcondition is true in the current state. Of course, a perception with general objects is never true in the current state. The general objects must be bound to specific objects before the rule can be evaluated. Likewise to use a general rule for prediction or goal oriented action selection the general objects must be bound to specific objects.

Furthermore, a general rule is valid if for every possible binding of specific objects to the general objects the resulting specific rule is valid. Therefore, to evaluate a general rule the algorithm must evaluate all the specific rules resulting from every possible binding of specific objects. The algorithm **Test-All-Bindings** in Figure 5-9 evaluates all the specific rules when the input **test** is the **Probabilistic-Rule-Reinforce** function from Chapter 3. **Test-All-Bindings** can predict from all possible bindings of specific objects when **test** is the rule prediction function.

The operation of testing every possible binding of specific objects to the general objects is exponential in the number of general objects. If the environment contains n specific objects and a general rule contains k general objects, then the number of possible bindings is n^k. The **Test-All-Bindings** procedure reduces the possibilities somewhat by checking partial bindings and abandoning them if they result in a rule that does not apply. For example, if a rule contains the perception $TYPE(u) = ZB$ then an assignment of *Window 1* CB to u immediately results in a rule that does not apply since the $TYPE$ of *Window 1* CB is perceived to be CB. All possible bindings of other objects where *Window 1* CB is bound to u are abandoned.

This heuristic reduces the number of possibilities, but the operation remains exponential. Therefore, evaluating and predicting from general rules are time-consuming operations. The search problem with using general rules of this kind is a known problem in AI (see (Winston 1992) for a discussion). On the other hand, the number of general objects in a rule is typically not very large (for example, the algorithm generated no general rule with more than five general objects). Thus the search space is large but manageable.

Naturally, after general rules are accepted, the learning algorithm saves the general rules and uses them to remove the specific rules that match them (usually the rules used to create the general rule). This operation reduces the number of rules and makes a concise and readable world model.

5.5 Discussion

The purpose of rule generalization is to learn a world model that is not specialized to particular objects in the environment. The generalized model should apply to new objects that are not familiar to the agent. The question that remains is whether the rule-generalization algorithm can learn a complete, general world model.

We have seen the format of the general rules this algorithm learns. Therefore we know that the rules describe behaviors, in the environment, that are not specific to objects. We have also discussed algorithms that use the general rules to predict and plan. These algorithms are straightforward, although time consuming.

The representation of the world model, then, is sufficiently general, but we still have not answered our previous question. Can the rule-generalization algorithm learn a complete general model? The answer is yes — I believe — but not yet.

I believe that the algorithm is general enough that, given enough time and enough example environments, it can learn a complete, general model. Additional environments would give the algorithm more examples to generalize from and more time is needed to test the general rules created and to repeat the rule-generalization procedure.

At this point in the research there has not been enough time to learn a complete general model of the Macintosh Environment. The algorithm has had access to specific rules from environments with two windows, *Window 1* and *Window 2*, which means that the number of example rules for any general rule is small (often less than two example rules). Therefore many general rules are missed. Furthermore the time to learn has been limited. The current state of the world model is a combination of specific and general rules. This model is useful for prediction and planning in an environment with *Window 1* and *Window 2.*

In environments with other windows, e.g. *Window 4* and *Window 5*, the general rules will apply, but there are aspects of the environment (with the new windows) that are not explained by any general rule. These aspects of the environment are unexplained. In terms of evaluating the world model with prediction, the results would be better than prediction with no rules, but not as good as prediction with the specific model or the model with both specific and general rules.

A related question is how well the world model, with a combination of specific and general rules, explains an environment with the two familiar windows (*Window 1* and *Window 2*) and a third window (*Window 3*). This question has two facets. The first problem is explaining any event with *Window 3*, which is similar to the problem we discussed previously. Some aspects of the behavior of a window are captured with general rules so the world model can explain some of the events involving *Window 3*, but not all of them. The second difficulty is the interaction of three windows which has some behavior that is different, even contradictory, to the behavior of a two window environment. The three window interaction cannot be explained by a model trained only in a two window environment. The agent must train in a three window environment, learn specific rules, and generalize these rules before we can expect to find a complete general world model for three window environment.

5.6 Summary

This chapter presented a new algorithm that uses specific world knowledge (rules) and observations to learn general concepts about the environment. The learned concepts are represented as rules with relations on general objects. A general object can be bound to any specific object in the environment resulting in concepts that are true for multiple specific objects. Experiments of rule generalization in the Macintosh Environment result in concepts that are much like people's description of the Macintosh window interface.

Chapter 6

Picking the Best Expert from a Sequence

Suppose you are looking for an expert, such as a stock broker. You have limited resources and would like to efficiently find an expert who has a low error rate. There are two issues to face. First, when you meet a candidate expert you are not told his error rate, but can only find this out experimentally. Second, you do not know a priori how low an error rate to aim for. This chapter presents an algorithm to find a good expert given limited resources, and show that the algorithm is efficient in the sense that it finds an expert that is almost as good as the expert you could find if each expert's error rate was stamped on his forehead (given the same resources).

If each expert's error rate were stamped on his forehead then finding a good expert would be easy. Simply examine the experts one at a time and keep the one with the lowest error rate. If you may examine at most n experts you will find the best of these n experts, whose expected error rate we denote by b_n. You cannot do any better than this without examining more experts.

Since experts do not typically come marked with their error rates, you must test each expert to estimate their error rates. We assume that we can generate or access a sequence of independent experimental trials for each expert.

If the number of available experts is finite, you may retain all of them while you test them. In this case the interesting issues are determining which expert to test next (if you cannot test all the experts simultaneously), and determining the best expert given their test results. These issues have been studied in reinforcement learning literature and several interesting algorithms have been developed (see Watkins (1989), Sutton (1990), Sutton (1991), and Kaelbling (1990) for some examples).

Here we are interested in the case where we may test only one expert at a time. The problems in this case are: (1) what is the error rate of a "good" expert, and (2) how long do we need to test an expert until we are convinced that he is good or bad?

First consider the case that we have a predetermined threshold such that an error rate below this threshold makes the expert "good" (acceptable). This is a well-studied statistical problem. There are numerous statistical tests available to determine if an expert is good; we use the ratio test which is the most powerful among them. The ratio test is presented in Section 6.2.1.

However, in our problem formulation we have no prior knowledge of the error rate distribution. We thus do not have an error-rate threshold to define a good expert, and so cannot use the ratio test. The algorithm in Section 6.2.2 overcomes this limitation by setting lower and lower thresholds as it encounters better experts. Section 6.2 contains the main result of this paper: our algorithm finds an expert whose error rate is close to the error rate of the best expert you can expect to find given the same resources.

Section 6.3 presents a similar expert-finding algorithm that uses the sequential ratio test (Wald 1947) rather than the ratio test. Wald (1947) shows empirically that the sequential ratio test is twice as efficient as the ratio test when the test objects are normally distributed. While the theoretical bound for the sequential-ratio expert-finding algorithm is weaker than the bound for the ratio-test expert-finding algorithm, empirical results with specific distributions in Section 6.4 indicate that the former algorithm performs better in practice.

6.1 An AI Application: Learning World Models

The expert-finding problem is related to the problem of learning a causal world model in this thesis. Recall that the world model is a set of rules of the form

$$\text{precondition} \rightarrow \text{action} \rightarrow \text{postcondition}$$

with the meaning that if the preconditions are true in the current state and the action is taken, then the postcondition will be true in the next state.

An algorithm to learn rules uses triples of previous state, S, action, A, and current state to learn. It may isolate a postcondition, P, in the current state, and generate preconditions that explain the postcondition from the previous state and action. For any precondition PC that is true in state S, the rule $\mathrm{PC} \rightarrow \mathrm{A} \rightarrow \mathrm{P}$ has some probability p of predicting incorrectly. To learn a world model, the algorithm must find the rules with low probability of prediction error, and discard rules with high probability of prediction error. Unlike the world model in the rest of this thesis, which contains many rules for any action and postcondition pair, this section attempts to find exactly one *best* rule for each action and postcondition pair.

The problem of finding a good rule to describe the environment is thus an expert finding problem. It fits into the model discussed here since (1) each rule has an unknown error rate, (2) the distribution of rules' error rates is unknown and depends both on the environment and the learning algorithm, and (3) the learning algorithm can generate arbitrarily many rules.

6.2 Finding Good Experts from an Unknown Distribution

First, let us reformulate the expert-finding problem as a problem of finding low error-rate coins from an infinite sequence $c_1, c_2, \ldots$ of coins, where coin c_i has probability r_i of "failure" (tails) and probability $1 - r_i$ of "success" (heads). The r_i's are determined by independent draws from the interval $[0, 1]$, according to some unknown distribution. We want to find a "good" coin, i.e. a coin with small probability r_i of failure (error). We are not given the r_i's, but must estimate them using coin flips (trials).

The main result of this section is:

Theorem 5 *There is an algorithm (algorithm* **FindExpert***) such that when the error rates of drawn coins are unknown quantities drawn from an unknown distribution, after t trials, with probability at least $1-1/t$, we expect to find a coin whose probability of error is at most $b_{t/\ln^2 t} + O(\frac{1}{\sqrt{\ln t}})$.*

This theorem states that after t trials, we expect the algorithm to find an expert that is almost as good as the best expert in a set of $t/\ln^2 t$ randomly drawn experts (who would have expected error rate $b_{t/\ln^2 t}$). Note that this result depends in a natural manner on the unknown distribution.

