UNDERSTANDING WHAT MAY HAVE HAPPENED
IN DYNAMIC, PARTIALLY OBSERVABLE ENVIRONMENTS

by

Matthew Molineaux

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

| | |
|---|---|
| _____ | Dr. Gheorghe Tecuci, Dissertation Director |
| _____ | Dr. Mihai Boicu, Committee Member |
| _____ | Dr. Daniel Menasce, Committee Member |
| _____ | Dr. Kenneth De Jong, Committee Member |
| _____ | Dr. David W. Aha, Committee Member |
| _____ | Dr. Sanjeev Setia, Department Chair |
| _____ | Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering |
| Date: _____ | Summer Term 2017<br>George Mason University<br>Fairfax, VA |

Understanding What May Have Happened in Dynamic, Partially Observable Environments

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Matthew Molineaux
Master of Science
Depaul University, 2008
Bachelor of Science
Eckerd College, 2001

Director: Dr. Gheorghe Tecuci, Professor
Department of Computer Science

Summer Term 2017
George Mason University
Fairfax, VA

# Table of Contents

# List of Tables

# List of Figures

# List of Algorithms

# Abstract

UNDERSTANDING WHAT MAY HAVE HAPPENED IN DYNAMIC, PARTIALLY OBSERVABLE ENVIRONMENTS

Matthew Molineaux, PhD

George Mason University, 2017

Dissertation Director: Dr. Gheorghe Tecuci

In this work, we address the problem of understanding what may have happened in a goal-based deliberative agent's environment after the occurrence of exogenous actions and events. Such an agent observes, periodically, information about the state of the world, but this information is incomplete, and reasons for state changes are not observed. We propose methods a goal-based agent can use to construct internal, causal explanations of its observations based on a model of its environment. These explanations comprise a series of inferred actions and events that have occurred and continue to occur in its world, as well as assumptions about the initial state of the world. We show that an agent can more accurately predict future events and states by reference to these explanations, and thereby more reliably achieve its goals. This dissertation presents the following novel contributions: (1) a formalization of the problems of achieving goals, understanding what has happened, and updating an agent's model in a partially observable, dynamic world with partially known dynamics; (2) a complete agent (DHAgent) that achieves goals in such environments more reliably than existing agents; (3) a novel algorithm (DiscoverHistory) and technique (DiscoverHistory search) for rapidly and accurately iteratively constructing causal explanations of what may have happened in these environments; (4) an examination

of formal properties of these techniques; (5) a novel method (EML), capable of inferring improved models of an environment based on a small number of training scenarios; (6) experiments supporting performance claims about the novel methods described; and (7) an analysis of the efficiency of two DiscoverHistory algorithm implementations.

# Chapter 1: Introduction

Autonomous agents are proliferating quickly as computers become embedded in every aspect of daily life. Agents are embodied in the real world as autonomous vehicles, interact with humans in military simulations and games, and are used for complex predictions in agent-based models. The environments in which they act grow more and more complex as we rely on them more heavily. In the future, robots will be used in domains such as anti-submarine warfare, where the world is very hard to perceive and enemies may be present at any time. Simultaneously, as more robots are deployed, interest in greater autonomy increases. Where simple reactive or reflex agents were once acceptable, more complex deliberative goal-based agents are now desired to work in these complex environments. In particular, these agents must be able to achieve a large range of goals, specified at execution time, without constant human oversight. These goals must be specified with respect to objects and relationships in worlds with many objects and relationships. Some deliberative goal-based agents developed by the artificial intelligence community are already capable of handling large domains with many objects and relationships, but they do so by making strong assumptions about the nature of the environment. In fact, most goal-based agent research makes a variety of strong assumptions that do not hold in more realistic environments. In this work, we specify a novel agent and techniques capable of operating in larger worlds than comparable research systems, while relaxing the following assumptions:

1. the **static environment assumption**, which states that the environment is affected only by the agent, and by no other actors or natural processes,

2. the **full observability assumption**, which states that all information that impacts the agent's performance can be observed, and

3. the **known dynamics assumption**, which states that all possible environment transitions are known to the agent in advance.

We assert that goal-based agents that operate in large partially observable, dynamic, partially known environments are a critically important advance. In this dissertation, we will describe techniques for **explanation generation** and **explanation-based model learning** [1] based on DiscoverHistory search that make such agents possible. We will demonstrate in test environments that a novel goal-based agent that uses these capabilities, DHAgent, is able to achieve its goals more reliably, and achieve higher accuracy, lower computational complexity, and through use of fewer resources, when compared to other goal-based agents using identical knowledge.

In order to achieve its goals, a goal-based agent requires several reasoning capabilities. The first such capability is **planning**, a type of reasoning which constructs a series of actions that cause an agent's environment to transition from an initial state to one of a space of desired states, defined by its **goal**. Early research in artificial intelligence focused on planning, which is sufficient by itself to accomplish goals in environments that meet the above assumptions. However, in a dynamic environment, the agent must also be capable of **monitoring** the execution of its plans and **replanning** when they go awry. Here, replanning is essentially the same problem as planning, but must occur during execution, in response to developments found by direct perceptions of the environment made by monitoring. In a partially observable environment, an agent must also perform **belief management**, to infer unobservable aspects of a changing state that affect its goals. Finally, partially known dynamics force an agent to perform **model learning** to better predict the transitions along the path to a goal. Existing research has demonstrated goal-based agents that deal with the static environment assumption through planning, monitoring, and replanning; by performing belief management and model learning using novel techniques DHAgent will

[1] Explanation-based model learning is not closely related to explanation-based learning (DeJong and Mooney 1986; Mitchell, Keller, and Kedar-Cabelli 1986). We consider explanation-based model learning, which finds novel transition rules that supplement an environment model to correct false predictions. In contrast, explanation-based learning finds implication rules that supplement a logical domain to simplify inferences that the domain already supports.

relax all three assumptions. Figure 1.1 depicts a standard replanning agent and DHAGENT using these capabilities. Note that DHAGENT's learning cycle need not be synchronous with its execution cycle.

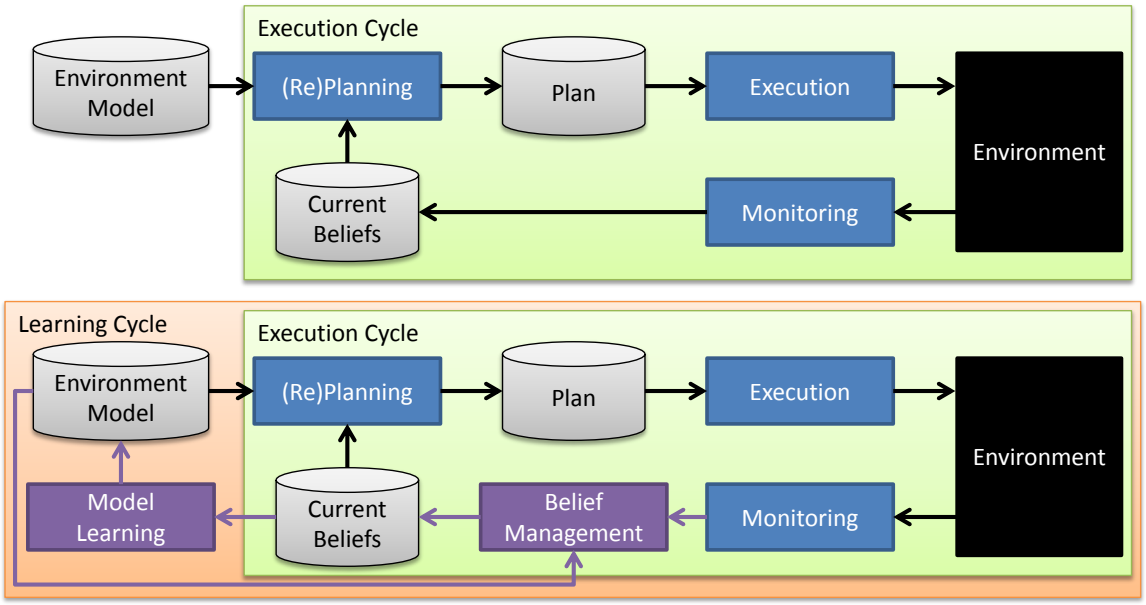


Figure 1.1: Frameworks for goal-based agents. Top shows a goal-based replanning agent. Bottom shows design of DHAGENT. Novel components and relationships shown in purple.

**Belief management** techniques estimate the current state of the environment for use in planning. This task is pivotal because incorrect beliefs cause planner mistakes. A complication to this task for a goal-based agent is the representation of a state using relations and objects, which is necessary for representing large environments with many types of information and arbitrary goals. Following Russell and Norvig (2009), we refer to a state representation based on relations and objects as a **structured representation**. Structured representations are strictly more expressive than propositional and feature-value representations, and allow algorithms the flexibility to work in larger and more complex environments. Belief management using a structured representation of the environment, however,

has received little attention from the academic community. ***A primary contribution of this dissertation is the*** DISCOVERHISTORY ***search technique, which performs belief management based on incremental improvements to explanations.*** Using DISCOVERHISTORY search as part of a goal-based agent permits the relaxation of the static environment and full observability assumptions.

**Model learning** allows an agent to improve a model of its environment during execution that allows it to achieve its goals. When a goal-based agent does not start with a complete and correct model of its environment, it may repeatedly construct plans with similar errors. For example, suppose that a Mars rover agent's directional sense is skewed by a nearby magnetic source. If it does not learn to model that effect, the rover's plans will consistently move it in the wrong direction. Learning a model that correctly predicts the direction of movement, however, would allow it to make correct plans. In order to perform this learning, it is important to understand where and how the agent's existing model is incorrect. This leads us to consider an **explanation-based model learning** technique that responds to understanding failures by considering revisions to the model. ***This technique is novel, as is our method for performing model learning as part of a goal-based agent.***

To manage beliefs, our agent performs **explanation generation**. The technique we use for this is called DISCOVERHISTORY search, based on the refinement method DISCOVERHISTORY which explores a search node in a space of explanations. Managing beliefs by reference to a maintained explanation of the environment ensures that the beliefs of an agent will, at any time, be consistent with observations made in the past, and its model of the environment. We assume that a goal-based agent receives structured observations of its world. For example, an observation might include the **literal** (AT ROBOT P1), which describes a relationship AT between two objects, ROBOT and P1. Explanation generation constructs an **occurrence history** consistent with these structured observations, a series of hypothetical assumptions about the past and past transitions that explain how the environment might have changed. By forming beliefs derived from an occurrence history, the agent takes into account unobserved external dynamics that otherwise it would have no

knowledge of.

Explanation generation can also help in learning, as the process of constructing occurrence histories given real observations provides an opportunity for reflection on and invalidation of an agent's environment model. We will expand in this dissertation on how to use opportunities in the explanation process to improve the agent's model.

To clarify these topics, we provide an example of how humans have generated and acted on explanations to improve the performance of a real world agent. In May 2005, NASA's Opportunity rover was crossing a dune on the surface of Mars when its human operators noticed an *inconsistency* [2] with their expectations: Opportunity did not move as far as the operators had projected when ordered to do so (Webster 2005). The operators were not able to observe the surroundings of the rover fully and precisely, but they nevertheless *explained* this inconsistency by *assuming* that the rover was stuck in loose soil. This explanation enabled the operators to formulate a new plan to escape from the unobserved loose soil and continue the mission. As Opportunity was out of contact with its operators for many hours each day when Mars faced in the opposite direction, this process was time-consuming and expensive. If Opportunity had been able to perform explanation generation, it would have had the capability to explain its own circumstances and then act to get itself out of its predicament, saving time and money.

To summarize, ***this work is concerned with the development of a goal-based agent and algorithms that relax three important assumptions: full observability, static environment, and known dynamics***. Unlike prior agents, DHAgent will use modules for belief management and model learning to overcome these assumptions. Both capabilities are based on DiscoverHistory search. We will describe and test claims regarding how reliably DHAgent achieves goals compared to other goal-based agents, as well as the accuracy and computational complexity of explanation generation, and the resources required to achieve goals.

[2] Future sections will address the role of inconsistencies in the explanation process.

## 1.1 Motivation Behind Explanation Generation Approach

In this subsection, we provide additional detail on why we perform explanation generation. Specifically, we provide additional clarity on how explanation generation supports reasoning under partial observability and how occurrence histories support learning, as well as other capabilities of future agents.

### 1.1.1 Partial Observability

The early history of AI research was characterized by papers that made a great number of assumptions about the environments with which agents would interact, so as to simplify the challenges facing early reasoning techniques. Over time, a great deal of work has gone into modifying algorithms that previously made restrictive assumptions about the world to operate in more realistic environments. One such assumption is the assumption of full observability: early planning systems were guaranteed to find plans provided that the environment was completely transparent. Unfortunately, many interesting domains are not fully observable.

Today, many researchers focus on relaxing the assumption of full observability. In partially observable domains, planning is more difficult because hidden factors affect the environment, so the effects of an agent's actions may be partially or completely unknown. Most current research into partially observable planning pursues one of two strategies. Agents that pursue the first strategy reason about a probabilistic state and probabilistic effects of actions (e.g., Kaelbling, Littman, and Cassandra 1998; Aberdeen and Buffet 2007; Yoon et al. 2008; Sanner and Boutilier 2009), and thereby consider the degree of likelihood of any individual fact being true. Agents pursuing the second strategy reason about a *belief state* (e.g., Hoffmann and Brafman 2005; Bryce, Kambhampati, and D. Smith 2006; Albore, Palacios, and Geffner 2009; Palacios and Geffner 2009), which is a composite state representing a large set of states that the agent might be in. Both of these approaches have disadvantages. Probabilistic reasoning requires that a knowledge engineer specify the prior probability of all propositions that may be satisfied; these probabilities are often specified

arbitrarily, because accurate data is unavailable or too expensive to obtain. Furthermore, most modern probabilistic reasoning systems also fail to abstract over properties of a state and objects present in it, which causes their speed to sharply decrease in more complex domains where the number of properties tracked grows exponentially. Agents that pursue the second strategy, acting in a space of belief states, have a similar problem. The space of belief states, which encompasses all possible worlds, grows exponentially with the number of unknown facts. Therefore, in complex domains, these techniques tend to fail as well. Unless they refuse to consider unlikely scenarios, which entails some measure of risk, all probabilistic reasoning systems must fall into this trap as well.

We are committed to demonstrating that our agents can be successful working with larger, more realistic environment models. By investigating a new technique which avoids these pitfalls, we sidestep the complexity issues that restrict existing techniques to smaller environments. This is done through a design decision fundamentally different from both approaches discussed above. Each earlier approach is fundamentally *anticipatory*: techniques using either strategy must consider the space of states that might occur in the future. Our techniques deal with partially observable worlds *retrospectively*. Explanation generation occurs during execution rather than before it, so much more information is available. This extra information reduces the space of possibilities greatly. The primary disadvantage of this approach is that our agents must start interacting with the world without first preparing for all possibilities. When compared to existing methods that analyze all future possibilities, this is dangerous; however, in the larger domains we consider, the strategy of considering all futures becomes intractable. Therefore, the only logical course is to accept some measure of risk.

To summarize, explanation generation techniques aim to support agents that reason about hidden information in large partially observable worlds while avoiding the exponential explosion that occurs when considering all possible states the agent might be in. Addressing this hidden state is necessary to improve prediction in realistic domains and thereby to improve overall rate of success at achieving goals.

### 1.1.2 Occurrence Histories

Explanation generation is based on finding occurrence histories, sequences of actions and events that underlie the progression of the environment. Sensors provide information about the environment state, but not the actions and events that explain why successive observations are different. However, knowing the actual changes that occurred is powerful. We anticipate that knowledge of occurrence histories will have the following anticipated benefits for autonomous agents: (1) occurrence histories highlight gaps in an agent's knowledge where observations do not match up with expected events, (2) occurrence histories can show how multiple actors separately influence the state, and (3) occurrence histories provide the skeleton for a human-readable narrative. Of these, we will demonstrate the first and second benefit. We do not, however, show how the second benefit can increase performance of an agent. Examination of the third benefit is left for future research.

Occurrence histories may include gaps where no cause can be inferred for some event. Such gaps make clear that there is some unknown dynamic in the environment which the model does not account for. Knowledge of these gaps can be used to improve environment models over time by targeting these gaps for learning. We will test this hypothesis within the scope of this dissertation, and demonstrate how model learning in response to occurrence history gaps can improve the execution performance of an agent.

Causal links between occurrences are apparent in occurrence histories where the effect of one occurrence is a condition of a later one. By tracing these causal links backward in time, it becomes possible to attribute an event to one or more particular actors who caused it. This would otherwise be impossible when an agent's actions are temporally distant from the intended changes. This causal attribution gives evidence as to the intentions of those actors, which can be analyzed through an intent recognition or plan recognition process. In this dissertation, we will analyze the accuracy of our explanation generation techniques at attributing actions to actors. However, we will not determine the impact of performing plan recognition based on those actions, a study we leave for future work.

Finally, we anticipate that occurrence histories could form the basis of an agent capability to construct a *narrative* about its experiences. In general, narratives are based on a describing a series of events, whether real or fictional. We expect this capability to be useful in communications with humans in the future. However, this dissertation does not provide evidence for this; see Future Work in section 9 for more discussion.

In summary, we expect histories of occurrences to support model learning, attribution of actions, and construction of narrative. Thus, creating such histories is of great importance to autonomous agents and merits study.

## 1.2 Problem Statement

***The central contributions of this dissertation are the the*** DiscoverHistory ***search technique, which generates*** explanations ***of the environment an agent inhabits, and*** DHAgent***, an*** agent ***that uses*** DiscoverHistory ***search to achieve goals in partially observable, dynamic, partially known environments.*** In this section, we provide a clear statement of the problems addressed by DiscoverHistory search and DHAgent in formal language, which provides a clear breakdown of the problem of achieving goals into multiple subproblems. We follow this with a discussion of what guarantees can be made regarding those problems. In the following section, we give a more detailed discussion of the specific claims this dissertation will defend and the contributions made.

We model an environment $\Sigma$ as having a state space $S$, an action space $A$, and an event space $E$. An agent $\alpha$ interacts with its environment by receiving a sequence of deterministic, noiseless observations $\mathbf{o} = \{o_0, o_1, o_2, \dots\}$, and acting upon the environment through a series of actions $\mathbf{a}_\alpha = \{a_1, a_2, \dots\}$. An observation $o$ is generated by a deterministic, noiseless function $Obs(s)$, where $s$ is an instantaneous environment state. The first state for which the agent receives an observation is called the **initial state**, $s_0$.

Changes in the environment are modeled as a **transition function** $\lambda : S \times A \to S$. This function describes how actions and events cause the environment to transition to new states. Actions are intended by an agent (either $\alpha$ or some other agent), and occur only when an

agent decides they should. Events do not appear in the definition of the transition function because they are caused deterministically. We assume that in any state when one or more events apply, only the null action $\emptyset$ may occur. From the perspective of a reasoning agent, actions of other agents are termed **exogenous actions**. Events occur deterministically when their conditions are met; an event must occur directly after an action or other event.

An environment model $M_\Sigma$ defines a transition function $\lambda_\alpha$ and observation function $Obs_\alpha$ that are available to the agent. When the environment model is incomplete or incorrect, $\lambda_\alpha$ will not match the true environment transition function $\lambda$, and an agent predicting future states based on it will sometimes be incorrect. For more details, see Chapter 3.

**Problem 1** (Goal Achievement Problem)

*Let $\alpha$ be an agent in an environment $\Sigma$, and $\mathbf{a_x}$ be a sequence containing all exogenous actions taken in $\Sigma$ by agents other than $\alpha$. The goal achievement problem for $\alpha$ is as follows:*

- ***Given**: a (possibly incomplete) model of $\Sigma$ ($M_\Sigma$), a stream of deterministic noiseless observations* $\mathsf{obs}$*, and a boolean **goal function** over the state space $g : S \to \{T, F\}$,*
- ***Find**: an action sequence $\mathbf{a}_\alpha$ that cause the environment $\Sigma$ to transition to some state $s_g$ such that $g(s_g)$.*

*More formally: let $\mathbf{a}$ be a sequence containing all actions that take place in environment $\Sigma$ during some time period, starting with the action $a_0$ taken in $s_0$. Sequences $\mathbf{a}_\alpha$ and $\mathbf{a_x}$ are subsequences of $\mathbf{a}$; all entries in $\mathbf{a}$ not in $\mathbf{a}_\alpha$ and $\mathbf{a_x}$ must be the null action $\emptyset$. To succeed, the agent must arrange that the actions in $\mathbf{a}$ cause $\Sigma$ to transition to a goal state: $g(\lambda(\lambda(\ldots\lambda(\lambda(s_0, a_0), a_1), \ldots, a_{n-1}), a_n)) \to T$.*

Notice that $\alpha$ never has access to the set of exogenous actions $\mathbf{a_x}$, even though those actions may affect its plans. Furthermore, as the environment model may be incomplete, the agent may never be certain what state has occurred based on the observations it is given. However, the problem requires that a property hold for a state that has occurred. Therefore, there is no sound algorithm that solves this problem for all environments. Instead, we study approximate solutions. These solutions either assume the model is complete or attempt to

provide iteratively improved approximations to the transition model so that soundness can be reached in the limit.

This problem definition avoids common assumptions described earlier. The problem is partially observable through a sequence of observations, rather than assuming the agent has access to the states. The problem is dynamic due to the presence of exogenous actions and events. Finally, the agent's access to a possibly incomplete model of the environment rather than its true transition function results in partial knowledge of dynamics.

In order to solve the goal achievement problem, we break it into three subproblems, which we will describe now. First, we must be able to find a plan that will (at least in some possible world) achieve its goals. We expect that this plan may not succeed, but executing such a plan may nonetheless be helpful, either through decreasing the number of steps necessary to reach the goal, or providing new information about the environment. Planners that work under restrictive assumptions are well understood and easily available. We propose to use a standard solution to this type of problem, together with other algorithms that compensate for its inadequacies in the presence of partial observability, dynamic transitions, and an incomplete model. A standard continuous planning problem can be described as follows:

**Problem 2** (Continuous Planning Problem)

- ***Given****: a complete environment model $M_\Sigma$, a current state $s_c$, search knowledge $ks$, and a goal function $g : S \to \{T, F\}$,*
- ***Find****: a plan (sequence of actions) $\mathbf{a}_{\alpha,\mathbf{i}} = [a_{\alpha,i}, a_{\alpha,i+1}, \ldots a_{\alpha,i+m}]$ in the environment $\Sigma$ that will reach the goal state assuming the state is correct and no outside actions occur, i.e., $g(\lambda(\ldots \lambda(\lambda(s_c, a_{\alpha,i}), a_{\alpha,i+1}), \ldots, a_{\alpha,i+m})) \to T$.*

As many fast planners require advice beyond the transition function to achieve reasonable results, the goal achievement problem includes search knowledge $ks$ to describe any static body of information, not related to a specific planning scenario, used by a planner to guide its search.

As we intend to apply a planner that solves this problem as part of the solution of Problem 1, we need to contend with its flawed assumptions. As this problem assumes knowledge of the entire state, we must attempt to approximate it based on observations received; this is the task of belief management. We will also improve the internal environment model so that it better approximates a complete environment model; this is the model learning task. Accomplishing these two tasks can be done by solving the **hypothesis generation problem** and **hypothesis-based model learning problem**.

The hypothesis generation problem requires an agent to manage its beliefs in order to infer occurrence histories and state information that are consistent with its environment model, received observations, and past actions. The output of explanation generation is a set of **hypotheses** about the environment, each of which consists of a set of unproven logical statements about the environment, including statements about the occurrence of events and exogenous actions, and a set of initial state assumptions, and a set of transition discontinuities that describe gaps where the occurrence history does not match the model. A **plausible** hypothesis, taken together with the agent's actions and environment model, must imply the observations received. The space of possible hypotheses is written $H$.

**Problem 3** (Hypothesis Generation Problem)

- ***Given****: a (possibly incomplete) environment model $M_\Sigma$, observations of the environment $\mathbf{o} = [o_0, o_1, \ldots o_m]$, an action history $\mathbf{a_h} = [a_{h,1}, a_{h,2}, \ldots a_{h,n}]$ containing actions taken by $\alpha$, and a total order $\prec$ describing the temporal ordering of actions and observations,*
- ***Find****: a set of plausible hypotheses $H_\alpha$ such that $\forall h \in H_\alpha : h \cup \mathbf{a_h} \cup M_\Sigma \models \mathbf{o}$.*

Supposing that an environment model is incomplete, the transition function it represents must be incorrect for one or more states, by definition. Problem 3 accommodates this by allowing discontinuities, which describe where an occurrence history is not consistent with the model. If a hypothesis is correct with respect to all initial state assumptions and exogenous action assumptions, but still has discontinuities, the model itself must be

incorrect. This is the basis for a definition of the model learning problem. We search for small revisions to the model that add extra events to the occurrence history, eliminating the discontinuities. If the discontinuities are a representative sample, such changes can improve the accuracy of explanation, as well as the efficiency and reliability of plans that use the same model. For more details, see Chapter 8.

**Problem 4** (Model Learning Problem)

- ***Given****: a (possibly incomplete) environment model $M_\Sigma$, a set of transition discontinuities* $\mathbf{d}$*, observations of the environment* $\mathbf{o}$*, and an action history* $\mathbf{a_h}$*,*
- ***Find****: a new model $M'_\Sigma$, such that for some hypothesis $h$ with no discontinuities, $h \cup o_0 \cup \mathbf{a_h} \cup M'_\Sigma \models \mathbf{o}$.*

We have stated that Problem 1, the goal achievement problem, can be approximately solved by splitting it into three separate problems: Problem 2, the continuous planning problem; Problem 3, the hypothesis generation problem; and Problem 4, the model learning problem. Due to the approximate and highly uncertain nature of all these problems, we can rarely make any kind of guarantee of success. However, a notion of increased success drives all of our work. Therefore, we present here arguments for why an agent design based on solving the continuous planning problem, the hypothesis generation problem, and the hypothesis-based model learning problem will succeed under certain restrictive assumptions.

First, we argue that under certain conditions, explanation generation that improves an agent's knowledge must lead to success in a partially observable, static environment.

**Argument 1**

*Assuming a complete environment model in a partially observable, static, connected environment (where every state is reachable from every other state, and no exogenous actions occur), it is possible for an agent to solve the goal achievement problem by iteratively applying a solution to the continuous planning problem to its current beliefs, executing the resulting plan, and applying a solution to the hypothesis generation problem to change its*

*beliefs whenever it reaches a state that it could not predict, until its goal is reached. Each plan made by the agent will, according to its predictions, reach the goal, from the definition of the continuous planning problem. Therefore, every plan that does not reach the goal fails to match its predictions, and thus the observations found when executing that plan contain new information. The agent must continue executing plans until it reaches its goal, and the goal will always be attainable because the environment is connected. Therefore, as long as the agent fails to reach its goal, the available information in its observations must increase.*

*The explanation generation method forms hypotheses based on the available observations. As the information in those observations increases, the number of plausible hypotheses reduces. As no new information enters the environment (because it is static), the agent must eventually either know the entire state of the environment, because only one hypothesis is possible, or reach its goal by making plans with incomplete knowledge. Once the complete state of the environment is known, the agent should be able to predict the results of all future actions. This means that a plan returned will, by definition, be correct, since the agent's beliefs are correct. Therefore, its new plan will reach the goal.*

Next, we argue that the same solution works even in a dynamic environment, given further assumptions.

**Argument 2**

*Assuming a complete environment model in a partially observable, dynamic, connected environment (where every state is reachable from every other state, and exogenous actions occur), the solution of argument 1 still applies, but the information present in the environment can increase due to the actions of other agents. Therefore, the agent might never obtain enough information to be guaranteed to reach the goal.*

*If we assume that information is introduced to the environment more slowly than the agent gains it, then the agent will eventually find the correct state of the environment, as in argument 1. More generally, we can assume that there is some knowledge about the environment that can be attained despite its dynamic nature, which is sufficient to make the agent more likely to make forward progress than not (i.e., reduce the number of plan steps*

*necessary to achieve its goal). Given this assumption, the agent's actions will cause it to converge to a goal state.*

To understand this assumption, consider a predator-prey scenario. The predator wants to kill the prey, but the prey is always moving, and is faster than the predator. Usually its location is not observable to the predator and new information (the prey's actions) are entering the environment all the time. Many other features of the environment, such as feeding areas, weather, and light availability may be constantly changing and not observable; the predator has no hope of ever determining the true state of the environment. However, if the predator obtains a certain critical piece of information, the location of the prey's nest, a simple plan to lie in wait is likely to succeed despite the many unknowns in the environment. Therefore, the **assumption of sufficient knowledge to progress** is satisfied, and the predator can accomplish its goal of killing the prey.

Third, we show that under certain conditions, explanation generation that improves an agent's knowledge and model learning that improves an agent's model must lead to success in a partially observable, dynamic, partially known environment.

**Argument 3**

*Assuming an incomplete environment model and a connected, dynamic environment, it is possible to solve the goal achievement problem using the method described in argument 2, with the additional application of a solution to the hypothesis-based model learning problem in response to transition discontinuities. We distinguish two cases. In the first, sufficient knowledge can be attained by the agent to progress toward a goal,* ***even based on the current, incomplete environment model****. This reduces our argument to argument 2. In the second case, it is possible for the agent to obtain sufficient knowledge to progress, but must correctly predict some transitions that the current environment model does not. If neither is true, and sufficient knowledge to progress will not be available, no guarantee can be made.*

*In the second case, the agent will be unlikely to make progress until it improves its model. In order to obtain a complete model, we assume that, with some frequency, it makes*

*observations before and after a transition that its internal model predicts incorrectly. Furthermore, we must assume that sufficient information is gained about these transitions during an agent's attempts to reach its goal to eliminate hypotheses with incorrect discontinuities (as a correct hypothesis cannot be conjectured before learning). Under these assumptions, the agent will generate more and more examples of the discontinuities over time. Each time model learning occurs, the new model $M'_\Sigma$ will correctly predict a larger and larger percentage of transitions (by the definition of the hypothesis-based model learning problem). Eventually, the environment model will correctly predict enough transitions that sufficient knowledge to progress is attainable, by the definition of case 2, and argument 2 applies.*

In many practical situations, including the environments discussed later in this paper, the assumptions these arguments are based on do not hold. However, these arguments give us some intuition as to the limits of performance and what we can hope to achieve. We will rely on experimental evidence in future chapters to give us a picture of how well these techniques succeed under weaker assumptions.

## 1.3 Contributions

The contributions of this dissertation are as follows:

1. We introduce novel formalizations for the goal achievement problem, hypothesis generation problem, and hypothesis-based model learning problem (see Section 1.2).

2. We introduce a new algorithm, DiscoverHistory, used in a search process to form hypotheses about the past in partially observable, dynamic environments (see Chapter 4). DiscoverHistory search is a general solution to the hypothesis generation problem; its accuracy is equivalent to a state of the art deductive method, and it reduces execution time requirements by 95% in the ASM domain with respect to this method (see Chapter 7).

3. We describe sufficient conditions under which DiscoverHistory search is a sound

and complete solution to the hypothesis generation problem (see Section 4.5).

4. We introduce a new general method for acting to achieve goals in large partially observable, dynamic environments with partially known dynamics, DHAgent (see Section 4.6). DHAgent solves the goal achievement problem, and in our experiments achieves 20-63% more goals than a typical replanning agent in the Hazardous Rovers, and Hazardous Satellites domains (see Chapter 5).

5. We introduce a novel general method, the Explanation-Based Model Learner (EML), for inferring improved environment models based on explanations of what has occurred in a partially observable, dynamic, partially known environment. EML is the first general solution to the hypothesis-based model learning problem, and achieves 90% of an optimal performance improvement within 5 learning trials in the Mudworld and Satellites domains (see Chapter 8).

6. We present experiments demonstrating the performance improvements described for DHAgent, DiscoverHistory, and EML. (see Chapters 5, 7, and 8)

7. We present a parametric analysis of the computational efficiency of two implementations of the DiscoverHistory algorithm (see Chapter 6.

In this dissertation, we use experimental evidence to defend three claims about the quality of the solutions found by the new techniques DHAgent, DiscoverHistory, and EML:

**Claim 1**

*Belief management using* DiscoverHistory *search causes the goal achievement performance (number of goals achieved successfully) of* DHAgent *in a partially-observable, single-agent context to improve relative to a traditional replanning agent,* IncuriousAgent*, which uses a more naive belief management strategy.*

**Claim 2**

DiscoverHistory *search is faster than a deductive strategy,* DEG*, for generating and*

*maintaining explanations of exogenous actions in a dynamic, partially observable, multi-agent environment, and maintains comparable accuracy.*

**Claim 3**

*Our novel explanation-based model learning algorithm,* EML*, infers successively improved environment models that reduce the execution cost incurred by* DHAgent *in accomplishing goals in dynamic, partially observable environments.*

## 1.4 Organization of the Dissertation

Chapter 2 will discuss related work. Chapter 3 discusses definitions and representations used throughout the dissertation. Chapter 4 introduces the central explanation generation technique, DiscoverHistory search, and DHAgent, an agent that solves the goal achievement problem. Chapter 5 discusses the suitability of DiscoverHistory search for belief management, including an empirical investigation of the goal achievement performance of DHAgent, addressing Claim 1. Chapter 6 describes an analysis of DiscoverHistory's efficiency. Chapter 7 discusses the accuracy of explanations generated by DiscoverHistory search in a multi-agent environment, and addresses Claim 2. Chapter 8 introduces our novel model learning algorithm, EML, and provides empirical results demonstrating its efficacy at improving performance in a domain, addressing Claim 3. Chapter 9 reviews our claims and contributions, and discusses limitations and extensions planned for future work.

# Chapter 2: Related Work

In this chapter we survey related work that defines pivotal concepts, resembles our own, and provides algorithms for solving similar problems. We cover the following topics: Abductive Inference and Explanation, which provides context and logical underpinnings for explanation generation; Representations for Change, which covers ideas on how to represent dynamic worlds; Belief Change, which discusses how an agent should revise and update its beliefs after taking actions and making observations; Model-Based Diagnosis, which describes how to reason about systems that do not act as anticipated; and Planning, which describes agents that construct plans to achieve goals given descriptions of their environment.

## 2.1 Abductive Inference and Explanation

According to John and Susan Josephson (1996), abduction is "a form of inference that goes from data describing something to a hypothesis that best explains or accounts for the data". In general, abductive inference can be described as a procedure which infers *possible causes* for *observed evidence*. Explanation, therefore, is an abductive procedure.

Abductive inference has been classified into multiple categories by Eco (1983) and Thagard (1993), as reported by Schum (2001). Eco's categorization is based on a notion of degree of creativity. From least to most creative, his categories are: *overcoded*, referring to single-step inferences that use prior knowledge without ambiguity, as when only one event is possible; *undercoded*, referring to single-step inferences that use prior knowledge with ambiguity, as when multiple alternative events may have occurred to cause the same observed evidence; and *creative*, which refers to single-step inferences without prior knowledge, as

when no known event might cause the observed evidence, but nevertheless an event is posited which did. In addition to these three categories, Eco describes *meta abduction*, which refers to a recursive inference procedure that allows support that cannot be immediately found to be inferred by abduction itself.

Thagard's classification of abductive inference is based on the types of information that are inferred. Thagard describes *simple abduction*, in which unobserved properties of some object are given as explanation for the presence of an unusual property of an observed object; *existential abduction*, in which the existence of some unobserved object is posited in order to account for an observed object having an unusual observed property; *analogical abduction*, in which a hypothesis is refined or extended to include additional explanatory information, based on prior experience; and *rule-forming abduction*, in which generalization theories are formed that allow one to perform inference at the point that inference is needed.

According to this taxonomy of abductive inference, we are concerned particularly with algorithms that perform overcoded, undercoded and meta-abduction using simple and existential techniques. There's a long tradition in artificial intelligence of reasoning using abductive inference, going back at least to Pople's (1973) system for generating logical hypotheses.

Levesque (1989) gave a formal semantics for a logical operator EXPLAIN, which conducts abductive inference over beliefs . Morgenstern and Stein first recognized the problem of explanation as it applies to sequences of actions (1988). In this work, they suggest that a reasoning process of explanation complements the reasoning process of temporal projection, and that explanation should be conducted whenever prior predictions prove false. They provided an early formalization of both temporal projection and explanation based on a simple temporal logic that resembles a subset of the situation calculus.

Shanahan (1993) examined a problem he referred to as *temporal explanation* or *postdiction*, in the context of the situation calculus. He defined temporal explanation as "reasoning backwards in time from events to causes." This definition is effectively equivalent to our own, and Shanahan asserts that the problem is "as important as prediction" and is "a

fundamental mode of reasoning in its own right." Essentially, he argues that explanation should be given more attention as a basic style of reasoning. Shanahan considers both deductive and abductive modes of inference, but argues that abduction is the more natural approach and deduction must simulate abduction to perform explanation. Shanahan points out that many questions remain, including which predicates should be abducible, why some facts demand explanation, and how to deal with the situation where multiple minimal consistent explanations exist. A robot created by Shanahan (1996) performs sensor data assimilation using this process (see Section 2.5.3).

The scientific discovery system STAHL (Rose and Langley 1986), the planning/interpretation system GORDIUS (Simmons and Davis 1987), and the case-based reasoner CHEF (Hammond 1990) all share a focus on abductive explanation of failures in their own outputs and repairing them. The overall goal of adding explanation to each was to produce a stronger result by adding a metareasoning algorithm that found inference failures and fixed them. Each system conducts explanation by doing a search within the space of the internal data structures created by that system.

Work in abductive inference that is most closely related to ours includes the Inductive Process Modeling precedure of Bridewell et al (2008) and Forbus and Falkenhainer on Self-Explanatory Simulations (1990). Both methods construct explanations made up of processes based on a model of possible processes, given a time series of world states to explain. These systems are highly advanced in their ability to operate in continuous worlds and generate explanations based on events, but their representations are specialized to, in the first case, continuous descriptions of the world, and, in the second case, qualitative descriptions.