Recall that in t trials if the experts' error rates are known we can find the best of t experts' error rates (b_t). Compared to this, our algorithm must examine fewer experts because it must spend time estimating their error rates. For some distributions (such as for fair coins) $b_{t/\ln^2 t}$ and b_t are equal, while for other distribution they can be quite far apart.

The rest of this section gives the ratio test, the algorithm for finding a good expert, and the proof of Theorem 5

6.2.1 The Ratio Test

Since we do not know the error rates of the coins when we draw them, we must estimate them by flipping the coins. If we knew that "good" coins have error rate at most p_1, we could use standard statistical tests to determine if a coin's error rate is above or below this threshold. Because it is difficult to test coins that are very close to a threshold, we instead use the ratio test, which tests one hypothesis against another. In this case the hypotheses are that the coin has error rate at most p_0, versus that the coin has error rate at least p_1, where p_0 is a fixed value less than p_1.

The Problem Given a coin with unknown rate of failure p.

Test if $p \leq p_0$ vs. $p \geq p_1$. Accept if $p \leq p_0$. Reject if $p \geq p_1$.

Requirements The probability of rejecting a coin does not exceed α if $p \leq p_0$, and the probability of accepting a coin does not exceed β if $p \geq p_1$. [1]

The Test Let m be the number of samples, and f_m be the number of failures in m samples. The likelihood ratio is the probability of f_m failures under the hypothesis that $p = p_0$ (H_0), over the probability of f_m failures under the hypothesis that $p = p_1$ (H_1). The test rejects if this ratio is smaller than a predetermined threshold. For Bernoulli trials the ratio test is equivalent to testing if

$$f_m \geq C_m$$

where C_m is some constant.

Due to the requirement that $\Pr\{\text{reject } H_0 | H_0 \text{ true}\} \leq \alpha$, and using Chernoff bound we can show that the ratio test becomes

$$\begin{array}{ll} reject & \text{if } f_m \geq (p_0 + \sqrt{\frac{\ln 1/\alpha}{2m}})m \\ accept & \quad otherwise. \end{array}$$

The Sample Size From the requirements that $\Pr\{acceptH_0 | H_0 false\} \leq \beta$, and $C_m = (p_0 + \sqrt{\frac{\ln 1/\alpha}{2m}})m$, using Chernoff bounds we find the necessary number of samples

$$m \geq \frac{\left(\sqrt{\ln 1/\alpha} + \sqrt{\ln 1/\beta}\right)^2}{2(p_1 - p_0)^2}.$$

The Probability of Accepting a Coin Again using Chernoff bounds we can compute the following bounds on the probability that a coin with probability of failure p will be accepted.

$$\begin{aligned} \Pr\{accept|p\} &= \Pr\{f_m < (p_1 - k)m|p\} \\ &< \exp\{-2m(p - p_1 + k)^2\} \end{aligned}$$

[1]We choose the ratio test since it has the most power, i.e., for a given α, i.e. it gives the least β (probability of accepting when the hypothesis H_0 is wrong (see (Rice 1988).)

$$
\begin{aligned}
&= \exp\{-2m(p-p_1)(p-p_1+2k) - \ln 1/\alpha\} \text{ if } p \geq p_1 - k \\
\Pr\{accept|p\} &= 1 - \Pr\{reject|p\} \\
&= 1 - \Pr\{f_m > (p_1-k)m|p\} \\
&> 1 - \exp\{-2m(p-p_1+k)^2\} \\
&= 1 - \exp\{-2m(p-p_1)(p-p_1+2k) - \ln 1/\alpha\} \text{ if } p \leq p_1 - k
\end{aligned}
$$

Where $k = \sqrt{\frac{\ln 1/\alpha}{2m}}$.

6.2.2 An Algorithm for Finding a Good Expert

We know how to test if a coin is good given a threshold defining a good error rate, but when we do not know the error-rate distribution we can not estimate the lowest error rate b_t that we can expect to achieve in t trials. The following algorithm overcomes this handicap by finding better and better coins and successively lowering the threshold for later coins.

The algorithm for finding a good coin is the following.

Algorithm 17 FindExpert
Input: t, an upper bound on the number of trials (coin flips) allowed.

```
Let BestCoin = Draw a coin.
Flip BestCoin $\ln^3 t$ times to find $\hat{p}$.
Set $p_1 = \hat{p}$.
Repeat until all $t$ trials are used
    Let $p_0 = p_1 - \epsilon(p_1)$, where $\epsilon(p_1) = \sqrt{4/\ln(t)}$.
    Let Coin = Draw a coin.
    Test Coin using the ratio test:
        Flip Coin $m = \ln^2 t$ times.
        Accept if $f_m < (p_1 - \epsilon(p_1)/2)m$.
    If the ratio test accepted then
        Set BestCoin = Coin.
        Flip BestCoin an additional $\ln^3 t$ times to find an improved $\hat{p}$.
        Set $p_1 = \hat{p}$.
Output BestCoin.
```

6.2.3 Efficiency of Algorithm FindExpert

This section proves Theorem 5. The following outline clarifies the main steps of the proof which is long and detailed.

Description of the proof: Since the error-rate distribution is unknown, we do not have any estimate of b_t, so the algorithm uses better and better estimates. It starts with a random coin and a good estimate of its error rate. It prepares a test to determine if a new coin is better than the current coin (with high probability). Upon finding such a coin it prepares a stricter test to find a better coin, and so on. We show that *the time to test each coin is short*, and thus *we see many coins.* Since *we almost always keep the better coin* we can *find a coin whose error rate is at most the expected best error rate of the the algorithm saw (plus a small correction).*

Lemma 3 shows that the ratio test with $\ln^2 t$ samples fulfills the required probability bounds on erroneous acceptances and rejections.

Lemma 3 $\ln^2 t$ *samples are sufficient for the ratio test with parameters* p_1, $p_0 = p_1 - \epsilon$, $\alpha = \beta = 1/t^2$, *and* $C = (p_0 + \frac{\epsilon}{2})m$.

Proof: *With these parameter values, a sufficient sample size is*

$$m \geq \frac{\left(\sqrt{\ln 1/\alpha} + \sqrt{\ln 1/\beta}\right)^2}{2\epsilon^2} = \ln^2 t.$$

Now let us consider the effects of estimating the error probability of the best coins. One effect is that an estimated error probability that is lower than the true error probability gives us a tougher than necessary test. In other words, we are likely to reject a better coin that lies in the range $[\widehat{p}, p]$. The following lemma shows that this range is small compared to the testing gap, ϵ.

Lemma 4 *With probability* $1 - 1/t^2$, *estimating the error probability of a coin with* $\ln^3 t$ *coin tosses gives a testing gap of size* $O(\epsilon) = O(1/\sqrt{\ln t})$.

Proof:

If the estimate of the error probability of the coin, $\widehat{p}$, *is smaller than* p, *the true error probability of the coin, then the true testing gap for the test is larger than* ϵ *because* *the ratio test is only guaranteed to accept coins with error probability* $\leq p_0$. *Thus the true acceptance gap for this test is the range* $[p_0, p]$ *which has size*

$$g = \epsilon + (p - \widehat{p}).$$

After testing the coin for $\ln^3 t$ *trials the standard deviation of the estimate* $\widehat{p}$ *is* $\sigma_{\widehat{p}} = \sqrt{\frac{p(1-p)}{\ln^3 t}} \leq \frac{1}{2\sqrt{\ln^3 t}}$. *With probability* $1 - 1/t^2$, $\widehat{p}$ *is not farther than* $O(\sqrt{\ln t})$ *standard deviations from* p *(see Appendix B for the proof.) Thus, with probability* $1 - 1/t^2$,

$$g \leq \epsilon + \sqrt{\ln t}\,\sigma_{\widehat{p}} \leq \epsilon + \sqrt{\ln t}\frac{1}{2\sqrt{\ln^3 t}} = \epsilon + \frac{1}{2\ln t} = O(\epsilon).$$

If the estimate $\widehat{p} \geq p$ *then clearly* $g = O(\epsilon)$.

When the true error probability (p) is smaller than the estimated error probability, p lies in the testing range $[p_0, p_1]$. Recall that in this range we have a reasonable chance of accepting or rejecting. Thus we have a reasonable chance of accepting coins in the range $[p, p_1]$ which are worse than the current coin. Lemma 5 shows that since p is close to $\widehat{p}$ the probability of accepting coins in that range remains small.

Lemma 5 *With probability* $1 - 1/t^2$, *estimating the error probability of a coin with* $\ln^3 t$ *coin tosses gives the ratio test an* $O(\frac{t^{\frac{2}{\sqrt{\ln t}}}}{t^2})$ *probability of accepting a coin with error probability greater than the best coin's error probability.*

Proof: *If the estimate* $\widehat{p} \leq p$ *then it is clear that the probability of accepting a coin with error probability greater than* p *is at most* $\beta = 1/t^2$ *since the probability of accepting a coin with error probability* p *is monotonically decreasing with increasing* p.

If $p < \widehat{p}$ *there is a higher probability of accepting coins with error probability less than* p, *because* p *is in the range* $[p_0, p_1]$. *The probability of accepting any coin in the range* $[p, p_1]$ *is at most* $\Pr\{accept|p\}$, *which is the value we would like to compute but cannot since we do not know* p.

Appendix B shows that with probability $1 - 1/t^2$, $\widehat{p}$ *is within* $O(\sqrt{\ln t})$ *standard deviations of* p. *So we want to compute* $\Pr\left\{accept|(p_1 - \sqrt{\ln t}\,\sigma_{\widehat{p}})\right\}$.