AbRA (Bridewell and Langley 2011) also incrementally constructs explanations. It can perform online plan recognition, a sequential task, but does not infer specific occurrences based on a domain model. UMBRA (Meadows, Langley, and Emery 2013) performs plan understanding by incrementally constructing explanations, and infers an agent's beliefs, desires, and intentions to explain observed actions, rather than inferring occurrences to explain observed states. As, for example, Ram (1993) and Leake (1995) have noted, these tasks are

likely to be highly important to a range of cognitive agents. However, their requirements differ significantly from the task of inferring occurrences that explain observations.

We will discuss further systems that perform abductive reasoning to find explanations for observed evidence as part of the research field *model-based diagnosis* (see Section 2.4).

## 2.2 Representations for Change

There are many frameworks and representations in Artificial Intelligence for reasoning about dynamic systems and environments. We survey here those representations which provide important context for our own work, in particular those representations which describe ways of reasoning about more complex environments.

The *situation calculus* (McCarthy and Hayes 1969) is one of the earliest and most influential frameworks for describing change over time in Artificial Intelligence. This framework describes the world in terms of situations, which correspond to discrete time points; fluents, which are facts about or properties of situations; and actions, which move the described world from one situation to the next. All fluents in the situation calculus include an argument describing the situation they refer to, so that one can state that a fact was true at one time (in one situation), and later false (in another situation). The effects and preconditions of actions, as well as constraints between fluents, are described by rules in first-order logic. Through use of these rules, as well as a logic reasoner or theorem prover, questions about the values of fluents in situations following the application of a sequence of actions can be answered via a proof procedure. Doing so is called *projection*, and is a critical step in the planning process. However, planners using the situation calculus for projection are highly inefficient compared to modern algorithms. Logical frameworks such as the situation calculus are primarily useful for proving theorems about change.

Allen's interval logic (Allen 1984) solved the problem of concurrency by describing the world in terms of time intervals and fluents that hold over them rather than actions that cause situations to change. Allen's logic introduced the term *occurrence*, a general term for all types of change, including actions, *processes*, and events. The most important novel

feature of Allen's logic is abstract time periods called *intervals* and a basic set of relations that range over them. These interval relations allow the expression of constraints between occurrences, as well as the expression that facts hold during intervals rather than at specific time points or in the "eternal now" corresponding to non-temporal logic. This provides a foundation for a great deal of work in logic and explanation, but does not address issues of time measurement and cannot easily express instantaneous change, which is easily handled by the situation calculus.

The AI planning community works with representations of change as well, which are somewhat different due to their focus on efficiency. It has been argued (H. J. Levesque and Brachman 1984) that there is an inevitable tradeoff between the tractability of a representation language and its expressiveness. In the planning community, therefore, somewhat less expressive representations are used that are less computationally complex. The Action Description Language (ADL), created by Pednault (1987), is a popular and influential representation used in the AI planning community. Pednault (1989) described the language as a compromise between efficiency and expressiveness; in particular, he provides a conversion to the situation calculus, showing that it is strictly less expressive, but also much more succinct, and allows for more powerful and efficient inference.

A successor language to ADL, the Planning Domain Definition Language (PDDL), provides the basis for the most common formalisms used in the planning community today. An extended language, PDDL+, created by Fox and Long (2002; 2006), includes change representations based on models of actions, events, and processes. In PDDL+, processes describe continuous change, and events cause discrete change. While PDDL+ processes take some amount of time to occur, events are instantaneous, just like executed actions. Both processes and events can occur due to exogenous causes. PDDL+ does not represent intervals, due to its lineage from the situation calculus; however, because planning languages make no restriction on planner implementation, it is not necessary for a system using PDDL+ to maintain the computationally expensive global situations of the situation calculus.

## 2.3 Belief Change

When an agent takes actions and receives observations, it needs to change its beliefs about the world. *Belief revision* is the process of minimally updating an agent's beliefs in response to receiving new information, which may contradict its prior beliefs. This process was initially described by the logical theorists Alchourrón, Gärdenfors, and Makinson (1985), who defined a set of constraints referred to as the *AGM postulates* that any reasonable operator for updating beliefs should follow. While much useful work follows the AGM postulates, Darwiche and Pearl (1997) showed that belief revision operators that follow them do not necessarily describe a rational process when used in an iterated update cycle; in response, they proposed a new model for *iterated belief revision*. However, belief revision by itself is not appropriate for agents interested in maintaining beliefs about a changing world.

Updating beliefs in response to the world changing is a different process, called *belief update*, codified by Katsuno and Mendelzon (1991). Belief update has been recognized as equivalent to finding the state following an action given the action and prior state, also known as the task of *progression* (Del Val and Shoham 1994; Liberatore 2000; Lang 2007). However, Lang (2007) showed that Katsuno and Mendelzon's definition of belief update does not properly model changes in beliefs about the past, and therefore cannot be used to perform correct inferences in response to sensing actions or observations. Lang suggested that this is a separate task, called *reverse update*, and invented two operators for performing this task that correspond to rational algorithms for *postdiction* (determining what was true in the past) and *goal regression* (determining whether an expression will be true after a sequence of actions). However, neither the original belief update nor either reverse update provides any guidance as to how to choose what beliefs to give up when contradictions occur.

Dupin de Saint-Cyr and Lang (2002; 2011) suggested a new operator for belief modification, called *belief extrapolation*, which is based on a notion of explanation. Belief

extrapolation describes how to find a minimal set of changes to the truth of propositions that take place during a *history*, an ordered list of observations. Each possible state path consistent with the observations is called a *trajectory*. Once a set of trajectories is found, the problem of deciding what to believe can be reduced to determining which trajectory is most preferred; the state at the end of the most preferred trajectory should be believed. Dupin de Saint-Cyr and Lang provide a list of possible trajectory preference relations. These include *number of changes*, a relation that minimizes the number of times any variable's value changes during a trajectory; *change set inclusion*, which minimizes the number of time points at which changes occur; *inclusion of changing fluents*, which minimizes the number of distinct state properties that change; and *penalty*, in which each type of fluent change is assigned a cost, and that cost is minimized across the trajectory. *Chronological minimization*, a technique from research on nonmonotonic logic, prefers trajectories in which changes are made as late as possible; its reverse is *anti-chronological minimization*. Finally, the *event penalty* preference relation requires that changes occur as the result of events or event sequences; the relation minimizes the cost of those sequences, where each event has an associated cost. The choice of preference relation determines what explanations are accepted, and therefore what is believed by the agent. Like belief revision, belief extrapolation helps decide what to believe when new beliefs contradict old beliefs; like belief update, belief extrapolation accommodates belief change due to changes in the world. Dupin de Saint-Cyr and Lang (2011) identify strong connections with work on dynamical diagnosis, some of which might be construed as instances of the belief extrapolation model; however, belief extrapolation is defined only for the case of propositional databases. This work provides strong theoretical results, but falls short of practicality, providing no notion of legal transitions or intentionality, nor any implementations.

Boutilier and Becher (1995) described a logical framework that supports the definition of abductive rationales for belief revision. This relationship is established without violating the traditional definition of belief revision codified in the AGM postulates due to Alchourrón *et al* (1985). The belief revision operators described by Boutilier and Becher modify the

beliefs of an agent by choosing those beliefs implied by the most preferred explanation that predicts a new observation (i.e., a new belief to be adopted). Boutilier and Becher showed how this notion captures the reasoning performed in early diagnosis systems that did not reason about causality. Boutilier refers to causal explanations as being an important area for extensions to this work.

Boutilier (1995) extended this framework with a description of a belief update operator based on abduction, arriving at a generalized update operator that finds possible successor worlds using both revision and update . Although this new operator incorporates a notion of events, it does not reason about history. Instead, Boutilier (1998) created an iterated belief semantics by which an agent could maintain a qualitative ranking over all possible worlds. This ranking, described first by Friedman and Halpern (1994), is also referred to as an *epistemic state*, because it constitutes knowledge about an agent's own beliefs. Based on the epistemic state, an agent can be said to believe, at any time, facts that are true in the intersection or union of all maximally plausible worlds. Neither Boutilier nor Friedman and Halpern commit to a particular ranking method, preferring to establish frameworks useful for any such ranking. An agent is required to fully update and rank all possible worlds after every observation, at high computational cost. Unlike our work, none of Boutilier's contributions consider the possibility of event sequences between observations or concurrent events, nor do they generalize beyond propositional logics.

Many agents use a process of *projection* for maintaining their beliefs, which means determining all possible states of the world by inferring a first-order logic expression that is consistent with all possible occurrences following an initial state, also described in first-order logic. Simple and efficient procedures exist for doing so in deterministic, fully observable closed worlds, and in this setting the problem has been shown to be equivalent to a series of relational database updates (Reiter 1987b). However, it is more difficult in partially observable worlds, where belief update at execution time may be necessary. Provably correct and efficient local projection algorithms exist for certain specialized cases (De Giacomo and H. J. Levesque 1999; Liu and H. J. Levesque 2005), but in general the complexity of

full projection in these environments is unknown. Projection results in provably correct descriptions of the world under certain assumptions, and therefore is used by most planning algorithms, which prize correctness above efficiency. Use of projection in contingent and conformant planners is primarily responsible for their high computational complexity (Albore, Palacios, and Geffner 2009; Bonet and Geffner 2012), due to the need to represent the large number of possible future states of a conditional planning problem.

Liu and Levesque (2005) showed that progression is tractable despite incomplete knowledge under the assumption that all actions have only local effects. This assumption requires that actions change only properties of objects named in those actions parameters, and is violated by any domain in which agents can influence objects they did not intend to. Vassos and Levesque (2007) extend their theory to apply to the Situation Calculus with functional fluents.

Son and Baral (2001) devised a set of algorithms that maintain an approximation of an uncertain world state. Use of these approximations substantially reduces the theoretical complexity of planning problems (Baral, Kreinovich, and Trejo 2000), but planning based on such approximated states is not guaranteed to succeed, even when a feasible plan exists. These approximated states, known as $\omega$-*approximations*, incorporate all facts that can be unambiguously determined from a short history of length $\omega$. Each fluent in these approximations is assigned a value "true", "false" or "uncertain". In the least complex version, *0-approximation*, successor states are computed by finding all possible worlds that could result from the application of an action, and assigning "uncertain" as the value of any fluent that has different values in two of those possible worlds. At higher levels of approximation, more computation is performed to try to determine the value of a fluent. Specifically, the values of fluents are determined by projecting the effects of the prior $\omega$ actions in all possible worlds consistent with the prior state that occurred $\omega$ time steps prior. Then the value "uncertain" is assigned to any fluent that has different values in two of the resulting possible worlds.

Amir and Russell (2003) introduced the term **logical filtering**, which is equivalent to a

specific form of belief update consistent with new beliefs based on observations rather than actions. They show that logical filtering can not in general be performed in polynomial time, but give several restrictions under which it can. Their algorithms function in nondeterministic, partially-observable domains described in propositional logic. Logical filtering has been observed (Liu and H. J. Levesque 2005) to be a form of progression, and therefore provides no explanatory power.

Shahaf (2007) described a novel system, *C-Filter*, which performs logical filtering based on a logical circuit representation. During execution, this filtering process represents the value of each state property by a pointer into an and-or tree with leaves in the values of the initial state. The complexity of determining truth therefore grows with time. This does represent the belief state compactly with respect to enumerative state representations, but requires frequent expensive inferences. This system works correctly in partially-observable worlds with deterministic actions, but is not extended to deterministic events or concurrency.

Petrick and Bacchus (2002) described the PKS system, based on traditional planning algorithms, which uses a specialized domain representation. Instead of describing how actions affect the world, they describe what an agent should know after executing the actions. This reduces belief update to an efficient fully observable projection problem, but requires extra human knowledge to be inserted into the domain a priori. It's not clear how much effort is required to generate this type of domain knowledge. In general, it is probably not reasonable to describe all belief changes a system should undergo in complex domains. A formal account of the reasoning involved in such a knowledge-centric theory is given by Patkos and Plexousakis (2009).

Problems analogous to belief update are researched in probabilistic settings under the terminology of Partially Observable Markov Decision Processes and reasoning under uncertainty (e.g., Lovejoy 1991; Kaelbling, Littman, and Cassandra 1998). Similar problems with continuous dynamical systems are dealt with in the state estimation literature (e.g., Åström 1965; Choi, Guzman-Rivera, and Amir 2011). While such techniques are certainly

appropriate for a large class of problems, we seek to limit the size of the body of related work surveyed here, and the techniques studied in these fields are not directly applicable to the problems studied here.

## 2.4 Model-based Diagnosis

The problem of diagnosis focuses on finding what components have failed in a modeled system, given a sequence of observations. This is analogous to the general problem of explaining how states have changed, because every environment can be described as a system, and surprising events are much like faults. In diagnosis, a set of faulty components explain why a system fails, whereas incorrect assumptions can explain why an agent fails to predict surprising events. There is a long tradition of abductive reasoning in the diagnosis community, and recent diagnostic systems solve problems in changing worlds that are isomorphic to a belief change problem.

Historically, diagnosis has been conducted using one of three methods. The first, **rule-based diagnosis**, is the type of reasoning generally conducted by expert systems. To perform this type of reasoning, a large rule base of deductive rules that reason from symptoms to causes is generated in advance, and the diagnostic system computes the closure of these rules with known information (i.e., symptoms) about the system to be diagnosed. This closure includes a set of possible causes, which may or may not be ranked. In the late 1970s and 1980s, several groups attempted to conduct diagnosis without such an exhaustive rule base, using only *models* of the system and its *components*, and *observations* of its behavior. Motivations for this included simplifying the task of building new diagnostic systems (Genesereth 1984), modeling human diagnostic reasoning (Reggia, D. Nau, and P. Wang 1983), criticism of production rules in existing systems (Reggia 1978), applicability to novel symptoms (Davis 1984), reusability of knowledge among related domains (Davis 1984; Genesereth 1984) and ease of maintenance (Davis 1984). Most of these early systems (e.g., De Kleer 1976; Genesereth 1984; Davis 1984; Hamscher and Davis 1984) were later

described as performing **consistency-based diagnosis** (Reiter 1987a), because the diagnoses generated were required to be *consistent* with observations of the system's behavior. In contrast, the term **abductive diagnosis** was coined to refer to diagnosis algorithms for generating diagnoses that *explain* or *imply* all observations made (e.g., Reggia, D. Nau, and P. Wang 1983; Console and Torasso 1990). Consistency-based systems only model the intended behavior of a system, and therefore only output a description of what components are not operating as intended. Abductive diagnosis systems, in contrast, model unintended behavior as well, which is necessary to obtain the explanatory power associated with these systems.

Early diagnosis research emphasized the importance of new representations and the type of information that should be modeled. Genesereth (1984) claimed that his DART Program, an "automated diagnostician" used deep theories of the structure of devices rather than "shallow" rules. Hamscher and Davis (1984; 1984) and De Kleer and Williams (1976; 1987) used models of the structure and behavior of digital circuits to identify faults. Finally, Reggia, Nau, and Wang (1983) used a bipartite graph to describe the relationship between causes and symptoms. These new representations were intended to capture underlying characteristics of the represented domains, rather than human expertise about how to fix them.

Early diagnostic systems did not represent time or changes occurring within the systems modelled; each system was modeled as a static set of relationships. This was a fundamental limitation, as a concept of time is necessary to represent causation. Dvorak and Kuipers (1989) created a system called MIMIC , which performed abductive diagnosis of a running system using qualitative simulation. They were motivated by the need to perform diagnosis of critical systems without shutdown, and the need to model the dynamics of ongoing processes. The MIMIC system was also innovative in its incremental construction of hypotheses over time. If multiple faults occurred during MIMIC's monitoring, they could be added to an existing hypothesis. However, the nature of qualitative simulation restricted MIMIC, in that it was unable to predict the timing of changes. Furthermore,

MIMIC required two simplifying assumptions: first, that only one fault occurred at a time, and second, that faults could not cascade. Therefore, we do not consider this a solution to the hypothesis generation problem.

Friedrich and Lackinger (1991) extended Reiter's diagnostic formalism for reasoning about temporal model-based diagnosis using a notion of failure intervals, motivated by the notion of transient failures. They also provided a method for ranking diagnoses based on Markov Decision Processes. Unlike Dvorak and Kuipers, they provided no system or algorithms for inferring temporal faults.

Cordier and Thiébaux (1994) created the first diagnostic framework for reasoning about the history of a system as a sequence of events. They claimed many benefits from this, including a common representation for exogenous events, expected events, and actions taken for diagnostic purposes; a clear description of a system's history; simple expression of constraints via event preconditions; and the applicability of techniques from automated planning research, such as the expression of preferences. Cordier and Thiébaux introduced several preference criteria over event-based diagnoses, including the "most probable diagnosis" criterion, which prefers diagnoses with the greatest likelihood, where likelihood is the product of event likelihoods; the "shortest diagnosis" criterion, which prefers diagnoses containing fewer events; and the "least redundant diagnosis" criterion, which prefers diagnoses that do not contain (as a subset) smaller diagnoses. This framework introduces a number of useful concepts for reasoning about past events, but unlike DiscoverHistory, is incapable of reasoning about simultaneous events.

Gamper's (1996) PhD thesis describes a system for generating *Abstract Temporal Diagnoses* (ATD) based on a detailed temporal reasoning framework. This system is capable of reasoning about modelled processes that explain observations described using qualitative temporal relationships. In Gamper's work, the term "process" is used to refer to well-understood series of (lengthy) events. These diagnoses construct rich minimal-commitment temporal networks that can be used to identify an underlying cause for a long series of observations taken during many different processes; in demonstration, the system is applied

to diagnose cases of Hepatitis B at a rate much higher than a prior system, HEXAXPERT-1. The ATD system uses only forward-chaining inference, based on sets of pre-generated assumptions, to come up with its theories. Therefore, while the ATD system is capable of generating diagnoses that reason based on events, it requires an explicit framework of possibilities to do so, and cannot generate diagnoses from arbitrary models. This is limiting as it requires generation of problem-specific knowledge, and is thus unsuitable for use as part of an autonomous agent.

Thielscher (1997) provides an alternative theory of *dynamic diagnosis* for diagnosing dynamic systems. Thielscher's theory, based on the Fluent Calculus, accommodates causal information apart from actions. The theory makes no distinction between natural events and exogenous actions for this purpose.

Recent innovations (e.g., S. A. McIlraith 1998; Sohrabi, Baier, and S. McIlraith 2010; Iwan 2001) have extended and applied diagnosis techniques to the problem of agents whose actions may fail. In this work, action histories are found that resolve contradictions by assuming the occurrence of faulty actions and/or missing assumptions about the initial state. Gspandl *et al* (2011) also use the diagnoses found to manage beliefs about the system, which informs some kind of task accomplishment technique. This work describes an agent, IndiGolog, that uses a belief management technique similar to DISCOVERHISTORY search. However, their belief management is based on a notion of "invariants" and "variations" that have no analogue in the environment and transition knowledge used by DISCOVERHISTORY, so it's not clear how they can be compared. Furthermore, IndiGolog is not comparable to DHAGENT, as it does not achieve goals. Instead, it finds actions that execute "some sketchy high-level non-deterministic program" (Giacomo et al. 2009). In summary, work on belief management in IndiGolog is based on similar motivations to our work on DHAGENT and DISCOVERHISTORY, but solves a different set of problems that are non-equivalent.

## 2.5 Planning

Work in planning generally attempts to find optimal (least-cost) plans to achieve a goal as efficiently as possible. The planning problem is closely related to the problem of constructing an explanation about the past: the solution to a planning problem is a sequence of desired occurrences (i.e., actions) which are expected to take place in the future, whereas the solution to an explanation problem is a sequence (i.e., history) of occurrences that have already happened. In this section, we briefly discuss **contingent planning** systems, which approach the problem of deciding how to act in partially observable worlds using computationally intensive methods; as well as two approaches to creating systems that monitor the environment during execution, maintain rational beliefs about the world, and replan as necessary: **real-time control** and **continual planning** systems.

### 2.5.1 Contingent Planning

Contingent planners find tree-structured plans for partially observable domains that branch on observations. These plans can be executed by taking the appropriate branch every time an observation arrives, and can be proved optimal. Because they do not replan (mostly), these systems will only generate a plan for domains where success can be guaranteed. In hazardous domains, where some possible initial state leads to unavoidable failure, no plan will be generated that works for the other states. In order to be correct, these systems must consider every possible initial state. This is highly computationally expensive, and these algorithms do not scale to large domains where many possible initial states exist. Modern contingent planners include POND (Bryce, Kambhampati, and D. Smith 2004), MBP (Bertoli and Cimatti 2002), Contingent-FF (Hoffmann and Brafman 2005), and CLG (Albore, Palacios, and Geffner 2009).

Shani and Brafman (2011) described the SDR planner, which maintains beliefs by reconsidering facts from the past and initial state through a regression process. Unlike other contingent planners, SDR considers only a small, randomly selected subset of the possible

initial states in its planning, reducing computational overhead. Because it approximates the possible states of its (partially observable) environment, SDR must change its beliefs at execution time. SDR senses its environment to ensure that the preconditions of its next action to be executed are not met; when sensed information contradicts its beliefs, it gives them up and constructs a new set of possible initial states, as well as a new plan that succeeds in those states. Because it only performs sensing occasionally, SDR may persist with incorrect beliefs after they could have been rejected. As with other approximation approaches, SDR may generate plans that lead to dead-ends in some domains. SDR attempts to solve a limited version of the goal achievement problem described in Chapter 1, with no support for learning. Its dependence on planning techniques that propositionalize the environment, however, are essential to its strategy; as a result, it runs out of memory on large problems. Furthermore, it does not recognize the concept of deterministic events we use, nor general observations, using instead a sensing action formalism. As such, it constitutes an interesting attempt at a related problem, but is not directly comparable.

### 2.5.2 Real-Time Control and Execution

Existing research on **real-time control and execution** in Artificial Intelligence typically employs a reactive planning foundation, where the agent decides on an action and executes it immediately (Musliner et al. 2008; R. Goldman et al. 2002; Kabanza, Barbeau, and St-Denis 1997). Sometimes, the systems decide on the action to be executed by using planning heuristics. Sometimes, they generate a complete plan, off-line, that achieves the goal, and then execute the plan. These systems can not operate in situations where some possible worlds cause inevitable failure.

The Cooperative Intelligent Real-time Control Architecture (CIRCA) is an autonomous planning and control system that builds and executes safety-preserving plans in the face of unpredictable events (Musliner, Durfee, and Shin 1993). CIRCA includes a Reaction Planner which devises a plan to accomplish mission goals while avoiding or preventing failures. The Reaction Planner takes in a problem description that specifies the initial state

of the world, a set of goal states that the planner attempts to reach, a distinguished failure state that the planner must avoid, and a set of transitions that move the world from one state to another. Unlike most planning systems, CIRCA reasons about uncontrollable sources of changes, such as environmental disturbances, failures, and adversaries. The transition models also include timing characteristics that specify how fast the various transitions can occur. The Reaction Planner uses formal verification techniques to check its plans and ensure that failure is not reachable.

### 2.5.3 Continual Planning

Continual planning refers to the overall process of creating a plan, and then monitoring and maintaining it during execution. This larger problem has been recognized for many years as essential to the overall problem of robot control, and in general to the problem of acting in complex worlds (e.g., Fikes, Hart, and Nilsson 1972; Ambros-Ingerson and Steel 1988; desJardins et al. 1999).

The first continual planning system was the controller of SHAKEY (Fikes, Hart, and Nilsson 1972), a famous robot that performed planning using a system called STRIPS and monitoring using a system called PLANEX. This early system could recognize contradictions between the observed world state and the expected effects described for its actions (i.e., *discrepancies*) and perform replanning in response.

The Integrated Planning, Execution, and Monitoring (IPEM) system (Ambros-Ingerson and Steel 1988) went beyond this by *revising* its plan when discrepancies were recognized; IPEM began executing actions during planning, as soon as an action was added to the plan whose preconditions were already met, and continued to revise the plan as execution occurred, either because preconditions of a later action were simply not yet met, or because a precondition of a later action was altered, as by an exogenous event.

In two 1995 papers, Sooriamurthi and Leake (Sooriamurthi and Leake 1995; Leake 1995) described an architecture for goal-based interactive explanation that is designed to perform explanation to recover from failures during action execution. This architecture

used a case-based component to generate explanations based on the situation encountered, and was demonstrated in a scenario where an action to drive to the airport fails due to a bad battery. However, no serious studies were ever performed with this architecture, and some details of its operation are unclear.

The Continuous Planning and Execution Framework (CPEF) (Myers 1999) was described as a first step in the development of a planning system that employs plan generation, execution, monitoring, and repair capabilities to solve complex tasks in unpredictable and dynamic environments. CPEF assumes that plans are dynamic, that is, they must "evolve" in response to the changes in the environment. CPEF employs HTN planning and plan repair capabilities using SIPE-2 (Wilkins 1988).

CASPER (Knight et al. 2001) uses a continuous planning approach to achieve a high level of responsiveness in dynamic planning situations. Its planner maintains a model of expected future states for monitoring purposes. Based on incoming observations, CASPER's goals can change at any time, triggering a replanning process.

Warfield *et al* (2007) presented a system called RepairSHOP that performs plan repair in response to unexpected changes that prevent execution of the plan. RepairSHOP uses a general and expressive data structure called GoalGraph to represent causal links between actions and nonprimitive task in a hierarchical task network, which permits efficient detection of causal link failures. When failures are detected, only the failed portions of the hierarchical task network must be replaced, resulting in minimal changes to the existing plan at low computational cost.

Göbelbecker et al (2011) describe a continual planner that represents probabilistic information in the description of actions and sensors. This system switches between two planners, one of which, a classical planner, ignores probabilistic information for the sake of efficiency. A second planner, which uses probabilistic information, employs a factored representation of the domain to solve subproblems only when needed. Like other continual planners, this system performs a belief revision step during execution to inform replanning.

Unfortunately, most Continual Planning systems have historically been associated with

robots or specific domains, and have very complex architectures compared to other AI planning systems. Due to this complexity, and the unavailability of most existing systems, there are no comparisons of the performance of these existing systems, and it's unclear how to perform such a comparison. While many of these strategies focus on repairing the plan when observations indicate that it cannot succeed, ignoring other environment changes, they may not go far enough. Explanation allows an agent to infer facts that can not be directly observed, allowing an agent to project problematic situations before problems can be observed directly. Many other papers describe continual planning systems without the use of explanation (Ayan et al. 2007; X. Wang and Chien 1997; Erol, Hendler, and D. S. Nau 1994; Yoon, Fern, and Givan 2007; Kambhampati and A. 1992; Myers 1996; Schoppers 1987; Verfaillie and Schiex 1994; Bernard et al. 1998; Talamadupula et al. 2010; Wyatt et al. 2010; Kraft et al. 2008).

**Continual Planning with Explanation**

A small number of continual planning systems have incorporated explanation in a planning and execution agent for the purpose of improving performance in partially observable domains. As such, these agents are in some ways similar to DHAgent. However, none of them applies to the goal achievement problem motivated and described in Chapter 1, and the agents created are not directly comparable to DHAgent.

In some sense, Dvorak and Kuipers' (1989) MIMIC system, a qualitative diagnostic system, is a continual planning system. Although it is limited to monitoring an existing physical system, and taking actions only for diagnostic purposes, its diagnostics do change its understanding of the world. However, this system cannot be applied to the more general goal achievement problem.

Beetz and McDermott's (1994) XFRM system acts upon complex policy-like plans that express conditionals and loops. While it does not construct these plans automatically, it does modify them automatically in a knowledge-intensive process requiring detailed models of all faults that may be encountered by the agent. Explanations in XFRM are constructed

to explain problems found while projecting the consequences of plans from a plan library. XFRM uses a representation of undesirable states that it uses to test future states found in the projection process. To avoid these forbidden states, XFRM conducts explanations that describe why the states would arise during execution of the plan; this explanation process discovers faults, which are used by a separate component to repair the plan. This capability is not used online, so XFRM is incapable of representing failures due to unexpected events or actions, and does not solve the same problem as DHAgent.

Shanahan and Witkowski controlled a Khepera robot through interleaved planning, planning execution, and perception steps, all implemented in logic (Shanahan 2000; Shanahan and Witkowski 2000). This agent used a description of the world based on Shanahan's extensions to the Event Calculus. Its perception process essentially conducts belief revision using an abductive explanation process that assimilates sensor data. This work is described as preliminary, demonstrated only in an office environment described as static and small. No empirical analysis is made of this agent, but the problem it solves is similar to the goal achievement problem in a static environment.

Eiter *et al* (2007) described an approach to *optimistic planning* , which generates plans that succeed in at least one possible world. They implement a very optimistic type of monitoring that only requires action when it is certain that the plan will leave a space of preferred trajectories described in domain knowledge. Because of this, their optimistic planner may execute multiple actions along trajectories that are very unlikely to succeed, even after detection becomes possible. Eiter et al claim that this is an important step to reduce the cost of monitoring. Explanations in this system only describe a prior time point when a discrepancy first occurred, and not the cause of the discrepancy. Exogenous events are not represented at all.

None of these approaches to continual planning with explanation are entirely satisfying, and there is still a clear need for further research into agents that use explanatory knowledge.

## 2.6 Learning Environment Models

Related work on learning environment models focuses on models of an agent's action. Pasula et al. (2007) describe how an agent can learn models of a world with realistic physics, modelled stochastically and with noise, but fully observable. Zhuo et al (2010) describe LAMP, which infers a sophisticated deterministic action model representation including conditionals and universal quantifiers, from executed plan traces, to reduce software engineering effort. They report it was accurate with ablated information. Mourao et al (2012) employ a two-step learning process to boost the accuracy of models learned from noisy plan traces. Our work differs from these prior studies in its focus on exogenous events.

Several studies address the task of explaining surprises in the current state. SWALE (Leake 1991) uses surprises to guide story understanding and goal-based explanation to achieve understanding goals. Weber, Mateas, and Jhala's (2012) GDA agent learns explanations from expert demonstrations when it detects a surprise, where an explanation predicts a future state obtained by executing an adversary's expected actions. Hiatt, Khemlani, and Trafton (2012) describe an **explanatory reasoning** framework that identifies and explains surprises, where explanations are generated using a cognitively-plausible simulation process. In Ranasinghe and Shen's (2008) Surprise-Based Learning process an agent learns and refines its action models, which can be used to predict state changes and identify when surprises occur. Nguyen and Leong's (2009) Surprise Triggered Adaptive and Reactive (STAR) framework dynamically learns models of its opponents' strategies in response to surprises. None of these systems infers improved environment models, so they are not applicable to the hypothesis-based model learning problem described in Chapter 1.

Several methods exist for learning environment models such as action policies, opponent models, or task decomposition methods for planning (e.g., Zhuo et al. 2009). Techniques also exist for learning other types of models under different assumptions. Inductive Process Modeling (Bridewell et al. 2008) can learn process models from time series data, and predict the trajectories of observable variables. Qualitative Differential Equation Model

Learning (Pang and Coghill 2010) methods can be used to study real-world non-interactive dynamic systems. Reverse Plan Monitoring (Chernova, Crawford, and Veloso 2005) can automatically perform sensor calibration tasks by learning observation models during plan execution. In contrast, we address the problem of inferring models of exogenous events for use in subsequent prediction, planning, and explanation in an execution environment.

In model-free reinforcement learning (Sutton and Barto 1998), agents learn environment models. Our work differs in that it is goal-oriented rather than reward-driven, and thus it allows frequent goal change without requiring substantial re-learning of a policy.

# Chapter 3: Problem Representation

Building on the problem definitions given in chapter 1, we present a syntax and semantics for the data and knowledge used by our algorithms. In particular, this chapter will provide concrete meanings for the terms *environment model, action, event, process, transition function, observation, observation function, goal*, and *transition discontinuity* as used by our algorithms. We will then describe and justify deviations from the languages PDDL, PDDL+, and first-order logic on which it is based.

## 3.1 Environment Model

Each of the four problems addressed in this dissertation requires an environment model $M_\Sigma$. Here, we give a definition for the environment models to be used in this dissertation. This model describes:

1. an ontology of possible world states, incorporating:
   (a) a list of object types, described as symbols;
   (b) a list of functions, described as a function symbol followed by a tuple of argument types, and a value type;
   (c) a list of relations, described by a relation symbol followed by a tuple of argument types;
   (d) a mapping from functions to default values;
2. a set of event models, each describing a class of possible events, including:
   (a) conditions that trigger those events, described as sentences of first-order logic that require relationships and properties between objects in a state, and inequalities between environment properties, and

(b) effects of those events on the world state, described as new values taken by specific environment relationships and properties when an event occurs;

3. a set of action models, each describing a class of possible actions, including:

   (a) conditions under which an action may be caused to occur by a given agent, described as sentences of first-order logic, and

   (b) the effects of those actions on the world state, described as new values taken by specific environment relations and functions when an event occurs;

4. a finite subset of the relation and function symbols that describe observable relations and functions; and

5. a finite subset of the relation and function symbols that describe static relations and functions.

We will now make this more precise using formal definitions; the environment model includes statements based on a restricted first-order logic formalism, as do other languages based on PDDL. The alphabet of this language consists of:

- parentheses ( and ), which are used for grouping;
- **variables**, which are denoted by symbols starting with a question-mark, such as ?X, ?Y, or ?LOC;
- **real and integer constants**, denoted using their base 10 representation, e.g., 1001 or 3.1415
- an enumerated set of **object constant symbols**, which are used to refer to objects in the environment;
- an enumerated set of **type symbols**, which categorize environment objects;
- the type symbols INT and REAL, which refer to integer and real numbers, respectively;

- the binary logical operator symbols AND and OR;
- the unary logical negation operator symbol NOT;
- the equality symbol EQ and inequality symbol NEQ;
- numeric comparison symbols <, >, <=, >=, and =;
- an enumerated set of **relation symbols** or **predicate symbols**, which are used to describe relationships between objects;
- an enumerated set of **function symbols**, which are used to describe functions over objects and numbers;
- an unbounded set of **object symbols**, which consist of all symbols not relegated to one of the above categories.

A **variable definition** assigns a variable a type. If $x$ is a variable, and $T$ is a type, $x$ - $T$ is a variable definition.

A **constant definition** assigns an object constant a type. If $c$ is a constant, and $T$ is a type, $c$ - $T$ is a constant definition.

An environment model describes object constants, relations, and functions using the following:

1. A **type symbol list** describes available types. The set ***Ty***, the set of types in the environment model, consists of these, as well as the standard types INT and REAL.
2. A list of **object constant definitions** specifies the list of object constants ***OC***, and defines the mapping $constantType : \boldsymbol{OC} \rightarrow \boldsymbol{Ty}$. For example, the object constant definition ADAM - PERSON implies both ADAM $\in \boldsymbol{OC}$ and $constantType(\text{ADAM}) \mapsto$ PERSON.
3. a list of **relation definitions**, which name a relation and describe its legal arguments. These are defined as follows: if $P$ is a relation symbol and $\boldsymbol{X}$ is a list of variable definitions, ($P$ $\boldsymbol{X}$) is a relation definition. This defines:

- the list of relation symbols $\boldsymbol{Rs}$,
- a mapping $relationArity : \boldsymbol{Rs} \to \mathbb{I}^+$ ($\mathbb{I}^+$ being the set containing all positive integers), which describes how many objects are referred to by a relation, and
- a mapping $relationArgType : \boldsymbol{Rs} \times \mathbb{I}^+ \to \boldsymbol{Ty}$, which gives the types of relation arguments.

For example, (NEARBY ?P1 - PLACE ?P2 - PLACE) is a relation definition describing how to refer to relationships between two objects of type PLACE. Given a model including this definition, we have NEARBY $\in \boldsymbol{Rs}$, $relationArity(\text{NEARBY}) \mapsto 2$, $relationArgType(\text{NEARBY}, 1) \mapsto$ PLACE, and $relationArgType(\text{NEARBY}, 2) \mapsto$ PLACE.

4. a list of **function definitions**, each of which names a function, describes its legal arguments, and describes its return type. These are defined as follows: if $f$ is a function symbol and $\boldsymbol{X}$ is a list of variable definitions, and $T$ is a type, ($f$ $\boldsymbol{X}$) - $T$ is a relation definition. This defines:
   - the list of function symbols $\boldsymbol{Fs}$,
   - a mapping $functionArity : \boldsymbol{Fs} \to \mathbb{I}^{0+}$ ($\mathbb{I}^{0+}$ being the set containing all positive integers and 0) that describes how many objects are arguments to the function,
   - a mapping $functionArgType : \boldsymbol{Fs} \times \mathbb{I}^{0+} \to \boldsymbol{Ty}$, and
   - a mapping $functionReturnType : \boldsymbol{Fs} \to \boldsymbol{Ty}$ that describes the types of relation arguments.

   For example, (DISTANCE ?P1 - PLACE ?P2 - PLACE) - REAL is a function definition describing how to refer to the real-valued function DISTANCE over two objects of type PLACE. Given a model including this definition, we know the following:

   DISTANCE $\in \boldsymbol{Fs}$, $functionArity(\text{DISTANCE}) \mapsto 2$, $functionArgType(\text{DISTANCE}, 1) \mapsto$ PLACE, $functionArgType(\text{NEARBY}, 2) \mapsto$ PLACE, $functionReturnType(\text{NEARBY}) \mapsto$ REAL.

**Terms** in our language consist of all object constant symbols, all real and integer constants, all variables, all object symbols, and the boolean values TRUE and FALSE. The set of all terms is denoted $\boldsymbol{Te}$. The function $type : \boldsymbol{Te} \rightarrow \boldsymbol{Ty}$ describes the type of a term. We do not define terms to include functions, as is standard in first-order logic; this is consistent with other work in PDDL.

A **relation instance** groups a relation symbol with terms of the appropriate number and type. This has the form $(P\ t_1\ t_2\ \ldots\ t_n)$, where $P$ is a relation symbol, and $t_1, t_2, \ldots, t_n$ are terms. A relation instance is *legal* iff $relationArity(P) \mapsto n \wedge \forall i \in 1 \ldots n\ type(t_i) = relationArgType(P, i)$. We denote the relation type of a relation instance $ri$ using the function $relationType(ri) = P$. We denote its arguments using the function $relationArgs(ri) = t_1, t_2, \ldots, t_n$. For example, given the above relation definition (NEARBY ?P1 - PLACE ?P2 - PLACE) and objects WASHINGTON and BALTIMORE such that $type$(WASHINGTON) $\mapsto$ PLACE and $type$(BALTIMORE) $\mapsto$ PLACE, the relation instance (NEARBY WASHINGTON BALTIMORE) is legal (regardless of whether it is true). For this instance, we have $relationType$((NEARBY WASHINGTON BALTIMORE)) = NEARBY and $relationArgs$((NEARBY WASHINGTON BALTIMORE)) ={WASHINGTON, BALTIMORE}. The relation instances (NEARBY WASHINGTON) and (NEARBY WASHINGTON BALTIMORE ARLINGTON) are not legal, because the arity of the nearby relation does not match the number of arguments. The relation instance (NEARBY WASHINGTON 10.5) is not legal, because the type of 10.5 (REAL) does not match the required argument type, PLACE, of the second argument to the relation NEARBY.