Since $p_1 - \sqrt{\ln t}\sigma_{\widehat{p}} \geq p_1 - \frac{1}{2\ln t} > p_1 - k$, for the parameter settings of the ratio tests of algorithm **FindExpert**

$$\begin{aligned}
\Pr\left\{accept|p_1 - \frac{1}{2\ln t}\right\} &< \exp\{-2m(p-p_1)(p-p_1+2k) - \ln 1/\alpha\} \\
&= \exp\{-2m(p-p_1)(p-p_1+\epsilon) - 2\ln t\} \\
&= \exp\{-2\ln^2 t\left(-\frac{1}{2\ln t}\right)\left(-\frac{1}{2\ln t} + \frac{2}{\sqrt{\ln t}}\right) - 2\ln t\} \\
&= \exp\{\ln t\left(\frac{-1+4\sqrt{\ln t}}{2\ln t}\right) - 2\ln t\} \\
&\leq e^{2\sqrt{\ln t} - 2\ln t} \\
&= \frac{t^{2/\sqrt{\ln t}}}{t^2}
\end{aligned}$$

This completes the proof.

The next step of the proof computes the number of coins we expect the algorithm to test.

Lemma 6 *The expected number, N, of coins algorithm* **FindExpert** *tests is $O(t/\ln^2 t)$*
Proof*:*

We know that the algorithm takes total time t, and we can compute the time that the algorithm takes for N coins. Estimating the initial coin takes $\ln^3 t$ time. Testing each of the N coins takes $\ln^2 t$ time, and each time we accept a coin (at most $\log N + o(1)$ times from Appendix D) it is tested $\ln^3 t$ times. Summarizing we have

$$\ln^3 t + N\ln^2 t + (\log N + o(1))\ln^3 t = t$$

Thus

$$N = O(\frac{t}{\ln^2 t}).$$

And now we are ready to prove the main theorem (Theorem 5.)

Proof of main theorem: *With probability $1 - Pr(any\ problem\ with\ the\ test\ sequence)$, after t trials we will see $O(t/\ln^2 t)$ coins and have the lowest error coin among them. Notice that the last series of tests (i.e., since the last accepted coin) may have rejected* *coins in the range $[p_0, p_1]$ which may be better than the current coin we have. Therefore, the error probability of our best coin may be as much as the gap size of the last sequential test away from the true best coin. Since we know from Lemma 2 that the gap size is $O(\epsilon)$, the error probability of the coin algorithm* **FindExpert** *outputs is*

$$b_{\frac{t}{\ln^2 t}} + O(\epsilon) = b_{\frac{t}{\ln^2 t}} + O(\sqrt{\frac{1}{\ln t}})$$

It remains then to compute the probability that there are any problems with the test sequence.

$$\begin{aligned} Pr(\textit{any problem with the test sequence}) &= Pr(\textit{ever accepting a bad coin}) \\ &+ Pr(\textit{not accepting any good coin}) \\ &+ Pr(\textit{the last gap is greater than } O(\epsilon)) \end{aligned}$$

We will compute each of these in turn. The probability of accepting one bad coin is shown to be $\frac{t^{\frac{2}{\sqrt{\ln t}}}}{t^2}$ with probability $1 - 1/t^2$ in Lemma 3, and with probability $1/t^2$ it could be as high as 1. Summing we get $Pr(\textit{accepting one bad coin}) = O(\frac{t^{\frac{2}{\sqrt{\ln t}}}}{t^2})$.

The probability of ever accepting a bad coin is, at most, the number of lead changes times the probability of accepting a bad coin,

$$Pr(\textit{ever accepting a bad coin}) \leq (\ln \frac{t}{\ln^2 t}) \frac{t^{\frac{2}{\sqrt{\ln t}}}}{t^2} = O(\frac{(\ln t) t^{\frac{2}{\sqrt{\ln t}}}}{t^2}).$$

Note that $(\ln t) t^{\frac{2}{\sqrt{\ln t}}} < (\ln t) t^{\gamma}$ for $t > e^{(\frac{2}{\gamma})^2}$. For $\gamma < 1$, $(\ln t) t^{\gamma}$ is known to be $O(t)$. Thus we can conclude that the

$$Pr(\textit{ever accepting a bad coin}) = O(\frac{1}{t}).$$

To finish the computation, it is easy to see that

$$Pr(\textit{not accepting any good coin}) \leq (\textit{Number of coins tested})\alpha$$

$$= \frac{t}{log^2 t} \cdot \frac{1}{t^2}$$
$$= O(\frac{1}{t})$$

and

$$Pr(\textit{the last gap is greater than } O(\epsilon)) = \frac{1}{t^2} = O(\frac{1}{t})$$

from Lemma 2.

To summarize, Theorem 5 shows that the algorithm **FindExpert**, which finds a low error coin from coins drawn according to an unknown distribution using the ratio test to test each coin, does almost as well as we can do knowing the error probabilities, but seeing only $t/\ln^2 t$ coins. The result of Theorem 5 is independent of the distribution of the coins. The length of the ratio test does not change for any distribution, but depending on the distribution the best error rate of $t/\ln^2 t$ randomly chosen coins is very close or very far away from the best error rate of t randomly chosen coins. For example for a jar of fair coins $b_{t/\ln^2 t} = b_t$ because no coin will be better than the first coin you pick. On the other hand, for coins distributed uniformly $b_t = 1/t$ which is much less than $b_{t/\ln^2 t} = \ln^2 t/t$.

6.3 A Faster (?) Test for Experts

A disadvantage of the ratio test in the previous section is that the length of each test is fixed. This length is chosen so as to guarantee (with high probability) a good determination as to whether the tested coin has error rate at least ϵ better than the current best coin. For coins that are much better or much worse, it may be possible to make this determination with many fewer trials.

The sequential ratio test given by Wald (1947) solves precisely this problem. After each coin toss it assesses whether it is sufficiently sure that the tested coin is better or worse than the current best coin. If not, the test continues. The sequential ratio test thus uses a *variable* number of flips to test a coin. One can hope that for the same probability of erroneous acceptances and rejections, the sequential ratio test will use fewer coin flips than the ratio test. Although the worst case sample size is larger for the sequential ratio test, Wald (1947) shows that in experiments with normally distributed error rates the sequential test is on average twice as efficient as the ratio test. Section 6.4 gives our experimental results comparing expert-finding algorithms based on the ratio test and on the sequential ratio test.

The rest of this section gives the sequential ratio test and the corresponding expert-finding algorithm.

6.3.1 The Sequential Ratio Test

This section describes the sequential ratio test due to Wald (1947). It furthermore gives the operating characteristic function and the average sample number of the test, which are important to the proof analysis of the expert-finding algorithm.

The Problem Given a coin with unknown failure rate p, and thresholds p_0, p_1 with $p_0 < p_1$. Test if $p \leq p_0$ vs. $p \geq p_1$. Accept if $p \leq p_0$. Reject if $p \geq p_1$.

Requirements The probability of rejecting a coin does not exceed α if $p \leq p_0$, and the probability of accepting a coin does not exceed β if $p \geq p_1$.

The Test Let m be the number of samples, and f_m be the number of failures in m samples.

Reject if

$$f_m \geq \frac{\log \frac{1-\beta}{\alpha}}{\log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0}} + m \frac{\log \frac{1-p_0}{1-p_1}}{\log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0}}.$$

Accept if

$$f_m \leq \frac{\log \frac{\beta}{1-\alpha}}{\log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0}} + m \frac{\log \frac{1-p_0}{1-p_1}}{\log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0}}.$$

Otherwise, draw another sample.

The sequential ratio test defines two lines with different intercepts and the same slope. The region above the upper line is a reject region. The region below the lower line is the accept region. The test generates a random walk starting at the

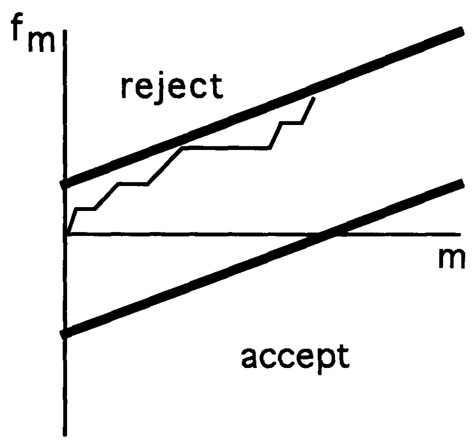


Figure 6-1: A graphical depiction of a typical sequential ratio test

origin which terminates when it reaches one of the two lines. (See Figure 6-1 for a graphical depiction of the sequential ratio test.)

Operating Characteristic Function Let the function

$L(p)$ = probability that the coin will be accepted when p is the true probability of failure.

The value of the function $L(p)$ is

$$L(p) = \frac{\left(\frac{1-\beta}{\alpha}\right)^h - 1}{\left(\frac{1-\beta}{\alpha}\right)^h - \left(\frac{\beta}{1-\alpha}\right)^h}$$

where

$$p = \frac{1 - \left(\frac{1-p_1}{1-p_0}\right)^h}{\left(\frac{p_1}{p_0}\right)^h - \left(\frac{1-p_0}{1-p_1}\right)^h}.$$

The parameter h can be any non-zero value. For any arbitrary value of h, the point $[p, L(p)]$ is a point on the Operating Characteristic function. Specific values of $L(p)$ of interest are $L(0) = 1$, $L(1) = 0$, $L(p_0) = 1 - \alpha$, and $L(p_1) = \beta$. A typical operating characteristic function looks like the graph in Figure 6-2.

Average Sample Number Let the random variable n be the number of observations required by the test procedure, and $E_p(n)$ be the expected value of n.

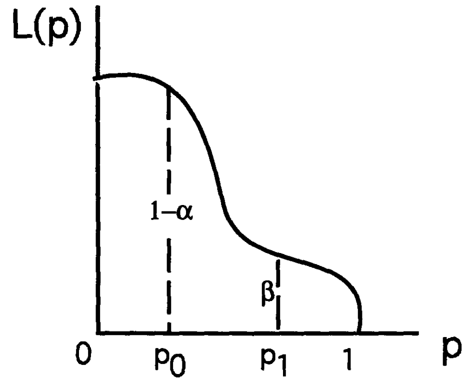


Figure 6-2: A typical operating characteristic function of the sequential ratio test

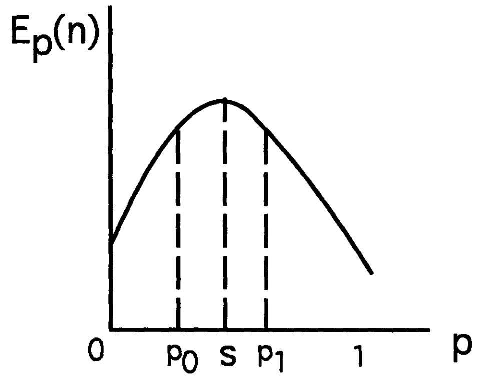


Figure 6-3: The typical shape of the average sample number of the sequential ratio test

Wald shows that

$$E_p(n) = \frac{L(p)\log\frac{\beta}{1-\alpha} + (1-L(p))\log\frac{1-\beta}{\alpha}}{p\log\frac{p_1}{p_0} + (1-p)\log\frac{1-p_1}{1-p_0}}$$

The average sample number function has the shape of the graph in Figure 6-3. Its value is largest at (or close to) the point $p = s = \frac{\log\frac{1-p_0}{1-p_1}}{\log\frac{p_1}{p_0} - \log\frac{1-p_1}{1-p_0}}$. The value of the average sample number at this point is

$$E_s(n) = \frac{-\log\frac{\beta}{1-\alpha}\log\frac{1-\beta}{\alpha}}{\log\frac{p_1}{p_0}\log\frac{1-p_0}{1-p_1}}.$$

6.3.2 Finding a Good Expert Using the Sequential Ratio Test

The algorithm for finding a good coin using the sequential ratio test is as follows.

Algorithm 18 SeqFindExpert:
Input t, an upper bound on the number of trials allowed.

```
Let BestCoin = Draw a coin.
Flip BestCoin $\ln^3 t$ times to find $\widehat{p}$.
Set $p_1 = \widehat{p}$.
Repeat until all $t$ trials are used:
    Let $p_0 = p_1 - \epsilon(p_1)$, where $\epsilon(p_1) = \sqrt{\frac{4p_1(1-p_1)}{\log t}}$.
    Let Coin = Draw a coin.
    Test Coin using the sequential ratio test
        with parameters $p_0$, $p_1$, and $\alpha = \beta = 1/t^2$.
    If the sequential test accepts then
        Set BestCoin = Coin.
        Flip BestCoin $\log^3 t$ more times to find an improved $\widehat{p}$.
        Set $p_1 = \widehat{p}$.
Output BestCoin.
```

6.3.3 Efficiency of Algorithm SeqFindExpert

Because the worst case number of coin flips for the sequential ratio test is larger than the (fixed) number of coin flips for the ratio test, the bound in Theorem 6 for **SeqFindExpert** ratio test is not as strong as the bound shown above for **FindExpert**.

Theorem 6 *There is an algorithm (***SeqFindExpert***) such that when the coins are drawn according to an unknown error-rate distribution, after t trials, with probability at least $1 - 1/t$, we expect to find a coin whose probability of error is at most $b_{t/\log^3 t} + O(\frac{1}{\sqrt{\log t}})$.*

This theorem states that after t trials, we expect the algorithm to find an expert that is almost as good as the best expert in a set of $t/\log^3 t$ randomly drawn experts ($b_{t/\log^3 t}$). Like Theorem 5 this result is independent of the distribution. Since the result is based on the worse case sample size the tests may be shorter and thus the algorithm may examine more coins. The rest of this section proves Theorem 6. The proof is very similar to the proof of Theorem 5.

Lemma 7 shows that the expected length of each test is short ($\log^3 t$ trials at most).

Lemma 7 *The expected length of each sequential ratio test with parameters* p_1, $p_0 = p_1 - \epsilon$, $\alpha = \beta = 1/t^2$ *is at most* $\log^3 t$.

Proof: *We compute the value of the average sample number with the given parameters for the point* $p = s$ *where the average sample number is largest.*

$$\begin{aligned} E_s(n) &= \frac{-\log\frac{\beta}{1-\alpha}\log\frac{1-\beta}{\alpha}}{\log\frac{p_1}{p_0}\log\frac{1-p_0}{1-p_1}} \\ &\approx \frac{4\log^2 t}{\log\frac{p_1}{p_1-\epsilon}\log\frac{1-p_1+\epsilon}{1-p_1}} \\ &\approx \frac{4\log^2 t}{\frac{\epsilon}{p_1}\frac{\epsilon}{1-p_1}} \\ &= \frac{4(\log^2 t)p_1(1-p_1)}{\frac{4p_1(1-p_1)}{\log t}} \\ &= \log^3 t. \end{aligned}$$

Again we must consider what the effects of estimating the error probability of the best coins are. Lemma 2, which shows that *with probability* $1 - 1/t^2$, *estimating the error probability of a coin with* $\log^3 t$ *coin tosses gives a testing gap of size* $O(\epsilon) = O(1/\sqrt{\log t})$, holds for **SeqFindExpert** since its proof relies only on the sample size used to estimate the error probability of the coin.

Recall that when the true error probability of the current best coin (p) is smaller than the estimated error probability, i.e., $p \in [p_0, p_1]$, we have some chance of accepting coins with error probability in the range $(p, p_1])$. These coins are worse than the current best. We must show that this probability is small for the sequential ratio test using the operating characteristic function.

Lemma 8 *With probability* $1 - 1/t^2$, *estimating the error probability of a coin with* $\log^3 t$ *coin tosses gives the sequential test an* $O(\frac{t^{\frac{2}{\sqrt{\log t}}}}{t^2})$ *probability of accepting a coin* *with error probability p_1.*

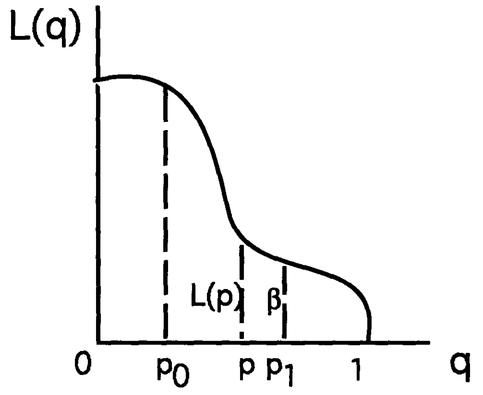
Proof: *If the estimate $\widehat{p} \leq p$ then it is clear that the probability of accepting a coin with error probability greater than p_1 (which is equal to $\widehat{p}$ and thus is less than p) is at most $\beta = 1/t^2$ since the operating characteristic function is monotonically decreasing with increasing p (see the figure below).*

If $p < \widehat{p}$ then p is in the range $[p_0, p_1]$ since $p_1 = \widehat{p} > p$. Thus there is a higher probability of accepting coins with error probability less than p, since the operating characteristic function is monotonically increasing with decreasing p.

Recall the operating characteristic function for the sequential ratio test is

$$L(q) = \textit{the probability of accepting a coin with probability } q.$$

The probability of accepting any coin in the range $[p, p_1]$ is at most $L(p)$ (as the following graph demonstrates.)



Appendix B shows that with probability $1 - 1/t^2$, $\widehat{p}$ is within $O(\sqrt{\log t})$ standard deviations of p. So we want to compute $L(p_1 - \sqrt{\log t}\,\sigma_{\widehat{p}})$.

Recall that the operating characteristic function is

$$L(q) = \frac{\left(\frac{1-\beta}{\alpha}\right)^h - 1}{\left(\frac{1-\beta}{\alpha}\right)^h - \left(\frac{\beta}{1-\alpha}\right)^h}$$

where

$$q = \frac{1 - \left(\frac{1-p_1}{1-p_0}\right)^h}{\left(\frac{p_1}{p_0}\right)^h - \left(\frac{1-p_0}{1-p_1}\right)^h}$$

and h is any non-zero value.

For the parameter settings in the sequential tests of algorithm **SeqFindExpert** *we find that*

$$L(q) = \frac{(t^2-1)^h - 1}{(t^2-1)^h - (t^2-1)^{-h}}$$

and from Appendix C we know that

$$q \approx p_1 - \frac{h+1}{2}\epsilon.$$

We can now compute the value of the operating characteristic function at the point $q = p_1 - \sqrt{\log t}\sigma_{\hat{p}}$. First we find the value of h at this point. We know that $\frac{h+1}{2}\epsilon \approx \frac{1}{2\log t}$. Solving for h we get

$$h = \frac{1}{\epsilon \log t} - 1 = -1 + \delta$$

where $\delta \leq \frac{1}{\sqrt{\log t}}$.