A **function instance** is defined analogously; it has the form $(f\ t_1\ t_2\ \ldots\ t_n)$, where $f$ is a function symbol and $t_1, t_2, \ldots, t_n$ are terms. A function instance is *legal* iff $functionArity(f) \mapsto n \wedge \forall i \in 1 \ldots n\ type(t_i) = functionArgType(f, i)$. We denote the function type of a function instance $fi$ $functionType(fi) = f$, and the arguments $functionArgs(fi) = t_1, t_2, \ldots, t_n$.

We define three types of **literals**: **relation literals**, **function literals**, and **comparison literals**. A **relation literal** is either a relation instance or its negation, e.g., (NOT (NEARBY WASHINGTON BALTIMORE)).

A **function literal** has the form (EQ $fi$ $v$), where $fi$ is a function instance and $v$ is a term called the **value term**. A function literal is *legal* iff $legal(fi) \wedge type(v) = functionReturnType(functionType(fi))$. For example, given the above function definition (DISTANCE ?P1 - PLACE ?P2 - PLACE) - REAL and objects WASHINGTON and BALTIMORE such that $type$(WASHINGTON) $\mapsto$ PLACE and $type$(BALTIMORE) $\mapsto$ PLACE, the function literal (EQ (DISTANCE WASHINGTON BALTIMORE) 10.5) is legal (regardless of whether it is true). The function literals (EQ (DISTANCE WASHINGTON) 10.5) and (EQ (DISTANCE WASHINGTON BALTIMORE ARLINGTON) 10.5) are not legal, because the arity of the distance relation does not match the number of arguments. The function literal (EQ (DISTANCE WASHINGTON SUPERMAN) 10.5) is not legal, because the type of Superman does not match the required argument type, PLACE, of the second argument to the relation DISTANCE. Similarly, (EQ (DISTANCE WASHINGTON BALTIMORE) ARLINGTON) is not legal, as the type of Arlington does not match the required return type, REAL, of the function DISTANCE.

A **comparison literal** has the form ($Comp$ $fi$ $cv$) where $Comp$ is one of the comparison symbols {NEQ, <, >, <=, >=}, $fi$ is a function instance, and $cv$ is a term called the **comparison value term**. A comparison literal is *legal* iff

$$
\begin{aligned}
&legal(fi) \\
&\wedge\ (Comp = \text{NEQ} \\
&\quad \vee\ functionReturnType(functionType(fi)) \in \{\text{REAL, INTEGER}\}).
\end{aligned}
$$

This permits the statement of legal comparisons such as (<= (DISTANCE WASHINGTON BALTIMORE) 100.0). With the inequality symbol, this can also be used for non-numeric functions. Given the function definition (NEARESTCITY ?P - PLACE) - PLACE, the comparison literal (NEQ (NEARESTCITY WASHINGTON) BALTIMORE) is legal. We deem examples of illegal comparison literals to be unnecessary at this point.

We define several functions over literals. For a literal $l$, the function $target(l)$ gives

the function or relation instance the literal refers to. The function $value(l)$ gives the value term of a function or comparison literal; for a relation literal, it gives the value $False$ if the literal is negated, and the value $True$ otherwise. The function $ltype(l)$ gives the function type of a function or comparison literal's function instance, or the relation type of a relation literal's relation instance. Finally, the function $comp(l)$ gives the comparison symbol for a comparison literal. For all other literals, $comp(l) \mapsto \perp$.

We now inductively define the term **formula**.

1. Any literal is a formula.
2. The negation of any formula $\phi$, (NOT $\phi$) is a formula.
3. Given two formulas $\phi$ and $\psi$, (AND $\phi$ $\psi$) and (OR $\phi$ $\psi$) are formulas.
4. Given a variable $x$ and a formula $\phi$ that includes $x$, (FORALL $x$ $\phi$) and (EXISTS $x$ $\phi$) are formulas.

As is standard for PDDL, all variables that are not otherwise bound are assumed to be existentially quantified. Therefore, every formula is also a logical **sentence**. Typical first-order logic semantics apply to all formulas.

A few more necessary definitions:

1. A **ground literal** is a literal containing no variables.
2. The function **vars** returns the unique set containing all variable symbols present in an expression.
3. A **substitution** is a mapping $\{x_1 \mapsto t_1, x_2 \mapsto t_2, ...x_n \mapsto t_n\}$ from variables to terms, typically written as $\theta$.
4. The function **apply** replaces each $x_i$ in an expression with the corresponding term $t_i$.

### 3.1.1 States

A state is a function that assigns a legal value (i.e., a non-variable term) to every legal ground function and ground relation in the environment model. In general, a complete

state may never be known, and tends to represented implicitly rather than explicitly. An environment model does not include states, but specifies a space of states through a set of relation definitions and function definitions.

A **ground relation** has the form $(P\ c_1\ c_2\ \ldots\ c_n)$, where $P$ is a relation symbol, and $c_1, c_2, \ldots, c_n$ are non-variable terms. A ground relation is *legal* iff $relationArity(P) \mapsto n \wedge \forall i \in 1 \ldots n\ type(c_i) = relationArgType(P, i)$. Legal values for a ground relation in a state are true and false.

A **ground function** has the form $(f\ c_1\ c_2\ \ldots\ c_n)$, where $f$ is a function symbol, and $c_1, c_2, \ldots, c_n$ are non-variable terms. A ground function is *legal* iff $functionArity(f) \mapsto n \wedge \forall i \in 1 \ldots n\ type(t_i) = functionArgType(f, i)$. Legal values for a ground function with symbol $f$ in a state are all non-variable terms $c$ such that $type(c) = functionReturnType(f)$.

Formally, we say that $G$ is the set of all ground functions and relations, and $C$ is the set of non-variable terms. Then, a state is a function $state : G \to V$ that assigns each ground function and relation a (legal) value.

States give truth values to formulas; semantically, a non-negated ground relation literal is true in a state iff the state assigns it the value TRUE; a negated ground relation literal is true iff the state assigns it FALSE. Ground function literals are true iff the value term matches the value the state assigns to the function instance. Ground comparison literals are true iff the comparison holds when the function instance is replaced with its state assigned value. For example, if the state assigns the ground function (DISTANCE WASHINGTON BALTIMORE) the value 32.0, the comparison literal (<= (DISTANCE WASHINGTON BALTIMORE) 10.0) is false, but the comparison literal (<= (DISTANCE WASHINGTON BALTIMORE) 100.0) is true. Standard variable substitution semantics and boolean operators govern the truth of larger formulas.

### 3.1.2 Actions

Actions are instantaneous occurrences, intended by an agent, that affect the state of the environment. In general, for an action to occur in a state, that state must trigger no events,

the action must be executable by a specific performer, and that performer must decide to perform the action. The conditions for executability are given by an action's preconditions, which may refer to the performer of the action as well as features of the environment.

The environment model does not include actions directly, but instead describes models of actions. An action model is a tuple

$$\langle name, params, perf, pre, eff \rangle$$

Here, *param* is a list of variable definitions that describe the parameters of the action, representing values are chosen by the agent performing an action; *perf* is a variable representing the agent who performs the action; *pre* is a formula describing the states in which the action is *possible*; and *eff* is a list of **effects** describing how the environment changes in response to the action. We will use a functional notation, i.e., $name(am)$, $param(am)$, $perf(am)$, $pre(am)$ and $eff(am)$ to refer to these aspects of an action model $am$.

We define two types of **effects**: **relation effects** and **function effects**. **Relation effects** are described as either a relation instance or its negation. For instance, (NEARBY WASHINGTON BALTIMORE) and (NOT (NEARBY WASHINGTON BALTIMORE)) are legal relation effects, that express a change to the truth of the relation (NEARBY WASHINGTON BALTIMORE). We say that the relation instance referred to by a relation effect $e$ is its **target** and we denote this with the functional notation $target(e)$. Similarly, a relation effect has a target value, which we will describe as $value(e) \mapsto False$ if the effect is negated, or $value(e) \mapsto True$ otherwise.

A **function effect** has the form (SET $fi$ $v$), where $fi$ is a function instance and $v$ is a term describing the effect's target value. A function effect is legal iff $legal(fi) \wedge type(v) = functionReturnType(functionType(fi))$. For example, (SET (DISTANCE WASHINGTON BALTIMORE) 10.5) would be a legal example of a function effect. We describe the target of a function effect $e$ as $target(e) \mapsto fi$ and its target value as $value(e) \mapsto v$.

For an action model $am$ to be *legal*, the variables referred to by $pre(am)$ must be unambiguously determined by the action model's parameters and performer, and the variables

found in $eff(am)$ must be a subset of those found in the precondition, parameters, and performer. The first condition ensures that actions are unambiguous given an assignment to their parameters by a performer. Formally,

$$\forall s \in S \; \forall \theta Dom(\theta) = vars(params(am)) \cup \{perf(am)\} \implies \\ \exists_{<=1} \theta' s.t. Dom(\theta') = Dom(\theta) \cup vars(pre(am)) \wedge s \models pre(am).$$

The second condition ensures the same for the effects. Formally,

$$vars(eff(am)) \subset vars(params(am)) \cup vars(pre(am)) \cup \{perf(am)\}.$$

An **action** is a tuple $\langle am, \theta \rangle$, where $am$ is an action model, and $\theta$ is a substitution such that $Dom(\theta) \subset vars(params(am)) \cup vars(pre(am)) \cup \{perf(am)\}$. A **ground action** is one in which $\theta$ maps all variables in $params(am)$ to non-variable terms. Only a ground action is **executable**. Semantically, a ground action is **executable** for an actor $\alpha$ in a state $s$ if its preconditions hold, i.e., $s \models apply(\theta, pre(am))$, and the mapping $theta$ gives $\alpha$ as the performer, i.e., $apply(\theta, perf(am)) = \alpha$. The result of performing this ground action is a new state $s'$, which is identical to $s$ except that the value of each ground relation or function in an effect $e \in eff(am)$ is updated to $value(apply(\theta, e))$.

### 3.1.3 Events

Events are instantaneous occurrences, not directly caused by any agent, that affect the state of the environment. An event must occur immediately when its conditions are met, and at no other time.

Like actions, events are not directly included in an environment model. Instead, an environment model includes a set of event models, each of which is a tuple

$$\langle name, params, pre, eff \rangle.$$

The definitions of these are the same as those given for actions. We will use a functional notation, i.e., $name(em)$, $param(em)$, $pre(em)$ and $eff(em)$ to refer to these components of an event model $em$.

For an event model $em$ to be *legal*, the variables referred to by $pre(em)$ must be unambiguously determined by the event model's parameters, and the variables found in $eff(am)$ must be a subset of those found in the precondition and parameters. The first condition ensures that events are unambiguous given an assignment to their parameters. Formally,

$$\forall s \in S \; \forall \theta Dom(\theta) = vars(params(em)) \implies \exists_{<=1} \theta' s.t. Dom(\theta') = Dom(\theta) \cup vars(pre(em)) \wedge s \models pre(em).$$

The second condition ensures the same for the effects. Formally,

$$vars(eff(em)) \subset vars(params(em)) \cup vars(pre(em)).$$

An **event** is a tuple $\langle em, \theta \rangle$, where $em$ is an event model, and $\theta$ is a substitution such that $Dom(\theta) \subset vars(params(em)) \cup vars(pre(em))$. A **ground event** is one in which $\theta$ maps all variables in $params(em)$ to non-variable terms. A ground event is **triggered** in a state $s$ iff $s \models pre(em)$. When one or more events are triggered in a state $s$, their effects are applied simultaneously to arrive at a new state $s'$. The state $s'$ is identical to $s$, except that the value of each ground relation or function in an effect of a triggered event is updated to the value given by that effect. Note that it is *illegal* for an environment model to specify event models such that some state $s$ triggers two events with contradictory effects.

### Why We Represent Deterministic Exogenous Events

Here, we make an aside to describe the reasoning behind representing exogenous events as deterministic. Many other researchers focus on probabilistic exogenous events, and think that deterministic exogenous events appear to be conditional action effects. Modeling events as deterministic is in line with the PDDL+ formalism (Fox and Long 2006) and in keeping with the tradition of natural events (sometimes referred to as natural actions) in the diagnosis and action theory communities (Pinto 1994; Pinto 1997; Reiter 1996). Conditional effects on actions, as described in the planning literature (Pednault 1988), occur deterministically as part of an action. Unlike conditional effects, deterministic events can be triggered by *any* action, or another event; modeling the same information as effects of actions would

require the domain author to consider the effects of all series of events that could ever happen following an action's execution. Therefore, the size of an equivalent domain model using conditional effects rather than events would be exponential in the number of events represented originally.

The advantages of representing exogenous events as deterministic instead of nondeterministic are twofold: (1) we can determine (predictively or after the fact) the exact time when events must occur, reducing the set of potential explanations for a given series of observations and (2) interacting effects can combine without causing an explosion in the number of actions or events considered. Unpredictability in environments under our representation arises only from hidden facts, not a "choice" made by an environment as to whether an event will occur. Representing exogenous events as deterministic is a restrictive assumption that may limit the applicability of our approach; we believe, however, that many interesting and realistic domains can be represented as deterministic, as actions can still cause unpredictable branching of possible futures.

### 3.1.4 Observations and the Observation Function

In this work, an observation is a set of assignments of a legal value to ground functions and ground relations in the environment model. This is similar to the state definition, but observations do not range over all legal ground relations and functions.

To succinctly represent the observation function, we describe a set $\boldsymbol{Ob}$ of *observable* relation and function symbols and a set $\boldsymbol{St}$ of static relation and function symbols. These are subsets of the relation and function symbols specified by the relation and function definitions. For each observable function symbol, a default mapping $default : \boldsymbol{Ob} \mapsto NVT$ gives a default value for that function as a non-variable term of that function's value type. We assume that every non-static observable ground relation in a state valued TRUE, and every non-static observable ground function with a non-default value, is communicated as an assignment in an observation of that state. An initial observation is assumed to contain assignments for all static non-default ground functions and relations.

This representation is in line with our assumptions; there is a simple, information-preserving transformation from any environment model with a noiseless, deterministic observation model $Obs(S)$ to an environment model in which some relationship and property types are always observable and the rest are never observable. This transformation works whether or not observations are given in the language of the environment model:

1. Define a new environment, $\Sigma_n$, containing the relation definitions, function definitions, actions, and events, and processes of the original environment $\Sigma$.

2. For every relation definition $rd$ and function definition $fd$ in $\Sigma$, add to $\Sigma_n$ a new relation definition $rd_{ob}$ or function definition $fd_{ob}$ with a unique relation of function symbol type and identical input and output types.

3. Add a relation definition $rd$ or function definition $fd$ to $\Sigma_n$ that encodes every type of information output by $Obs(s)$ that is not described by relation and/or function definitions in $\Sigma$.

4. Add the new relation and function symbols to the set **Ob**, but none of the originals.

5. Add event models to $\Sigma_n$ that produce the output of $Obs(s)$ and are conditioned on the original properties and relationships.

Step 5 of this transformation effectively compiles the observation function into the transition function.

## 3.2 Modeled Transition Function

The environment model is used by an agent to describe an **internal transition function** $\lambda_\alpha : S \times A \to S$ as specified in the introduction, although it may not be the same transition function as the actual environment. This transition functions is defined by the application of the effects of all events that are triggered in a state $s$, or, if no events are triggered, the application of an agent's action. It is illegal for an agent to perform an action in a

state that triggers one or more events; we assume that if two or more agents attempt to perform actions in the same state, they will be executed in a nondeterministic order, with any triggered events resulting from an earlier action occurring before the next action.

Formally, a ground action $ga$ with model $am$ and substitution $\theta$ is legal for a performer $\alpha$ in a state $s$ when its preconditions are met by $s$, the action substitution $\theta$ maps the action performer to $\alpha$, and no event is triggered in $s$:

$$\begin{aligned} &legal(\langle am, \theta \rangle, \alpha, s) \equiv \\ &\quad s \models apply(\theta, pre(am)) \\ &\quad \wedge apply(\theta, perf(am)) \mapsto \alpha \\ &\quad \wedge \forall \langle em, \theta_{EV} \rangle \in \boldsymbol{E}\ s \not\models apply(\theta_{EV}, pre(em)). \end{aligned}$$

The special action $\emptyset$, which has no effects, is legal in any state that triggers at least one event:

$$legal(\emptyset, s) \equiv \exists \langle em, \theta_{EV} \rangle \in \boldsymbol{E}\ s \models apply(\theta_{EV}, pre(em))$$

We can define the set of effects that must occur in a state $s$ given an action $a$ that is performed (legally) in $s$:

$$\begin{aligned} effectsIn(s, a = \langle am, \theta_{ACT} \rangle) \equiv & apply(\theta_{ACT}, eff(am)) \cup \\ & \{ev = \langle em, \theta_{EV} \rangle \in E | apply(\theta_{EV}, eff(em))\}. \end{aligned}$$

Note that, as it is illegal for an environment model to specify event models such that some state $s$ triggers two events with contradictory effects, the function $effectsIn$ yields a one-to-one mapping from ground functions and relations to values. We now exploit this to define the transition function.

The internal transition function yields a state, which is itself a function over ground relations and functions. This function can now be defined for all legal actions:

$$\lambda_\alpha(s, a)(g) = \begin{cases} effectsIn(s, a)(g) & \text{if } g \in Dom(effectsIn(s, a)) \\ s(g) & \text{otherwise} \end{cases}$$

## 3.3 Goals

Goals are sets of states, which are described by formulas in the environment model language. We say that the agent has achieved its goal $g$ when the current environment state $s$ entails that formula: $s \models g$.

## 3.4 Transition Discontinuities

A transition discontinuity is an interval during which an internal transition function does not hold. These discontinuities are used to describe a belief that the transition function can not accurately describe the observed behavior of the system during some period of time. For more information, see Chapter 8. That is to say, for some enumerated discontinuous state-action pairs $\{\langle s_1, a_1\rangle, \ldots, \langle s_n, a_n\rangle\}$, an internal transition function $\lambda_\alpha$, and a true environment function $\lambda_\Sigma$, a transition discontinuity asserts that $\forall i \in 1 \ldots n \ \lambda_\alpha(s_i, a_i) \neq \lambda_\Sigma(s_i, a_i)$.

## 3.5 Relationship to other Representations

### 3.5.1 Relationship to First-Order Logic

The logical language used here to represent states, goals, and transition discontinuities is a restricted first-order logic formalism. In particular, we do not allow functions to be terms inside of relations or free variables. Nevertheless, we feel that the subset of first-order logic we support is meaningfully large, and slightly larger than much other work in planning, which does not support representation of functions at all.