And to finish the proof of the claim

$$\begin{aligned} L(p_1 - \sqrt{\log t}\sigma_{\hat{p}}) &\approx \frac{t^{2(-1+\delta)} - 1}{t^{2(-1+\delta)} - t^{-2(-1+\delta)}} \\ &\approx \frac{t^{2(1-\delta)}}{t^{4(1-\delta)}} \\ &= \frac{t^{2\delta}}{t^2} \\ &= \frac{t^{\frac{2}{\sqrt{\log t}}}}{t^2}. \end{aligned}$$

Now we can compute the number of coins we expect the algorithm to test.

Lemma 9 *The expected number, N, of coins algorithm* **SeqFindExpert** *tests is $O(t/\log^3 t)$*

Proof*:*

We know that the algorithm takes total time t, and we can compute the time that the algorithm takes for N coins. Estimating the initial coin takes $\log^3 t$ *time. Each time we accept a coin (at most* $\log N + o(1)$ *times from Appendix D), it is tested* $\log^3 t$ *times. And each sequential test takes* $\log^3 t$ *expected time in the worst case. Summarizing we have*

$$\log^3 t + N\log^3 t + (\log N + o(1))\log^3 t = t$$

Thus

$$N = O(\frac{t}{\log^3 t}).$$

Finally the proof of Theorem 6 is virtually identical to the proof of Theorem 5.

Proof of Theorem 6: *With probability* $1 - Pr(\textit{any problem with the test sequence})$, *after t trials we will see* $O(t/\log^3 t)$ *coins and have the lowest error coin among them. Notice that the last series of sequential tests (i.e., since the last accepted coin) may have rejected coins in the range* $[p_0, p_1]$ *which may be better than the current coin we have. Therefore, the error probability of our best coin may be as much as the gap size of the last sequential test away from the true best coin. Since we know from Lemma 2 that the gap size is* $O(\epsilon)$, *the error probability of the coin algorithm* **SeqFindExpert** *outputs is*

$$b_{\frac{t}{\log^3 t}} + O(\epsilon) = b_{\frac{t}{\log^3 t}} + O(\sqrt{\frac{1}{\log t}})$$

It remains then to compute the probability that there are any problems with the test sequence.

$$\begin{aligned} Pr(\textit{any problem with the test sequence}) &= Pr(\textit{ever accepting a bad coin}) \\ &+ Pr(\textit{not accepting any good coin}) \\ &+ Pr(\textit{the last gap is greater than } O(\epsilon)) \end{aligned}$$

$\Pr\{\textit{not accepting any good coin}\}$ *and* $\Pr\{\textit{the last gap is greater than } O(\epsilon)\}$ *are shown to be* $O(\frac{1}{t})$ *in the proof of Theorem 5.*

The probability of accepting one bad coin is $O(\frac{t^{\frac{2}{\sqrt{\log t}}}}{t^2})$*, and the probability of ever accepting a bad coin is*

$$Pr(\textit{ever accepting a bad coin}) \leq (\log \frac{t}{\log^3 t})\frac{t^{\frac{2}{\sqrt{\log t}}}}{t^2} = O(\frac{(\log t)t^{\frac{2}{\sqrt{\log t}}}}{t^2}) = O(\frac{1}{t}).$$

Theorem 6 shows that algorithm **SeqFindExpert**, which uses the sequential ratio test to find a low error-rate coin from coins drawn according to an unknown distribution, does almost as well as it could do if coins were labeled with their error rates, but it sees only $t/\log^3 t$ coins. The proof of Theorem 6 is similar to the proof of Theorem 5. The bound in Theorem 6 is not as tight as the bound for the **FindExpert**. In practice, however, **SeqFindExpert** often performs better because the test lengths are much shorter than the worst case test length used to prove Theorem 6.

For some distributions, such as the uniform distribution, the coins tested are typically *much* worse than the current best. (After seeing a few coins the algorithm already has a fairly good coin and most coins are much worse.) Thus, the sequential ratio tests will be short. When the error rates are uniformly distributed we expect that the algorithm **SeqFindExpert** will see more coins and find a better coin than **FindExpert**. This argument is confirmed by our empirical results below. Our results also show the superiority of **SeqFindExpert** when the error rates are drawn from a (truncated) normal distribution.

6.4 Empirical Comparison of FindExpert and SeqFindExpert

To compare the performance of **FindExpert** and **SeqFindExpert** we ran experiments for uniform and normally distributed error rates. (The normal distribution has mean 0.5, standard deviation 0.09, and was truncated to lie within the interval $[0, 1]$.) Table 6.1 gives results for both algorithms on the uniform distribution. All results

	Coins Tested	**Test Length**	**Best Estimated Error Rate**	**Best Actual Error Rate**
FindExpert	7.6	49	.1085	.1088
SeqFindExpert	21	44	.0986	.0979

(a) Uniform distribution; limit of $t = 1000$ trials.

	Coins Tested	**Test Length**	**Best Estimated Error Rate**	**Best Actual Error Rate**
FindExpert	66	100	.0185	.0187
SeqFindExpert	230	33	.01	.0101

(b) Uniform distribution; limit of $t = 10000$ trials.

Table 6.1: Empirical comparison of **FindExpert** and **SeqFindExpert** with the uniform distribution. The numbers in the tables are averaged over 1000 runs.

reported are an average over 1000 repeated executions of the algorithm. Table 6.1(a) contains the average of 1000 runs each with trial limit $t = 1000$. Table 6.1(a) shows that the **SeqFindExpert** algorithm had shorter average test lengths and therefore tested more experts. **SeqFindExpert** was able to find experts with lower actual error rate (.0979 on the average compared with .1088 for **FindExpert**). The table contains both the average actual error rate of the best experts that the algorithm found and the average error rate from experiments for the same experts. Table 6.1(b) shows that given more time ($t = 10000$ trials) to find a good expert **SeqFindExpert** performs significantly better than **FindExpert**. The average test length is much shorter and the resulting best error rate is .0101 compared with .0187.

Experiments with the normal distribution used a normal with mean 0.5 and standard deviation 0.09. These results are reported in Table 6.2. Note that for this distribution most coins have error rate close to .5. Table 6.2(a) reports the average of 1000 executions with trial limit 1000. As expected, the average error probabilities of the best coin is lower for the **SeqFindExpert** algorithm. Table 6.2(b) shows that with a longer limit of 10000 trials the **SeqFindExpert** algorithm performs much better than **FindExpert**, giving an average best error rate of .2741 compared with .3352.

It is interesting that with a time limit of 1000 trials the **SeqFindExpert** both

	Coins Tested	**Test Length**	**Best Estimated Error Rate**	**Best Actual Error Rate**
FindExpert	13	49	.4361	.4395
SeqFindExpert	29	61	.4144	.4204

(a) Normal distribution; limit of $t = 1000$ trials.

	Coins Tested	**Test Length**	**Best Estimated Error Rate**	**Best Actual Error Rate**
FindExpert	85	100	.3292	.3352
SeqFindExpert	470	31	.2670	.2741

(b) Normal distribution; limit of $t = 10000$ trials.

Table 6.2: Empirical comparison of **FindExpert** and **SeqFindExpert** with the normal distribution (mean 0.5, standard deviation 0.09) truncated at 0 and 1. The numbers in the tables are averaged over 1000 runs.

tested more experts and had a longer average test length. The long average test is due to a few very long tests (to compare close experts). For example, it is possible for one test in one of the 1000 runs in Table 6.2(a) to have taken the complete 1000 trials allocated to that run — remember that we expect the length of the tests to be $\log^3 t$, but this expectation does not prohibit a long test. We find this problem with a normal distribution that is tightly distributed about the mean and with short run of 1000 trials. When the runs are longer (10000 trials) we find — as we expect — that more coins are tested by **SeqFindExpert** with a shorter average test length. The reason long tests are more rare with longer runs is that after some time the best coin is much better than the mean where the density of coins is higher, so most coins drawn are much worse than the current best with a high trial limit (but not with a low one).

It is also of interest to compare the performance of **SeqFindExpert** with that of **FindExpert** on a distribution where all the coins are fair. In this distribution both algorithms will find a best coin that has error probability .5. The question of interest is how many coins each algorithm examines. Table 6.3 shows results of running **FindExpert** and **SeqFindExpert** on a distribution of fair coins. We can

	Coins Tested	Test Length	Best Estimated Error Rate	Best Actual Error Rate
FindExpert	14	49	.4963	.5
SeqFindExpert	20	35	.4991	.5

(a) All coins are fair; limit of $t = 1000$ trials.

	Coins Tested	Test Length	Best Estimated Error Rate	Best Actual Error Rate
FindExpert	91	100	.4989	.5
SeqFindExpert	120	74	.4988	.5

(b) All coins are fair; limit of $t = 10000$ trials.

Table 6.3: Empirical Comparison of **FindExpert** and **SeqFindExpert** with the distribution where all the coins are fair. The numbers in the tables are averaged over 1000 runs.

see that **SeqFindExpert** examines a few more coins than **FindExpert** examines. The difference in the number of coins tested is not large, especially compared with the differences with other distributions. This result supports the interpretation that significant improvement using **SeqFindExpert** occurs when most coins tried are much worse than the best coin found.

The experimental results in this section show that **SeqFindExpert** performs better than **FindExpert** for distributions with different characteristics. The experimental results agree with the theoretical analysis in that some sequential tests are quite long (longer than the ratio tests), but the experiments also show that on the average the sequential test lengths are short especially when the trial limit is large. The average test length is short when the time limit is large because the best expert is already much better than the average population.

6.5 Conclusions

This chapter presented two algorithms to find a low error expert from a sequence of experts with unknown error-rate distribution, a problem that arises in many areas, such as the given example of learning a world model consisting of good rules. The two algorithms **FindExpert** and **SeqFindExpert** are nearly identical, but use the ratio test and sequential ratio test respectively to determine if an expert is good.

Theorem 5 shows that **FindExpert** finds an expert which is the best expert of $t/\ln^2 t$ experts, given trial limit t. This result is strong in the sense that it shows only a factor of $\ln^2 t$ loss from testing over the best expert we could find in t trials if we knew the exact error rate of each expert. Theorem 6 gives a weaker bound for **SeqFindExpert**. Empirical results in section 6.4, on the other hand, indicate that **SeqFindExpert** performs better than **FindExpert** in practice (at least for the uniform and normal distributions).

The obvious open question from this work is to prove that **SeqFindExpert** expects to find a lower error-rate expert for general or specific distributions than **FindExpert**.

Chapter 7

Conclusion

This thesis explored principles of efficient learning in environments with manifest causal structure. Algorithms to learn a rule-based world model and high-level concepts in environments with manifest causal structure were presented.

The rule-learning algorithm from chapter 3 excels at finding correlations in the environment. It aims toward simplicity using straightforward algorithms that rely heavily on perceptions and represent learned knowledge with the simplest possible format. Several environment independent heuristics are employed in the process of creating rules, such as observing the value of the affected relations in the previous state and using mysteries to replay unexplained effects. The process of evaluating and removing rules is based on sound statistical techniques which are important to proving the convergence of the rule-learning algorithm to a good predictive model in environments with manifest causal structure. Convergence does not guarantee that the world model is perfect after any finite time. Empirical results in the Macintosh environment, however, show that the learned world model is useful in a short amount of time.

The main drawback of the world model learned by the rule-learning algorithm is the large number of rules in the model. The abundance of rules is due in part to the learning algorithm, which makes many rules, and in part to the representation of the world model and perceptions. Since the rule-learning algorithm saves every valid rule, the resulting world model contains some redundant rules.

This thesis also presented algorithms for learning high-level concepts. The concept-learning algorithms are interesting both philosophically — to show that the concepts are learnable — and practically — to reduce redundancy in the world model. Two types of concept-learning algorithms were developed in this research. The first concept-learning algorithm uses NOACTION rules to find and collapse correlated perceptions which are interpreted as new relations and new objects. The second type of concept learning includes creating generalizations of the specific learned rules. In the Macintosh Environment these concept-learning algorithms learn concepts, such as the concept of an active window and the general rule that a click in a window causes it to be active. Both concept-learning algorithms are imperfect when the learned world model is incomplete or incorrect. They may develop incorrect concepts or miss an important concept. Since the rule-learning algorithm cannot guarantee perfect knowledge, an important direction for future research is to make the concept-learning algorithms robust to missing or incorrect rules.

The empirical effectiveness of the learning algorithm in this thesis, as well as the theoretical convergence result, shows that in environments with manifest causal structure world models are efficiently learnable. This research indicates that it pays to concentrate on learning "easy" aspects of the environment first. The difficult or hidden aspects of the environment can be learned as a next step.

Learning the manifest aspects of the environment first identifies the difficult aspects of the environment. After the agent has learned a world model it knows which aspects of the environment it does not understand because it has no rules to explain those aspects of the environment. Knowing what it does not know may be as important as knowing what it does know because the aspects of the environment that it does not understand are probably the difficult, hidden, or non-deterministic aspects of the environment.

The algorithms in this thesis learn a working world model of the Macintosh environment. The world model predicts well and contains valid rules about the Macintosh environment. Many of the rules in the world model are general and describe important concepts, but the number of rules in the world model remains large and includes many specific rules. A worthy goal for future research is to reduce the number of rules to a small set of general rules that correspond to the complete model people use for the Macintosh. Another direction for future work is to include additional aspects of the Macintosh operating system and applications in the environment.

Appendix A

More General Rules in the Macintosh Environment

This appendix presents additional rules that the rule generalization algorithm learned in the Macintosh environment. For each general rule an English description is given as well as the specific rules that led to its creation.

1. `A click in a close-box makes the grow-box of the same window disappear`

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 1}\ \texttt{CB} \rightarrow EXIST(\textit{Window 1}\ \texttt{GB}) = NP$$

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 2}\ \texttt{CB} \rightarrow EXIST(\textit{Window 2}\ \texttt{GB}) = NP$$

$$\begin{gathered} TYPE(y) = GB \wedge OV(y,x) = F \wedge TYPE(x) = CB \wedge X(x,y) = 1122 \\ \wedge Y(x,y) = 1122 \wedge OV(x,y) = F \\ \rightarrow \texttt{click-in}\ x \rightarrow EXIST(y) = NP \end{gathered}$$

2. `A click in a close-box makes the zoom-box of the same window disappear`

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 1}\ \texttt{CB} \rightarrow EXIST(\textit{Window 1}\ \texttt{ZB}) = NP$$

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 2}\ \texttt{CB} \rightarrow EXIST(\textit{Window 2}\ \texttt{ZB}) = NP$$

$$TYPE(y) = ZB \wedge OV(y,x) = F \wedge TYPE(x) = CB \wedge X(x,y) = 1122$$
$$\wedge Y(x,y) = 33 \wedge OV(x,y) = F$$
$$\rightarrow \texttt{click-in } x \rightarrow EXIST(y) = NP$$

3. A click in a close-box makes the active-title-bar of the same window disappear

$$NIL \rightarrow \texttt{click-in } \textit{Window 1 } \texttt{CB} \rightarrow EXIST(\textit{Window 1 } \texttt{ATB}) = NP$$

$$NIL \rightarrow \texttt{click-in } \textit{Window 2 } \texttt{CB} \rightarrow EXIST(\textit{Window 2 } \texttt{ATB}) = NP$$

$$TYPE(y) = ATB \wedge TYPE(x) = CB \wedge X(x,y) = 2112 \wedge Y(y,x) = 1122$$
$$\wedge OV(x,y) = T$$
$$\rightarrow \texttt{click-in } x \rightarrow EXIST(x) = NP$$

4. A click in a title bar makes that title-bar disappear.

$$NIL \rightarrow \texttt{click-in } \textit{Window 1 } \texttt{TB} \rightarrow EXIST(\textit{Window 1 } \texttt{TB}) = NP$$

$$NIL \rightarrow \texttt{click-in } \textit{Window 2 } \texttt{TB} \rightarrow EXIST(\textit{Window 2 } \texttt{TB}) = NP$$

$$TYPE(x) = TB \rightarrow \texttt{click-in } x \rightarrow EXIST(x) = NP$$

5. A click in a window rectangle make the corresponding title bar disappear.

$$NIL \rightarrow \texttt{click-in } \textit{Window 1} \rightarrow EXIST(\textit{Window 1 } \texttt{TB}) = NP$$

$$NIL \rightarrow \texttt{click-in } \textit{Window 2} \rightarrow EXIST(\textit{Window 2 } \texttt{TB}) = NP$$

$$TYPE(y) = TB \wedge OV(y,x) = T \wedge TYPE(x) = REC \wedge X(x,y) = 33$$
$$\wedge Y(x,y) = 2211$$
$$\rightarrow \texttt{click-in } x \rightarrow EXIST(y) = NP$$

6. A click in a window interior makes the corresponding title-bar disappear.

$$NIL \rightarrow \texttt{click-in } \textit{Window 1} \texttt{ INTERIOR} \rightarrow EXIST(\textit{Window 1} \texttt{ TB}) = NP$$

$$NIL \rightarrow \texttt{click-in } \textit{Window 2} \texttt{ INTERIOR} \rightarrow EXIST(\textit{Window 2} \texttt{ TB}) = NP$$

$$\begin{gathered} TYPE(y) = TB \wedge OV(y,x) = F \wedge TYPE(x) = REC \wedge X(x,y) = 33 \\ \wedge Y(x,y) = 2211 \wedge OV(x,y) = F \\ \rightarrow \texttt{click-in } x \rightarrow EXIST(y) = NP \end{gathered}$$

7. A click in a zoom-box makes a rectangle that is part of another window disappear.

$$\begin{gathered} EXIST(\texttt{NEW-WINDOW2}) = T \rightarrow \texttt{click-in } \textit{Window 1} \texttt{ ZB} \rightarrow \\ EXIST(\textit{Window 2}) = NP \end{gathered}$$

$$\begin{gathered} EXIST(\texttt{NEW-WINDOW2}) = T \rightarrow \texttt{click-in } \textit{Window 1} \texttt{ ZB} \rightarrow \\ EXIST(\textit{Window 2} \texttt{ INTERIOR}) = NP \end{gathered}$$

$$\begin{gathered} EXIST(z) = T \wedge TYPE(y) = REC \wedge OV(yx) = F \wedge PART\text{-}OF(y,z) = T \\ \wedge TYPE(x) = ZB \wedge X(x,y) = 2211 \wedge Y(x,y) = 1122 \wedge OV(x,y) = F \\ \rightarrow \texttt{click-in } x \rightarrow EXIST(y) = NP \end{gathered}$$

8. A click in a rectangle makes the close-box of another window disappear.

$$NIL \rightarrow \texttt{click-in } \textit{Window 1} \texttt{ INTERIOR} \rightarrow EXIST(\textit{Window 2} \texttt{ CB}) = NP$$

$$NIL \rightarrow \texttt{click-in } \textit{Window 1} \rightarrow EXIST(\textit{Window 2} \texttt{ CB}) = NP$$

$$\begin{gathered} TYPE(y) = CB \wedge X(y,x) = 1122 \wedge Y(y,x) = 2211 \wedge OV(y,x) = F \\ \wedge TYPE(x) = REC \\ \rightarrow \texttt{click-in } x \rightarrow EXIST(y) = NP \end{gathered}$$

9. a click in a rectangle makes the zoom-box of another window disappear.