### 3.5.2 Relationship to PDDL

Adoption of PDDL varies widely among researchers, as there are many different subsets different groups find important. Our language includes some of the more advanced features of PDDL (D. M. McDermott 2000; Fox and Long 2003), including numeric values, typing,

functions, quantifiers, and disjunctive preconditions. In supporting non-numeric functions, we go beyond the de facto PDDL standard; researchers familiar with SAS+ planning, which is based on state variables with non-binary domains, have noted (Helmert 2009) the reduction in representation size and reasoning time this affords; our desire to represent large domains compactly has led us in the same direction.

We do not represent the plan constraints or preferences expressible in PDDL3 (Gerevini and Long 2006), as they do not seem to impact belief management or model learning. We also ignore some other, more common extensions: we choose not to model conditional effects as they are redundant with events, which are more compact, as we argued earlier. Translation from a domain with conditional effects to one with events is trivial (but not vice-versa). Axioms, which represent rules for derivation from basic facts, we also do not represent. In general, these are considered quite useful by the knowledge-based reasoning community at large, but they are not necessary to demonstrate our techniques.

Another divergence from standard PDDL research is in specifying a list of observable predicate and function symbols. One standard way to represent partially observable worlds in planning domains is to supplement them with sensing actions (e.g., Golden and D. Weld 1996; Baral and Son 1997; D. S. Weld, Anderson, and D. E. Smith 1998). This allows small numbers of observable literals to be observed using explicit actions. This is convenient for contingent advance planning, where all possibilities can be thought out in advance, because each sensing action provides one bit of information, causing a single branch in that plan. In contrast, we consider the problem of large domains where a great deal of information is observable, even when most is hidden. Explicit sensing actions would be very tedious for obtaining tens of bits of information repeatedly. Therefore, we make the assumption that an agent receives *all* observable information about the environment periodically as part of its standard interaction, but some properties of the state are not observed.

### 3.5.3 Relationship to PDDL+

PDDL+ (Fox and Long 2006) introduced a concept of events and processes for purposes of temporal planning. While we have taken the concept of events from this work, processes, which are necessary for representation of continuous time, are not described herein, as they are not necessary to defend the claims made in this dissertation. However, this is an important area for future research.

# Chapter 4: DiscoverHistory

As stated in Chapter 1, the hypothesis generation problem is intended to find plausible hypotheses about the history of an environment. A **hypothesis** consists of a set of unproven logical statements about the environment, including statements about the occurrence of events and exogenous actions, and a set of initial state assumptions, and a set of transition discontinuities that describe gaps where the occurrence history does not match the model. A **plausible** hypothesis, taken together with the agent's actions and environment model, must imply the observations received.

A description of the hypothesis generation problem follows:

- ***Given***: a (possibly incomplete) environment model $M_\Sigma$, observations of the environment $\mathbf{o} = [o_0, o_1, \ldots o_m]$, an action history $\mathbf{a_h} = [a_{h,1}, a_{h,2}, \ldots a_{h,n}]$, containing actions taken by $\alpha$, and a total order $\prec$ describing the temporal ordering of actions and observations,
- ***Find***: a set of plausible hypotheses $H_\alpha$ such that $\forall h \in H_\alpha : h \cup \mathbf{a_h} \cup M_\Sigma \models \mathbf{o}$.

To our knowledge, no existing techniques solve this problem. Some techniques exist for solving similar problems, primarily differing in the hypothesis and environment model representation, (e.g., Sohrabi, Baier, and S. McIlraith 2010; Iwan 2001; Gspandl et al. 2011; Shani and Brafman 2011). None of these work on problems with the large state spaces and structured representations we address, nor do they consider transition discontinuities. In this section, we describe the novel algorithm DiscoverHistory, and discuss how it is used to find **plausible explanations**. We will define formal conditions for plausibility, and the relationship between a plausible explanation and plausible hypothesis. We will also describe conditions under which DiscoverHistory search is a complete and sound solution to the hypothesis generation problem. Finally, we will describe DHAgent, and how it uses DiscoverHistory search to manage its beliefs.

## 4.1 Design Decisions

In devising a solution to the explanation generation problem for large environments, we considered several characteristics a good solution should have, which we describe here.

The space of plausible hypotheses is at least as large as the space of possible states an agent might be in, as some hypotheses describe occurrences that converge on an identical state. In general, a partially observable environment will have a state space of size exponential in the number of hidden fluents. Enumerating this space poses both space and computation time problems, even it might lead to more correct behavior; instead, we concentrate on finding a smaller set of hypotheses which are **optimistic**. This means that the assumptions made in those hypotheses are minimal.

As described in Chapter 1, our intended agent must make frequent use of explanation generation in order to manage its beliefs. As a solution to the explanation generation problem is computationally expensive, we would like to take advantage of the frequent calls by devising an **incremental** solution which builds on the results of solutions to related problems.

Finally, we make a commitment to use a structured representation for the environment model $M_\Sigma$, as detailed in Chapter 3. Other work in finding beliefs consistent with histories, such as the work done by Dupin de Saint-Cyr and Lang (2011), tends to rely on a propositional representation. However, the propositional description of an example environment model that describes 100 hidden fluents is of size $2^{100}$. While methods we consider still have scaling problems, even describing an instantiated problem using propositional methods is futile in large environments.

## 4.2 Motivation

Before we get into details of the algorithm, there are two particular sources of motivation for research into explanation generation that shed light on the problem and what it means to solve it. The first is psychological, and describes what a similar process is like in the

human mind. The second comes from a notion of what it means for an agent to be robust.

### 4.2.1 Psychology

We are interested in developing agents that can understand or make sense of what is happening to themselves and around themselves, even when the environment is obscured and constantly changing. By constructing occurrence histories, an agent creates a logical sequence of events to explain to itself why the changes it observes happened, thereby understanding them better. According to psychological research, humans constantly perform an analogous task. This self-explanation has been likened to telling ourselves stories about what's going on. Psychologists refer to this phenomenon as "narrative construction of reality" (Bruner 1991). According to Bruner, "we organize our experience and our memory of human happenings mainly in the form of narrative". According to this research, we as humans are subject to a continuous stream of sensory information, most of which we continually forget; what we remember is our interpretation of what happened, the "events of the day", even though those events exist to us only because we infer them. For example, the concept of a thunderstorm is one that cannot be perceived all at once. Over time, a person perceives flashes of light, interpreted as lightning, hears loud booms, interpreted as thunder, and feels the cascade of rain falling down. Later, he will not remember the exact sensations, but will remember getting caught in a thunderstorm, which is an explanation he constructs about what happened in the world to cause his immediate sensations. The central thesis of this dissertation is that by generating explanations like this one, an agent is able to quickly understand what is happening or has been happening in its environment, and that this ability improves an agent's ability to achieve goals, beyond similar agents with a poorer understanding of their environments.

### 4.2.2 Robustness

Webster defines **robust** (n.d.) as "capable of performing without failure under a wide range of conditions". This quality can be particularly elusive for intelligent agents, for which the

range of conditions they are expected to work in is often much larger and more complicated than is initially understood. In general, as agents address more and more general conditions, they are almost guaranteed to fail, as their programming becomes more complex and handles fewer scenarios. However, it is possible to learn from failures and avoid repeating them. If we consider robustness as a goal that agents should strive to attain, identifying failures and not repeating them should help. Through explanation generation, and agent can identify its failures, and recognize the reason for them. This allows the agent to make new plans that do not repeat the same failures.

## 4.3 Definitions

We introduce the term **occurrence** to refer to an inference or assumption about the environment at a particular time. An **observation occurrence** is a pair of the form $\langle \mathsf{obs}, id \rangle$ where $\mathsf{obs}$ is an observation. An **action occurrence** is a pair of the form $\langle a, id \rangle$ where $a$ is an action. Finally, an **event occurrence** is a pair $\langle e, id \rangle$ where $e$ is an event. In all of the occurrence forms, $id$ is a unique identifying symbol that differentiates multiple occurrences of the same action, event, or observation.

Occasionally, we will also talk about the inferences of literals via occurrences. An **inferred literal occurrence** $ilo = \langle \mathsf{inferred}, l, id \rangle$ indicates that the literal $l$ is inferred to be true.

Two further types of occurrences are more abstract: initial state assumptions and transition discontinuity assumptions. An **initial state assumption** is of the form $\langle \mathsf{initial}, i, v, id \rangle$, and a **transition discontinuity assumption** is of the form $\langle \mathsf{discontinuity}, i, v, id \rangle$ where $i$ is function instance or relation instance and $v$ is a value that can be assigned to that instance. An initial state assumption describes an assignment believed to hold in the initial state; a transition discontinuity assumption describes an assignment that contradicts the transition model, as it is associated with no model of an action or event. These occurrences have no conditions, and cannot be inferred from other occurrences, which is why we refer to them as **assumptions**. Note that all occurrences of exogenous actions are also assumptions,

but not actions present in the action history $\mathbf{a_h}$.

An **explanation** is a description of all occurrences that an agent infers have taken place that can be used for inferring believed world states. Formally, we describe an explanation as a tuple $\chi = \langle O, D, R, C \rangle$, where $O$ is a finite set of occurrences. $D \subset O$ is a finite set of defeasible occurrences, which can be removed from the history. $R$ describes an ordering over $O$ via a set of pairs $\langle o_i, o_j \rangle$ with the semantics $o_i$ occurs earlier than $o_j$. Finally, $C$ is a set of inequalities and equations that describe relationships between terms referred to by occurrences in $O$. We denote the set of all explanations (plausible *and* not plausible) as $\mathbb{X}$.

In order to formalize the order of occurrences in $\chi$, we define the ordering relation $\prec_\chi$, which is the transitive closure of the set of statements $\{o_i \prec_\chi o_j | \langle o_i, o_j \rangle \in R\}$.

In order to simplify reasoning about action and event preconditions, we will split each action and event model into a set of models, based on the disjunctive normal form representation of its preconditions, and replace all universally quantified formulas with expanded conjunctions that include each possible instantiation of the universally quantified variables. For this reason, we assume that numeric variables, which have an infinite range, are never universally quantified, and that the all object symbols in the environment are identified in the initial observation. In this manner, we obtain a set of action and event models that define an equivalent transition function and have preconditions consisting of a conjunction of literals.

Now we define the functions knownbefore, constrainedbefore and knownafter, which are used to determine environment state and detect **inconsistencies** in an explanation. We use $\mathsf{knownbefore}(i, o, v)$ and $\mathsf{knownafter}(i, o, v)$ to refer to the value $v$ of a relation or function instance $i$ immediately before or after occurrence $o \in O$, and we use $\mathsf{constrainedbefore}(c, i, o, v)$ to refer to a constraint on the value of a relation or function instance $i$ immediately before an occurrence $o \in O$.

For an action or event occurrence, $\mathsf{knownbefore}(i, o, v)$ depends on its function and relation literal preconditions. For an action occurrence $o = \langle a = \langle am, \theta \rangle, id \rangle$, the preconditions $pre(o)$ are defined $pre(o) \equiv apply(\theta, pre(am))$. For an event occurrence $o = \langle ev =$

$\langle em, \theta \rangle, id \rangle$, $pre(o) \equiv apply(\theta, pre(am))$. We also define the "precondition" of an inferred literal occurrence $ilo = \langle l, o \rangle$ as $pre(ilo) \equiv \{l\}$. Due to the translation performed earlier, we can treat $pre(o)$ as a set of literals. For all action, event, and inferred literal occurrences, $\mathsf{knownbefore}(i, o, v) \equiv \exists l \in pre(o) : target(l) = i \wedge value(l) = v \wedge comp(l) = \perp$. Comparison literals are handled by the function constrainedbefore. For action, event, and inferred literal occurrences, $\mathsf{constrainedbefore}(c, i, o, v) \equiv \exists l \in pre(o) : target(l) = i \wedge value(l) = v \wedge comp(l) = c$.

We similarly define $eff(o)$ and $\mathsf{knownafter}(i, o, v)$ for action and event occurrences based on effects. For an action occurrence $o = \langle a = \langle am, \theta \rangle, t \rangle$, the effects $eff(o)$ are defined $eff(o) \equiv apply(\theta, eff(am))$. For an event occurrence $o = \langle ev = \langle em, \theta \rangle, t \rangle$, $eff(o) \equiv apply(\theta, eff(am))$. Then, we have $\mathsf{knownafter}(i, o, v) \equiv \exists e \in eff(o) : target(e) = i \wedge value(e) = v$. Inferred literal occurrences do not have effects, so given an inferred literal occurrence $ilo$, we know that $\mathsf{knownafter}(i, ilo, v) \equiv \perp$.

We must also define knownafter and knownbefore for initial state assumptions and transition discontinuity assumptions. For these, it's only important that the mappings hold subsequently. Therefore, for an initial state assumption $isa = \langle i, v, t_0 \rangle$ or a transition discontinuity assumption $tda = \langle i, v, t \rangle$ we have $\mathsf{knownafter}(i', isa, v') \equiv i = i' \wedge v = v'$ and $\mathsf{knownbefore}(i', isa, v') \equiv \perp$.

For an observation occurrence, the closed world assumption applies. Therefore, the relations knownbefore and knownafter are defined for all literals whose types are in $Ob$, as follows:

$$\begin{aligned}\mathsf{knownbefore}(i, \mathsf{obs}, v) \equiv \mathsf{knownafter}(i, \mathsf{obs}, v) \equiv \\ \exists l \in \mathsf{obs} : target(l) = i \wedge value(l) = v) \\ \vee ((\nexists l \in \mathsf{obs} : target(l) = i) \wedge itype(i) \in \boldsymbol{Ob} \wedge v = default(itype(i))).\end{aligned}$$

We say that an occurrence $o$ is relevant to a instance $i$ iff:

$$\begin{aligned}\mathsf{relevant}(i, o) \equiv \exists v\ \mathsf{knownafter}(i, o, v) \vee \exists v\ \mathsf{knownbefore}(i, o, v) \\ \vee \exists c, v\ \mathsf{constrainedbefore}(c, i, o, v).\end{aligned}$$

For purposes of determining action executability, we define the performer of an action

occurrence $perf(\langle\langle am, \theta\rangle, id\rangle) \equiv apply(\theta, perf(am))$.

### 4.3.1 Predicting States Using an Explanation

We now define a mapping from the natural numbers to occurrences in an explanation that describes when those occurrences would happen consistent with a transition function $\lambda_\alpha$. This mapping is described by the function $occurs : \mathbb{X} \times \mathbb{N} \to 2^{\mathbb{A} \cup \mathbb{E} \cup \mathbb{O}}$. The function $occurs(\chi = \langle O, D, R, C\rangle, t)$ gives all occurrences in $O$ happening at a time $t$. It is defined such that the initial observation and all initial state occurrences occur at time 0, and each other occurrence happens at the first natural number after which its predecessors have occurred, subject to the limitation that actions cannot happen until after events have completed. We will now define $occurs(\chi, t)$ inductively.

$$occurs(\chi = \langle O, D, R, C\rangle, 0) \equiv \{\mathsf{obs}_0\} \cup \{\langle \mathsf{initial}, i, v, id\rangle | \langle \mathsf{initial}, i, v, id\rangle \in O\}$$

Based on this, we define the functions $earlierOccs(\chi, n)$ and $nextOccs(\chi, n)$

$$earlierOccs(\chi, n) \equiv \bigcup_{n'=1}^{n-1} occurs(\chi, n')$$

$$nextOccs(\chi = \langle O, D, R, C\rangle, n) = \{o \in O | o \notin earlierOccs(n) \wedge \forall o' \in O : \neg(o' \prec_\chi o) \vee o' \in earlierOccs(n)\}$$

$$occurs(\chi, n) = \begin{cases} nextOccs(\chi, n) & \nexists \langle e, id\rangle : e \in \mathbb{E} \wedge \langle e, id\rangle \in nextOccs(\chi, n) \\ \{o \in nextOccs(\chi, n) \mid o \text{ is not an action occurrence}\} & \text{otherwise} \end{cases}$$

We next define the relationship between an instance $i$ and the last occurrence $o$ in an explanation $\chi$ to affect it before a time $t$:

$$\begin{aligned} mostRecentEffect(i, o, \chi, t) \equiv \exists v, n : \mathsf{knownafter}(i, o, v) \wedge n < t \wedge o \in occurs(\chi, n) \\ \wedge \forall o' \in O : (\forall w : \neg \mathsf{knownafter}(i, o', w)) \vee o = o' \\ \vee (o' \in occurs(\chi, n') \wedge \neg(n < n' < t)) \end{aligned}$$

A **projected state** gives all beliefs that are implied by an explanation $\chi$ just before time $t$. It consists of all literal assignments that are supported by $\chi$. A logical sentence formed by the conjunction of these assignments is true in the possible world described by a plausible explanation at a time $t$, where all assumptions in $\chi$ are correct. The projection

of occurrences in $\chi$ at time $t$ is given by:

$$\mathsf{proj}(\chi, t) = \{(i, v) | \exists o : \mathsf{knownafter}(i, o, v) \wedge mostRecentEffect(i, o, \chi, t)\}.$$

The full projected state, with defaults, is given by:

$$\mathsf{projectedState}(\chi = \langle O, D, R, C\rangle, t)(i) = \begin{cases} v & \text{if } (i, v) \in \mathsf{proj}(\chi, t) \\ default(itype(i)) & \nexists w : (i, w) \in \mathsf{proj}(\chi, t) \end{cases}$$

In plain English, $\mathsf{proj}(\chi = \langle O, D, R, C\rangle, t)$ gives the set of all literals implied to be true after occurrences in $O$ that precede the time $t$, except those literals that are changed by some subsequent occurrence that also precedes $t$.

Projected events are the set of events that must happen at occurrence point $t$ because their preconditions are met in the projected state $\mathsf{proj}(t)$:

$$\mathsf{projectedEvents}(\chi, t) = \{\langle e, t\rangle \mid \mathsf{projectedState}(\chi, t) \vdash pre(e)\}.$$

These definitions support belief management, by giving the beliefs consistent with an explanation, and forward projection (a.k.a. prediction or causal deduction) of explanations from a set of assumptions.

### 4.3.2 Plausible Explanations

In this section, we define what it means for explanation to be plausible. In a plausible explanation, all event occurrences have **proximate causes**. A proximate cause of an event occurrence $\langle e, id\rangle$ is an immediately preceding occurrence $o$ that triggers the event $e$ by causing an effect that establishes its precondition. It must be an occurrence $o$ that satisfies the following condition with respect to an explanation $\chi$:

$$\begin{aligned} &proximateCause(o, o' = \langle e, id\rangle, \chi) \equiv \\ &\quad \exists i, v : \mathsf{knownafter}(i, o, v) \wedge (\mathsf{knownbefore}(i, o', v) \vee \exists c : \mathsf{constrainedbefore}(c, i, o', v)) \\ &\qquad \wedge \nexists o'' \in O : o \prec_\chi o'' \wedge o'' \prec_\chi o' \end{aligned}$$

Because action occurrences do not necessarily occur when their conditions are met, but

rather when a performer intends them to, action occurrences do not have proximate causes. Similarly, observations have no proximate causes, as they are not triggered by occurrences.

An **inconsistency** in an explanation $\chi = \langle O, D, R, C \rangle$ is a tuple $\langle i, o, o', v, c \rangle$ where $o \in O$ is the prior occurrence (written $prior(n) = o$) and $o' \in O$ is the next occurrence (written $next(n) = o'$), $i$ is a relation or function instance given different values by the effects of $o$ and preconditions of $o'$, $v$ is a value referred to by $o'$, and $c \in Comp \cup \{=\}$ describes the relationship between $i$ and $v$ in $o'$. The space of inconsistencies is denoted $\mathbb{N}$.

To help in formalizing the conditions that cause an inconsistency, we first define a relationship between an occurrence $o$ and the most recent prior occurrence $o'$ in $\chi$ to affect a function or relation instance $i$:

$$\begin{aligned} &priorEffect(i, o, o', \chi = \langle O, D, R, C \rangle) \equiv \\ &\qquad o, o' \in O \wedge \exists w : \mathsf{knownbefore}(i, o, w) \\ &\qquad \wedge \forall u : \nexists o'' \in O : \mathsf{knownafter}(i, o'', u) \wedge o \prec_\chi o'' \wedge o'' \prec_\chi o' \end{aligned}$$

An inconsistency describes a contradiction between a pair of occurrences; these contradictions can occur for several different reasons. The following definition formalizes all necessary and sufficient conditions that cause an inconsistency $n$ in an explanation $\chi = \langle O, D, R, C \rangle$, written $n \in \mathsf{Inconsistencies}(\chi)$:

1. An occurrence $o'$ requires that an instance $i$ have a value $w$, but the most recent prior occurrence ($o$) to affect $i$ set its value to some other term $v$.

$$\begin{aligned} &priorEffect(i, o, o', \chi) \wedge \mathsf{knownafter}(i, o, w) \wedge \mathsf{knownbefore}(i, o', v) \wedge v \neq w \\ &\quad \implies \langle i, o, o', v, = \rangle \in \mathsf{Inconsistencies}(\chi) \end{aligned}$$

2. An occurrence $o'$ requires that an instance $i$ have a value that meets some condition, via a comparison literal, and the most recent prior occurrence $o$ to affect $i$ gave it a value that does not meet that condition.

$$\begin{aligned} &priorEffect(i, o, o', \chi) \wedge \mathsf{knownafter}(i, o, w) \wedge \mathsf{constrainedbefore}(c, i, o', v) \\ &\qquad\qquad \wedge \langle w, v \rangle \notin c \wedge \langle c, w, v \rangle \notin C \\ &\quad \implies \langle i, o, o', v, c \rangle \in \mathsf{Inconsistencies}(\chi) \end{aligned}$$

3. An instance $i$ with default value $default(itype(i)) = d$ is required by a precondition of an occurrence $o'$ to have a non-default value $v$, and no preceding occurrence is relevant to $i$.

$$\forall o \in O : \neg priorEffect(i, o, o', \chi) \wedge \mathsf{knownbefore}(i, o', v) \wedge v \neq default(itype(i)) \\ \implies \langle i, \mathsf{obs}_0, o', v, = \rangle \in \mathsf{Inconsistencies}(\chi)$$

4. An occurrence $o'$ requires that an instance $i$ with default value $default(itype(i)) = d$ have a value that meets some condition, via a comparison literal, the default value $d$ does not meet that condition, and no preceding occurrence is relevant to $i$.

$$\forall o \in O : \neg priorEffect(i, o, o', \chi) \wedge \mathsf{constrainedbefore}(c, i, o', v) \\ \wedge default(itype(i)) = d \wedge \langle d, v \rangle \notin c \wedge \langle c, d, v \rangle \notin C \\ \implies \langle i, o, o', v, c \rangle \in \mathsf{Inconsistencies}(\chi)$$

Notes on the definition of inconsistency:

- A pair of occurrences need not be ordered with respect to one another to be inconsistent.
- A literal $l$ with a variable may contradict a legal interpretation of $l$ (i.e., if some other legal interpretation of $l$ contradicts it).

**Example 1**

*Suppose that a rover* R *attempts to move after its wheel has, unobserved, become stuck. Figure 4.1 illustrates part of an inconsistent explanation $\chi$ corresponding to this situation. Four occurrences are ordered by this explanation, including two observations ($o_i$ and $o_{i+3}$), a* NAVIGATE *action ($o_{i+1}$), and a* ROVER-MOVES *event ($o_{i+2}$). These occurrences are totally ordered by $\chi$ such that $o_n \prec_\chi o_{n+1}$. After the* ROVER-MOVES *event, the rover* R *is expected not to be at location* L0*, and to instead be at the location* L1*, but the subsequent observation contradicts this.*

*Exactly one inconsistency exists between the occurrences in this group:* $\langle$(ROVER-AT R),

$o_{i+2}, o_{i+3}$,L0, $=\rangle$. *For reference, Figure 4.2 gives the action and event model behind the occurrences in this example.*

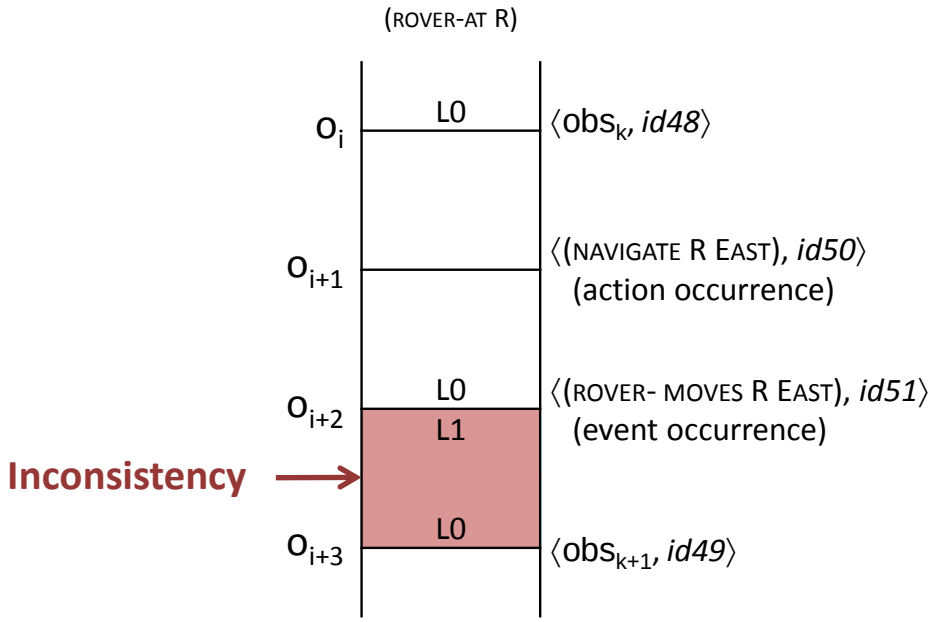


Figure 4.1: Example of an inconsistent explanation, with all occurrences shown totally ordered. Relevant action and event descriptions are given on the right. Values for the knownbefore and knownafter relations are given in the timeline; for example, the value L0 at the top indicates that the relation knownbefore((ROVER-AT R), $o_i$, L0) holds.

Some inconsistencies in an explanation $\chi$ may be **ambiguous**. An ambiguous inconsistency is one that would be resolved in multiple ways by applying different legal substitutions $\theta$ to unbound variables in $\chi$.

An explanation $\chi = \langle O, D, R, C \rangle$ is **strictly plausible** for an explanation generation problem with action history $\mathbf{a_h}$, observations $\boldsymbol{o}$, and ordering $\prec$ iff:

1. There are no inconsistencies in $\chi$.

2. Every event occurrence $\langle e, id \rangle \in O$ has a proximate cause in $O$.

3. Simultaneous preconditions and effects do not contradict. For all $i, o, o', v, w, n$:

$\mathsf{knownafter}(i, o, v) \wedge \mathsf{knownafter}(i, o', w) \wedge o, o' \in occurs(\chi, n) \vdash v = w,$

$\mathsf{knownbefore}(i, o, v) \wedge \mathsf{knownbefore}(i, o', w) \wedge o, o' \in occurs(\chi, n) \vdash v = w,$

4. For all $t \in \mathcal{N}$, the events in $\mathsf{projectedEvents}(t)$ must be in $O$.

5. O contains all observations and the action history: $O \supset \mathbf{a_h} \cup \boldsymbol{o}$.

6. The ordering $\prec_\chi$ is consistent with $\prec$.

7. All actions $a \in O$ not in the action history are exogenous (performed by another agent).

8. All actions $a \in O$ are totally ordered by $\prec_\chi$.

9. No occurrence in $O$ contains a variable.

In summary, an explanation $\chi$ must describe an uninterrupted causal network with no

```
(:ACTION NAVIGATE
 :PARAMETERS (?R - ROVER ?DIR - DIR)
 :PRECONDITION
    (AND (AVAILABLE ?R) (>= (ENERGY ?R) 8) (EQ (ENERGY ?R) ?E)
         (EQ (- ?E 8) ?E2))
 :EFFECT (AND (ATTEMPTING-MOVE ?R ?DIR) (SET (ENERGY ?R) ?E2))
)
(:EVENT ROVER-MOVES
 :PARAMETERS (?R - ROVER)
 :PRECONDITION
    (AND (ATTEMPTING-MOVE ?R ?DIR) (NOT (BLOCKED ?R))
         (EQ (ROVER-AT ?R) ?SRC) (COMPASS-POINTS ?R ?MAGDIR)
         (REAL-DIRECTION ?DIR ?MAGDIR ?TRUEDIR)
         (CAN-TRAVERSE ?R ?SRC ?DEST ?TRUEDIR) (VISIBLE ?SRC ?DEST)
    )
 :EFFECT
    (AND (SET (ROVER-AT ?R) ?DEST)
         (NOT (ATTEMPTING-MOVE ?R ?DIR)))
)
```

Figure 4.2: Excerpt of rover model describing PDDL+ action and event used in timeline.

inconsistencies, and the identities of all participants in all occurrences must be known, for $\chi$ to be strictly plausible.

A relaxed version of this concept is **ambiguous plausibility**. To be ambiguously plausible, explanation $\chi = \langle O, D, R, C \rangle$ must meet all conditions for strict plausibility except for conditions 1 and 9, which are replaced by:

1. There are no *unambiguous* inconsistencies in $\chi$.

9. There is at least one substitution $\theta$ that binds all variables in $O$, meets all constraints in $C$, and resolves all inconsistencies in $\chi$.

We will now show that a strictly plausible explanation with no transition discontinuity assumptions corresponds to a sequence of state transitions given by the environment model's transition function, and that the corresponding states visited would produce the observations received. We do so by showing that (1) each action occurrence point corresponds to a correct state transition, (2) each event occurrence point corresponds to a correct state transition, and (3) the state corresponding to an observation occurrence is consistent with the literals given by that observation.

We claim that for every action occurrence $\langle a, id \rangle$ in the occurrences $O$ of a strictly plausible explanation $\chi = \langle O, D, R, C \rangle$, the internal transition function holds for projected states before and after. Formally, $\langle a, id \rangle \in occurs(n) \implies \lambda_\alpha(\mathsf{projectedState}(t), a) = \mathsf{projectedState}(t{+}1)$. This follows from the definition of the internal transition and projected state functions, both of which keep states static except for effects of new occurrences. We know that for any $n$ such that $\langle a, id \rangle \in occurs(n)$, $\langle a, id \rangle$ must be the only occurrence with effects in $occurs(n)$, because $\chi$ is defined to have no discontinuities, $occurs(n)$ is defined never to contain both action and event occurrences, inferred literal occurrences do not have effects, and action and observation occurrences are totally ordered by conditions 6 and 8 of strict plausibility. Finally, an initial state assumption can not be in $occurs(n)$, because all initial state assumptions are in $occurs(0)$ with $\mathsf{obs}_0$, which is ordered with respect to

every action occurrence. Therefore, only the effects of that action will be different between $\mathsf{projectedState}(t)$ and $\mathsf{projectedState}(t+1)$.

We similarly claim that for every time $t$ such that there is no action occurrence $\langle a, id\rangle$ in an explanation, the internal transition holds for the special action $\emptyset$ and projected states before and after, or, formally: $\forall t \mid \nexists\langle a, id\rangle \in occurs(t) : \lambda_\alpha(\mathsf{projectedState}(t), \emptyset) = \mathsf{projectedState}(t+1)$. We know by the fourth condition of strict plausibility that all events in $\mathsf{projectedEvents}(t)$ must be in $\chi$. We further know that no event occurrence $\langle e, id\rangle$ can be in a strictly plausible explanation without also being in $\mathsf{projectedEvents}(t)$, because, by definition, $\langle e, id\rangle$ would have a precondition $l \in pre(\langle e, id\rangle)$ not met by $\mathsf{projectedState}(t)$. This would mean either that (a) the most recent occurrence $o$ to affect $target(l)$ before $t$ satisfied $\mathsf{knownafter}(target(l), o, w) \wedge w \neq value(l)$, by the first part of the definition of $\mathsf{projectedState}(t)$ or (b) no occurrence prior to $t$ affected $target(l)$ and $value(l) \neq default(itype(v))$, by the second part of the definition of $\mathsf{projectedState}(t)$. Either causes in an inconsistency, which by condition 1, cannot be in a strictly plausible explanation. By the definition of $\lambda_\alpha$, therefore, the following state $\lambda_\alpha(\mathsf{projectedState}(t), \emptyset)$ must be consistent with the effects of the events in $\mathsf{projectedEvents}(t)$. Since no other occurrences can affect the value of $\mathsf{projectedState}(t+1)$ but not $\mathsf{projectedState}(t)$, they will otherwise be the same.

Finally, we claim that for a strictly plausible explanation $\chi = \langle O, D, R, C\rangle$ all observation occurrences $\langle \mathsf{obs}, id\rangle \in \boldsymbol{O}$ contain literals consistent with $\mathsf{projectedState}(t)$. This is the case because if some literal $l \in \mathsf{obs}$ were not consistent, $\langle \mathsf{obs}, id\rangle$ would participate in an inconsistency with the prior affecting occurrence, and no inconsistencies are present in a strictly plausible explanation.

### 4.3.3 Hypotheses

To solve the explanation generation problem requires finding a hypothesis from which the observations can be derived. We've shown that the explanation matches up with a legal sequence of transition for a strictly plausible explanation with no transition discontinuities. The assumptions of such an explanation form such a hypothesis. Using the

assumptions, we can reconstruct the occurrences of a strictly plausible explanation by adding in the action history, and then progressively adding the events based on the value of $\mathsf{projectedEvents}(t)$ for each consecutive occurrence point. Adding in the transition discontinuities requires only that we replace $\lambda_\alpha$ with a modified function, $\lambda'_\alpha$, that delegates to $\lambda_\alpha(\mathsf{projectedState}(t), \emptyset)(i)$ except at each transition discontinuity $\langle \mathsf{discontinuity}, i, v, id \rangle$, where we have $\langle \mathsf{discontinuity}, i, v, id \rangle \in occurs(t) \implies \lambda'_\alpha(\mathsf{projectedState}(t), \emptyset)(i) = v$.

## 4.4 Generating Abductive Explanations

This section describes how to search a space of explanations for plausible explanations. We describe our search technique via the typical breakdown of expansion algorithm, termination condition, order of traversal, and heuristics. DiscoverHistory is an algorithm for node expansion in explanation search. In general, to solve the explanation generation problem, search must continue until at least one plausible explanation is discovered. In this dissertation, we will examine iterative deepening and best-first traversal orders. We leave discussion of specific heuristics for best-first search to later chapters.

Each node in the search is an explanation; internal nodes are implausible explanations. DiscoverHistory generates successor nodes by choosing an inconsistency $i$ in the current explanation $\chi = \langle O, D, R, C \rangle$ and finding a set of refined explanations $X$ based on $\chi$ that do not have that inconsistency.

Algorithm 1 shows the algorithm DiscoverHistory, which inputs an explanation $\chi$, environment model $M_\Sigma$, and refinement operators $\Pi$; and outputs a set of refined explanations $X$. In each invocation of DiscoverHistory, a single inconsistency of an explanation is addressed in all possible ways, or, if no inconsistencies are present, the function FindExtraEvents adds all projected events to the explanation (to satisfy plausibility condition 4). The refinement operators $\Pi$ are a set of functions $\pi : \mathbb{X} \times 2^{M_\Sigma} \times \mathcal{N} \to 2^{\mathbb{X}}$. DiscoverHistory maintains invariants by filtering out child explanations $\chi'$ that do not meet them, on line 10.

### 4.4.1 Checking Invariants

The ordering relation $\prec_\chi$ will become inconsistent if any cycle is added to $R$, causing an occurrence $o$ to precede itself, i.e., $o \prec_\chi o$. As such, the function InvariantsMet checks for this problem, and DiscoverHistory discards any explanation in which such a cycle has been created as impossible.

Constraints on variables in $O$ may be added that cannot be satisfied simultaneously; for example, the constraints $\langle <, ?\text{x}, 2\rangle$ and $\langle >, ?\text{x}, 5\rangle$ will not be satisfied by any mapping for ?x. InvariantsMet also checks for this, and returns FALSE for any explanation containing variables whose values can no longer be satisfied.

### 4.4.2 Refinement Operators

Refinement operators are responsible for finding new "child" explanations that are mostly identical to a "parent" explanation, but avoid an inconsistency present in the parent. In order to have a complete search, all possible ways of avoiding the inconsistency must be found.

Choice of which refinement operators to use is a key difference between various DiscoverHistory search techniques.

**Algorithm 1:** DiscoverHistory.

```
1  Procedure DiscoverHistory (χ, M_Σ, Π)
2  begin
3      if Inconsistencies(χ) = ∅ then
4          χ ← FindExtraEvents (χ)
5          if Inconsistencies(χ) = ∅ then return {χ}
6      i ← Select(Inconsistencies(χ))
7      X ← ∅
8      for π ∈ Π do
9          X ← X ∪ π(χ, M_Σ, i)
10     return {χ' ∈ X | InvariantsMet(χ')}
```

### Adding a Ground Action or Event Occurrence

Let $\chi$ be an explanation with an inconsistency $n = \langle i, o, o', v, c\rangle$, where $v$ is a non-variable term. One way to resolve the inconsistency is to show that some occurrence changed the value of the instance $i$ to $v$ between the preceding event $o$ and the following event $o'$. This occurrence must be an event $o''$ relevant to $i$ such that $o \prec_\chi o'' \prec_\chi o'$.

We define the set of ground actions and events that can resolve an inconsistency $n$ in this way as:

$$groundResolvers(M_\Sigma, n = \langle i, o, o', v, c\rangle) \equiv$$
$$\left\{ \langle am, \theta\rangle \in \mathbb{A} \;\middle|\; \begin{array}{c} params(am) \subset Dom(\theta) \wedge apply(\theta, perf(am)) \neq \alpha \\ \wedge \exists e \in apply(\theta, eff(am)) : target(e) = i \\ \wedge\ value(e) = w \wedge \langle w, v\rangle \in c \end{array} \right\}$$
$$\bigcup \left\{ \langle em, \theta\rangle \in \mathbb{E} \;\middle|\; \begin{array}{c} params(em) \subset Dom(\theta) \\ \wedge \exists e \in apply(\theta, eff(em)) : target(e) = i \\ \wedge\ value(e) = w \wedge \langle w, v\rangle \in c \end{array} \right\}$$

Note that the only actions included in groundResolvers are exogenous. Whenever we add an action, we must order it with respect to all other actions. As existing actions are ordered, this is done by ordering the new action between a pair of consecutive actions. The refinement operator $AddGround : \mathbb{X} \times 2^{M_\Sigma} \times \mathcal{N} \to 2^{\mathbb{X}}$ (Algorithm 2) is responsible for using these to create child explanations.