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 1}\ \texttt{INTERIOR} \rightarrow EXIST(\textit{Window 2}\ \texttt{ZB}) = NP$$

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 1} \rightarrow EXIST(\textit{Window 2}\ \texttt{ZB}) = NP$$

$$TYPE(y) = ZB \wedge X(y,x) = 2112 \wedge Y(y,x) = 2211 \wedge OV(y,x) = T$$
$$\wedge TYPE(x) = REC$$
$$\rightarrow \texttt{click-in}\ x \rightarrow EXIST(y) = NP$$

10. A click in a rectangle makes any button-dialog-item that overlaps that rectangles present.

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 1} \rightarrow$$
$$EXIST(\textit{Window 1}\ \texttt{BUTTON-DIALOG-ITEM Window 2}) = T$$

$$NIL \rightarrow \texttt{click-in}\ \textit{Background} \rightarrow$$
$$EXIST(\textit{Window 1}\ \texttt{BUTTON-DIALOG-ITEM Window 2}) = T$$

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 1}\ \texttt{INTERIOR} \rightarrow$$
$$EXIST(\textit{Window 1}\ \texttt{BUTTON-DIALOG-ITEM Window 2}) = T$$

$$TYPE(y) = \textit{BUTTON-DIALOG-ITEM} \wedge OV(y,x) = T$$
$$\wedge TYPE(x) = REC \wedge X(x,y) = 1221 \wedge Y(x,y) = 2211$$
$$\rightarrow \texttt{click-in}\ x \rightarrow EXIST(y) = T$$

11. The active-title-bar of a window does not overlap its interior after a click in a window's title-bar.

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 1}\ \texttt{TB} \rightarrow$$
$$OV(\textit{Window 1}\ \texttt{ATB},\ \textit{Window 1}\ \texttt{INTERIOR}) = F$$

$$NIL \rightarrow \texttt{click-in}\ \textit{Window 2}\ \texttt{TB} \rightarrow$$
$$OV(\textit{Window 2}\ \texttt{ATB},\ \textit{Window 2}\ \texttt{INTERIOR}) = F$$

$$
\begin{gathered}
TYPE(z) = REC \wedge OV(z, x) = F \wedge TYPE(y) = ATB \wedge X(y, z) = 33 \\
\wedge Y(y, z) = 132 \wedge OV(y, z) = F \wedge TYPE(x) = TB \wedge X(x, z) = 33 \\
\wedge Y(xz) = 132 \wedge OV(xz) = F \wedge OV(z, y) = F \\
\rightarrow \texttt{click-in}\ x \ \rightarrow \ OV(y, z) = F
\end{gathered}
$$

Appendix B

The Distance of $\widehat{p}$ from p

Claim 1 *With probability $1-1/t^2$, $\widehat{p}$ is not farther than $O(\sqrt{\ln t})$ standard deviations from p.*

Proof:

The random variable $\widehat{p}$ has standard deviation $\sigma^2_{\widehat{p}} = \sqrt{\frac{p(1-p)}{\log^3 t}}$. We want to find the value, q, such that the probability that $\widehat{p} \geq q$ is very small, say $1/t^2$.

Let q be some number of standard deviations from the true error probability p, i.e. $q = p - c\sigma_{\widehat{p}}$. We want to find c such that

$$Pr(\widehat{p} \geq p - c\sigma_{\widehat{p}}) \leq 1/t^2.$$

Let $\widehat{p} = S/n$, $n = \log^3 t$. We can then rewrite the left-hand side of the above equation as

$$Pr(S \geq \left(1 + c\sqrt{\frac{1-p}{p\log^3 t}}\right) pn).$$

Now using Chernoff bounds and simplifying we find that

$$Pr(\widehat{p} \geq p - c\sigma_{\widehat{p}}) \leq e^{c^2/3}.$$

We want

$$e^{c^2/3} = 1/t^2 \qquad \text{(B.1)}$$

$$c^2 = O(\ln t)$$

Thus with probability $1 - 1/t^2$, $\widehat{p}$ *is not farther than* $O(\sqrt{\ln t})$ *standard deviations from* p.

Appendix C

A Closed Form Estimate For the Operating Characteristic Function

Claim 2 *For small ϵ and $p_0 = p_1 - \epsilon$,*

$$p = \frac{1 - \left(\frac{1-p_1}{1-p_0}\right)^h}{\left(\frac{p_1}{p_0}\right)^h - \left(\frac{1-p_0}{1-p_1}\right)^h}$$

is approximated by

$$p = p_1 - \frac{h+1}{2}\epsilon.$$

Proof:

$$\begin{aligned} p &= \frac{1 - \left(\frac{1-p_1}{1-p_1+\epsilon}\right)^h}{\left(\frac{p_1}{p_1-\epsilon}\right)^h - \left(\frac{1-p_1}{1-p_1+\epsilon}\right)^h} \\ &= \frac{1 - \left(1 + \frac{\epsilon}{1-p_0}\right)^h}{\left(1 - \frac{\epsilon}{p_1}\right)^h - \left(1 + \frac{\epsilon}{1-p_0}\right)^h} \end{aligned}$$

Using the order 2 Taylor Polynomial approximation

$$(1-x)^h \approx 1 - hx + \frac{h(h-1)}{2}x^2$$

$$
\begin{aligned}
&\approx \frac{1-\left(1-\frac{h\epsilon}{1-p_1}+\frac{-h(-h-1)\epsilon^2}{2(1-p_1)^2}\right)}{\left(1+\frac{h\epsilon}{p_1}+\frac{-h(-h-1)\epsilon^2}{2p_1^2}\right)-\left(1-\frac{h\epsilon}{1-p_1}+\frac{-h(-h-1)\epsilon^2}{2(1-p_1)^2}\right)} \\
&= \frac{\frac{1}{1-p_1}+\frac{(-h-1)\epsilon}{2(1-p_1)^2}}{\frac{1}{p_1}-\frac{(-h-1)\epsilon}{2p_1^2}+\frac{1}{1-p_1}+\frac{(-h-1)\epsilon}{2(1-p_1)^2}} \\
&= \frac{p_1\left(1-\frac{(h+1)\epsilon}{2(1-p_1)}\right)}{1+\frac{(h+1)\epsilon(1-2p_1)}{2p_1(1-p_1)}} \\
&\approx p_1\left(1-\frac{(h+1)\epsilon}{2(1-p_1)}\right)\left(1-\frac{(h+1)\epsilon(1-2p_1)}{2p_1(1-p_1)}\right) \\
&\approx p_1\left(1-\frac{(h+1)\epsilon}{2(1-p_1)}-\frac{(h+1)\epsilon(1-2p_1)}{2p_1(1-p_1)}\right) \\
&= p_1-\frac{h+1}{2}\epsilon
\end{aligned}
$$

Appendix D

The Expected Number of Coins Accepted by Algorithm SeqFindExpert

Lemma 10 *The expected number of coins accepted by algorithm* **SeqFindExpert** *is* $\log N + o(1)$ *where* N *is the number of coins tested.*

Proof*: We want to compute* $F(i)$ = *the expected number of coins accepted from* i *coins*

We will show, by induction, that $F(i) \leq \log i + \frac{1}{(1-\alpha-\beta')^i} - 1$

Base Case: $F(1) = Pr(\textit{no mistakes}) \cdot \log 1 + Pr(\textit{mistake}) \cdot 1 = \alpha + \beta'$

We can verify that $F(1) \leq \log 1 + \frac{1}{1-\alpha-\beta'} - 1 \leq \alpha + \beta'$

Induction Step: *We know that to find the smallest of* N *numbers, the expected number of times we change the current minimum is* $\log N$ *(see (Knuth 1968).)*

Algorithm **SeqFindExpert** *will do worse by either accepting a worse coin than the current minimum (with probability* $\beta' = \frac{t^{\frac{2}{\sqrt{\log t}}}}{t^2}$ *or not accepting a lower coin than the current minimum (with probability* α*.) If this mistake happens at trial* i*, it will lead to at most* $F(N-i)$ *lead changes.*

The expected total number of lead changes is

$$F(N) \quad \leq \quad Pr(\textit{no mistakes}) \cdot \log N + \sum_{i=1}^{N} Pr(\textit{1st mistake at time } i) \cdot [F(N-i) + 1]$$

$$= (1-\alpha-\beta')^N \log N + \sum_{i=1}^{N}(1-\alpha-\beta')^{i-1}(\alpha-\beta')[F(N-i)+1]$$
$$= (1-\alpha-\beta')^N \log N + (1-(1-\alpha-\beta')^N) + (\alpha-\beta')\sum_{i=1}^{N}(1-\alpha-\beta')^{i-1}F(N-i)$$

Substituting $F(i) \leq \log i + \frac{1}{(1-\alpha-\beta')^N} - 1$ *for* $i < N$

$$\begin{aligned}
F(N) &\leq (1-\alpha-\beta')^N \log N + (1-(1-\alpha-\beta')^N) \\
&\quad +(\alpha-\beta')\sum_{i=1}^{N}(1-\alpha-\beta')^{i-1}(\log(N-i) + \frac{1}{(1-\alpha-\beta')^{N-i}} - 1) \\
&\leq (1-\alpha-\beta')^N \log N + (1-(1-\alpha-\beta')^N) \\
&\quad +(\alpha-\beta')(\log N + \frac{1}{(1-\alpha-\beta')^N} - 1)\sum_{i=1}^{N}(1-\alpha-\beta')^{i-1} \\
&= (1-\alpha-\beta')^N \log N + (1-(1-\alpha-\beta')^N) \\
&\quad +(\log N + \frac{1}{(1-\alpha-\beta')^N} - 1)(1-(1-\alpha-\beta')^N) \\
&= \log N + (1-(1-\alpha-\beta')^N)\frac{1}{(1-\alpha-\beta')^N} \\
&= \log N + \frac{1}{(1-\alpha-\beta')^N} - 1
\end{aligned}$$

This completes the inductive proof.

Now we know that $F(N) \leq \log N + \frac{1}{(1-\alpha-\beta')^N} - 1$. *Furthermore,* $\frac{1}{(1-\alpha-\beta')^N} - 1 \approx N(\alpha+\beta')$. *Lemma 4 shows that* $N = O(\frac{t}{\log^2 t})$ *which gives a number of lead changes*

$$F(N) \leq \log N + o(1)$$

for α *and* β' *in* $o(1/t)$.

Bibliography

Allen, J. F. (1984), 'Towards a general theory of action and time', *Artificial Intelligence* **23**, 123–154.

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

Baase, S. (1988), *Computer Algorithms: Introduction to Design and Analysis*, Addison–Wesley, California.

Baker, A. B. & Ginsberg, M. L. (1989), Temporal Reasoning and Narrative Conventions, *in* 'Proceedings, IJCAI-89', Cambridge, MA, pp. 15–21.

Berwick, R. C., ed. (1985), *The Acquisition of Syntactic Knowledge*, The MIT Press, Cambridge, MA.

Booker, L. B. (1988), 'Classifier Systems that Learn Internal World Models', *Machine Learning* **3**, 161–192.

Dean, T., Angluin, D., Basye, K., Engelson, S., Kaelbling, L. P., Kokkevis, E. & Maron, O. (1992), Inferring Finite Automata with Stochastic Output Functions and an Application to Map Learning, *in* 'Proceedings, AAAI-92', San Jose, CA, pp. 208–214.

Dean, T., Kaelbling, L. P., Kirman, J. & Nicholson, A. (1993), Planning With Deadlines in Stochastic Domains, *in* 'Proceedings, AAAI-93', Washington, DC, pp. 574–579.

Drescher, G. L. (1989), Made-Up Minds: A Constructivist Approach to Artificial Intelligence, PhD thesis, MIT.

Drescher, G. L. (1991), *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*, The MIT Press, Cambridge, MA.

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

Georgeff, M. P. & Lansky, A. L. (1987), Reactive Reasoning and Planning, *in* 'Proceedings of Robotics and Automation', Seattle, WA, pp. 677–682.

Grimmett, G. R. & Stirzaker, D. R. (1982), *Probability and random processes*, Oxford University Press, New York.

Hanks, S. & McDermott, D. V. (1985), Temporal Reasonsing and Default Logics, Technical Report YALEU/CSD/RR 430, Department of Computer Science, Yale University.

Hanks, S. & McDermott, D. V. (1987), 'Nonmonotonic Logic and Temporal Projection', *Artificial Intelligence* **33**, 379–412.

Holland, J. H. (1976), Adaptation, *in* R. Rosen & F. Snell, eds, 'Progress in theoretical biology (Vol. 4)', Academic Press.

Holland, J. H. (1985), Properties of the bucket brigade algorithm, *in* 'First International Conference on Genetic Algorithms and Their Applications', Pittsburg, PA, pp. 1–7.

Jaakkola, T., Jordan, M. I. & Singh, S. P. (1994), 'On the Convergence of Stochastic Iterative Dynamic Programming Algorithms', *Neural Computation* **6**(6), 1185–1201.

Kaelbling, L. P. (1987), An Architecture for Intelligent Reactive Systems, *in* M. P. Georgeff & A. L. Lansky, eds, 'Reasoning About Actions and Plans', Morgan Kaufmann, pp. 395–410.

Kaelbling, L. P. (1990), Learning in Embedded Systems, Technical Report TR-90-04, Teleos Research.

Kearns, M. J. & Vazirani, U. V. (1994), *An Introduction to Computational Learning*, The MIT Press, Cambridge, Massachusetts.

Knuth, D. E. (1968), *The Art of Computer Programming*, Addison–Wesley, California.

Laird, J. E., Newell, A. & Rosenbloom, P. S. (1978), 'SOAR: An Architecture for General Intelligence', *Artificial Intelligence* **33**, 1–64.

Lieberman, H. (1993), Mondrian: a Teachable Graphical Editor, *in* A. Cypher, ed., 'Watch what I do: Programming by Demonstration', The MIT Press.

Maes, P. (1991), Learning Behavior Networks from Experience, *in* F. Varela & P. Bourgine, eds, 'Toward A Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life', The MIT Press, pp. 48–57.

Maes, P. & Kozierok, R. (1993), Learning Interface Agents, *in* 'Proceedings, AAAI-93', Washington, DC, pp. 459–465.

Mataric, M. J. (1994), Reward Functions for Accelerated Learning, *in* W. W. Cohen & H. Hirsh, eds, 'Machine Learning: Proceedings of the Eleventh International conference', New Brunswick, NJ, pp. 181–189.

McDermott, D. V. (1982), 'A temporal logic for reasoning about processes and plans', *Cognitive Science* **6**, 101–155.

Newell, A., Shaw, J. C. & Simon, H. A. (1957), Preliminary Description of General Problem Solving Program-I (GPS-I), Technical Report CIP Working Paper 7, Carnegie Institute of Technology.

Pazzani, M. J., Brunk, C. A. & Silverstein, G. (1991), A knowledge-intensive Approach to Relational Concept Learning, *in* 'Proceedings of the Eighth International Workshop on Machine Learning', pp. 432–436.

Pearl, J. & Verma, T. S. (1991), A Theory of Inferred Causation, *in* 'Proceedings of the Second Internation Conference on the Principles of Knowledge Representation and Reasoning', pp. 441–452.

Ramstad, R. M. (1992), A Constructivist Approach to Artificial Intelligence Reexamined, PhD thesis, MIT.

Rice, J. A. (1988), *Mathematical Statistics and Data Analysis*, Wadsworth & Brooks/Cole, Pacific Grove, CA.

Richards, B. L. & Mooney, R. J. (1992), Learning Relations by Pathfinding, Technical Report AI 92-172, Artificial Intelligence Laboratory, The University of Texas at Austin.

Rivest, R. L. & Schapire, R. E. (1989), Inference of Finite Automata Using Homing Sequences, *in* 'Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing', Seattle, WA, pp. 411–420.

Rivest, R. L. & Schapire, R. E. (1990), A New Approach to Unsupervised Learning in Deterministic Environments, *in* 'Proceeding of the Twenty-Eighth Annual Symposium on Foundations of Computer Science', Los Angeles, CA, pp. 78–87.

Rumelhart, D. E. & McClelland, J. L. (1986), *Parallel Distributed Processing*, The MIT Press, Cambridge, MA.

Shen, W.-M. (1993), 'Discovery as Autonomous Learning from the Environment', *Machine Learning* **12**, 143–165.

Sheth, B. & Maes, P. (1993), Evolving Agents For Personalized Information Filtering, *in* 'Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications'.

Shoham, Y. (1986), Chronological Ignorance Time, Nonmonotonicity, Necessity and Causal Theories, *in* 'Proceedings, AAAI-86', pp. 389–393.

Shoham, Y. (1987), 'Temporal Logics in AI: Semantical and Ontological Considerations', *Artificial Intelligence* **33**, 89–104.

Stein, L. A. & Morgenstern, L. (1991), Motivated Action Theory: A Formal Theory of Causal Reasoning, Technical Report AIM-1338, MIT Artificial Intelligence Lab.

Sutton, R. S. (1990), First Results with DYNA, an Integrated Architecture for Learning, Planning, and Reacting, *in* 'Proceedings, AAAI-90', Cambridge, Massachusetts.

Sutton, R. S. (1991), Reinforcement Learning Architectures for Animats, *in* 'First International Conference on Simulation of Adaptive Behavior', The MIT Press, Cambridge, MA.

Sutton, R. S. & Barto, A. G. (1987), A temporal-difference model of classical condition, *in* 'Proceedings of the Ninth Annual Conference of the Cognitive Science Society', Seattle, WA.

Sutton, R. S. & Barto, A. G. (1989), Time-Derivative Models of Pavlovian Reinforcement, *in* M. Gabriel & J. W. Moore, eds, 'Learning and Computational Neuroscience', The MIT Press.

Wald, A. (1947), *Sequential Analysis*, John Wiley & Sons, Inc., Chapman & Hall, LTD., London.

Watkins, C. (1989), Learning from Delayed Rewards, PhD thesis, King's College.

Wilson, S. W. (1986), Knowledge Growth in an Artificial Animal, *in* K. Narendra, ed., 'Adaptive and Learning Systems', Plenum Publishing Corporation.

Winston, P. H. (1992), *Artificial Intelligence (Third Edition)*, Addison–Wesley, California.