### Example 2

*Continuing Example 1, the event* ROVER-MOVES *has an effect that causes the rover to enter a different location, so it could be added to resolve the inconsistency* $\langle$(ROVER-AT R), $o_{i+2}, o_{i+3}$,L0, =$\rangle$ *introduced in Example 1. In order for this to work, a new occurrence must be added between* $o_{i+2}$ *and* $o_{i+3}$ *(see Figure 4.3). AddGround creates a child explanation with added occurrence* $o_{new} = \langle$(ROVER-MOVES R WEST), *id*52$\rangle$. *It also adds new ordering constraints* $\langle o_{i+2}, o_{new}\rangle$ *and* $\langle o_{new}, o_{i+3}\rangle$. *There is a new inconsistency in the resulting*

**Algorithm 2:** Adds Ground Occurrences to Resolve an Inconsistency.

```
1  Procedure AddGround(χ = ⟨O, D, R, C⟩, M_Σ, n = ⟨i, o, o′, v, c⟩)
2  begin
3      X ← ∅
4      for ae ∈ groundResolvers(M_Σ, n) do
5          id ← new identifier
6          o″ ← ⟨ae, id⟩
7          O′ ← O ∪ {o″}
8          R′ ← R ∪ {⟨o, o″⟩, ⟨o″, o′⟩}
9          if ae ∈ 𝔸 then // ae is action
10             for {a′, a″ ∈ O | a′, a″ ∈ 𝔸 ∧ a′ ≺_χ a″ ∧ ∄a‴ ∈ 𝔸 : a′ ≺_χ a‴ ≺_χ a″} do
11                 R″ ← R′ ∪ {⟨a′, o″⟩, ⟨o″, a″⟩} // total ordering of actions
12                 X ← X ∪ {⟨O′, D, R″, C⟩}
13         else // ae is event
14             X ← X ∪ {⟨O′, D, R′, C⟩}
15     return X
```

*explanation $\chi_2$ (see Figure 4.3). The new inconsistency occurs because another precondition of the* ROVER-MOVES *event requires that the value of relation literal* (ATTEMPTING-MOVE R WEST) *be* TRUE*, but it is* FALSE *after the most recent relevant occurrence,* $\mathsf{obs}_0$*. This inconsistency,* ⟨(ATTEMPTING-MOVE R WEST)*,* $\mathsf{obs}_0$*,* $o_{new}$*,* TRUE⟩*, could be eliminated by calling* DISCOVERHISTORY *on* $\chi_2$*, typically as a result of further search.*

### Removing an occurrence

Another possible way to resolve an inconsistency $\langle i, o, o', v, c\rangle$, where $o$ and/or $o'$ is defeasible, is to create a new explanation in which either $o$ or $o'$ is removed.

The function $RemoveOcc : \mathbb{X} \times 2^{M_\Sigma} \times \mathcal{N} \rightarrow 2^{\mathbb{X}}$ accomplishes this by creating new explanations without $o$ or $o'$, provided that they are defeasible. However, this is not quite enough, because removing an event occurrence with preconditions that are met (i.e., $\exists t$ : $o' = \langle e, id\rangle \wedge o' \in occurs(t) \wedge proj(t) \models pre(o)$) in its state causes a new problem not marked by an inconsistency. In order to change the explanation such that $o'$ is no longer projected, *RemoveOcc* must ensure that at least one of its preconditions is prevented from being met.

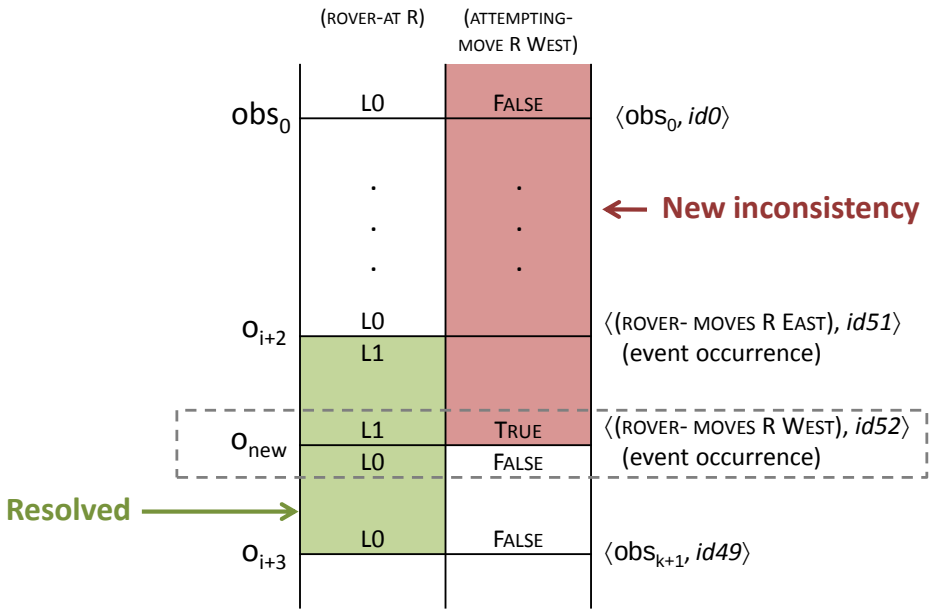


Figure 4.3: Example of adding an occurrence.

In this situation, *RemoveOcc* adds an inferred literal occurrence to the child explanation $ilo = \langle \neg l, \mathsf{occ}(o) \rangle$. This causes an inconsistency to occur for any descendant explanation that implies $o$. See Algorithm 3 for a definition of *RemoveOcc*.

**Example 3**

*RemoveOcc can resolve the inconsistency previously addressed in example 1. To do so, it creates a child explanation* $\chi_3$ *containing all events in* $\chi$ *other than* $o_{i+2}$.

*Note that no inconsistencies are generated by removing it (recall from example 1 that the rover really did not move). Next, the preconditions are examined to look for a precondition which could explain why* $o_{i+2}$ *might not have occurred. Two preconditions on* $o_{i+2}$ *are shown in Figure 4.4:* (EQ (ROVER-AT R) L0) *and* (NOT (PIT-AT L0)). *Two new explanations are created, each negating one precondition:* $\chi_4$ *has the inferred literal occurrence* $\langle \mathsf{inferred}, (\text{PIT-AT L0}), id53 \rangle$, *and* $\chi_5$ *has the inferred literal occurrence* $\langle \mathsf{initial}, (\text{NEQ (ROVER-AT R) L0}), id54 \rangle$. *Figure 4.4 shows the two possible outcomes of removing* $o_{i+2}$ *(without the new inferred literal occurrences).*

**Algorithm 3:** Removes an Occurrence to Resolve an Inconsistency.

**Procedure** RemoveOcc($\chi = \langle O, D, R, C\rangle, M_\Sigma, n = \langle i, o, o', v, c\rangle$)
**begin**
  $X \leftarrow \emptyset$
  **if** $o \in D$ **then**
    $X \leftarrow X \cup \{\langle O/\{o\}, D/\{o\}, R, C\rangle\}$
  **if** $o' \in D$ **then**
    **for** $l \in pre(o')$ **do**
      $id \leftarrow$ new identifier
      $ilo \leftarrow \langle \mathsf{inferred}, \neg l, id\rangle$
      $X \leftarrow X \cup \{\langle (O/\{o'\}) \cup \{ilo\}, D/\{o'\}, R, C\rangle\}$
  **return** $X$

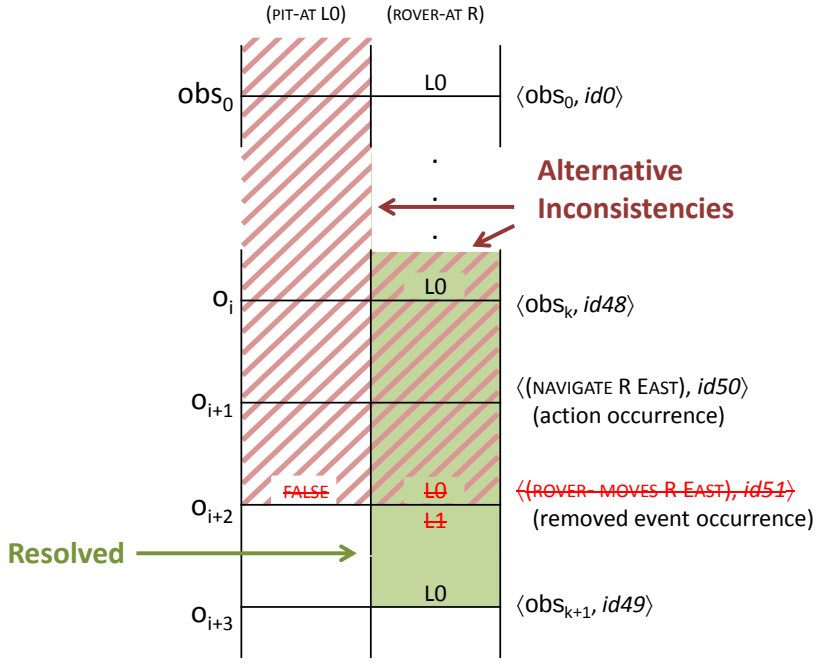


Figure 4.4: Example of removing an occurrence.

### Hypothesizing a Ground Constant Value Mapping in the Initial State

Given an inconsistency $n = \langle i, \mathsf{obs}_0, o, v, = \rangle$, where an occurrence requires an *unaltered ground* instance $i$ with $itype(i) \in \boldsymbol{Ob}$ to have a *non-default constant* value of $v$, a new initial state assumption $isa = \langle i, v, t_0 \rangle$ will resolve $n$.

The function $AssumeInitial : \mathbb{X} \times 2^{M_\Sigma} \times \mathcal{N} \rightarrow 2^{\mathbb{X}}$ (see Algorithm 4) is responsible for producing an explanation with the correct assumption added.

**Algorithm 4:** Creates an initial state assumption to resolve an inconsistency with no prior occurrence.

```
1 Procedure AssumeInitial(χ = ⟨O, D, R, C⟩, M_Σ, n = ⟨i, o, o′, v, c⟩)
2 begin
3     if itype(i) ∈ Ob ∧ o = obs_0 ∧ c ∉ Comp ∧ vars(v) = ∅ then
4         id ← new identifier
5         isa ← ⟨initial, i, v, id⟩
6         return {⟨O ∪ {isa}, D, R ∪ {⟨isa, o⟩ | o ∈ O}, C⟩}
7     else return ∅
```

### Example 4

*The explanation $\chi_4$, introduced in example 3 has the inconsistency* $\langle$(PIT-AT L0), $\mathsf{obs}_0$, $occ_{i+2}$, TRUE, $=\rangle$. *This inconsistency has the characteristics required for hypothesizing an initial state assumption, because a* PIT-AT *literal is not observable. Therefore, the discrepancy can be resolved by adding the initial state assumption* $\langle$initial, (PIT-AT L0), TRUE, $id55\rangle$, *as shown in Figure 4.5.*

### Adding a Minimally Bound Event or Action

This refinement method is much like *AddGround*, but requires adding an occurrence that is ground just far enough to resolve an inconsistency.

To determine the minimal binding necessary, we introduce the most general satisfier function, $MGS(e, c, i, v)$, which returns a tuple $\langle \theta, con \rangle$ such that $\theta$ is a substitution mapping

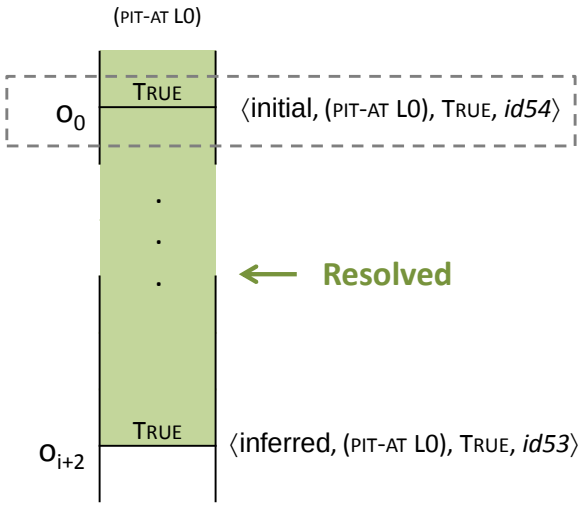


Figure 4.5: Example of hypothesizing an initial value.

from variables to terms, and *con* is a list of constraints on variables in $\theta$, in the form of tuples $\langle c, u, w \rangle$ where $c \in Comp$ is a comparator, $u$ is a variable, and $w$ is a term. We define MGS by example:

$$
\begin{aligned}
&MGS((\text{SET } (\text{ENERGY } ?\text{R}) \ ?\text{V}), =, (\text{ENERGY } \text{R1}), 30) \\
&\qquad = \langle \{?\text{R} \mapsto \text{R1}, ?\text{V} \mapsto 30\}, \emptyset \rangle \\
&MGS((\text{SET } (\text{ENERGY } ?\text{R}) \ ?\text{V}), =, (\text{ENERGY } \text{R1}), ?\text{E96}) \\
&\qquad = \langle \{?\text{R} \mapsto \text{R1}, ?\text{V} \mapsto ?\text{V96}\}, \emptyset \rangle \\
&MGS((\text{SET } (\text{ENERGY } ?\text{R}) \ ?\text{V}), <=, (\text{ENERGY } \text{R1}), 30) \\
&\qquad = \langle \{?\text{R} \mapsto \text{R1}, ?\text{V} \mapsto ?\text{V36}\}, \{\langle <=, ?\text{V36}, 30 \rangle\} \rangle \\
&MGS((\text{SET } (\text{ENERGY } ?\text{R}) \ ?\text{V}), <=, (\text{ENERGY } \text{R1}), ?\text{E96}) \\
&\qquad = \langle \{?\text{R} \mapsto \text{R1}, ?\text{V} \mapsto ?\text{E96}\}, \{\langle <=, ?\text{V36}, ?\text{E96} \rangle\} \rangle \\
&MGS((\text{SET } (\text{ENERGY } ?\text{R}) \ 40), <=, (\text{ENERGY } \text{R1}), ?\text{E96}) \\
&\qquad = \langle \{?\text{R} \mapsto \text{R1}, ?\text{E96} \mapsto ?\text{E153}\}, \{\langle >=, ?\text{E153}, 40 \rangle\} \rangle \\
&MGS((\text{SET } (\text{ENERGY } ?\text{R}) \ 40), <=, (\text{ENERGY } \text{R1}), 30) = \bot
\end{aligned}
$$

We define the set of minimally bound actions and events that can resolve an inconsistency $n$ as:

$$\begin{aligned} &minimalResolvers(M_\Sigma, n = \langle i, o, o', v, c\rangle) \equiv \\ &\qquad \{\langle\langle am, \theta\rangle \in \mathbb{A}, con\rangle | \exists e \in eff(am) : MGS(e, c, i, v) = \langle\theta, con\rangle\} \\ &\qquad \cup \{\langle\langle em, \theta\rangle \in \mathbb{E}, con\rangle | \exists e \in eff(em) : MGS(e, c, i, v) = \langle\theta, con\rangle\} \end{aligned}$$

We now define the *AddMinimal* function (Algorithm 5), which will resolve inconsistencies using minimally bound occurrences.

**Algorithm 5:** Adds Minimally Bound Occurrences to Resolve an Inconsistency.

```
1  Procedure AddMinimal(χ = ⟨O, D, R, C⟩, M_Σ, n = ⟨i, o, o', v, c⟩)
2  begin
3      X ← ∅
4      for ⟨ae, con⟩ ∈ minimalResolvers(M_Σ, n) do
5          id ← new identifier
6          o'' ← ⟨ae, id⟩
7          O' ← O ∪ {o''}
8          R' ← R ∪ {⟨o, o''⟩, ⟨o'', o'⟩}
9          C' ← C ∪ con
10         if ae ∈ 𝔸 then // ae is an action
11             for {a', a'' ∈ O | a', a'' ∈ 𝔸 ∧ a' ≺_χ a'' ∧ ∄a''' ∈ 𝔸 : a' ≺_χ a''' ≺_χ a''} do
12                 R'' ← R' ∪ {⟨a', o''⟩, ⟨o'', a''⟩} // total order of actions
13                 if perf(o'') is a variable then
14                     C' ← C' ∪ {⟨≠, perf(o''), α⟩} // ensure not α action
15                     X ← X ∪ {⟨O', D, R'', C'⟩}
16                 else if perf(o'') ≠ α then
17                     X ← X ∪ {⟨O', D, R'', C'⟩}
18         else X ← X ∪ {⟨O', D, R', C'⟩} // ae is an event
19     return X
```

### Unifying Inconsistent Occurrences

In some cases, an inconsistency $n = \langle i, o, o', v, c\rangle$ exists, but there is some substitution $\theta$ of variables in $o$ and $o'$ such that $\mathsf{knownafter}(i, o, apply(\theta, v))$. In this case, the function $UnifyInconsistent$ applies the bindings to the explanation, and the occurrences "come to agreement", resolving the inconsistency. See Algorithm 6.

**Algorithm 6:** Unifies Inconsistent Occurrences to Resolve an Inconsistency.

```
1 Procedure UnifyInconsistent(χ = ⟨O, D, R, C⟩, M_Σ, n = ⟨i, o, o', v, c⟩)
2 begin
3     X ← ∅
4     for e ∈ eff(o) : MGS(e, c, i, v) ≠ ⊥ do
5         ⟨θ, con⟩ ← MGS(e, c, i, v)
6         χ' ← ⟨apply(θ, O), apply(θ, D), apply(θ, R), apply(θ, C) ∪ con⟩
7         X ← X ∪ {χ'}
8     return X
```

### Set Ordering of Inconsistent Occurrences

In some cases, an inconsistency $n = \langle i, o, o', v, c\rangle$ exists when $o$ and $o'$ are unordered. If the inconsistency $o'$ came before $o$, the conditions of $n$ would not be met. In this case, the function $OrderInconsistent$ adds an ordering constraint to a child explanation to avoid the inconsistency. See Algorithm 7.

### Reorder Third Occurrence

In some cases, an inconsistency $n = \langle i, o, o', v, c\rangle$ exists between occurrences $o$ and $o'$, but a third occurrence $o''$, already in the explanation, would come between them and resolve the inconsistency in some plausible explanation. This is much like adding an occurrence to come in between two inconsistent occurrences, but in this case the third occurrence has already been created.

**Algorithm 7:** Sets Ordering of Inconsistent Occurrences to Resolve an Inconsistency

```
1  Procedure OrderInconsistent(χ = ⟨O, D, R, C⟩, M_Σ, n = ⟨i, o, o′, v, c⟩)
2  begin
3      X ← ∅
4      if ¬(o ≺_χ o′) then
5          X ← X ∪ {⟨O, D, R ∪ {⟨o′, o⟩}, C⟩}
6      return X
```

To determine this, the function *ReorderThird* must find an occurrence $o''$ in the explanation that can be bound to satisfy the requirements of $o'$, and can be reordered to come in between $o$ and $o'$. See Algorithm 8

**Algorithm 8:** Reorders Third Occurrence in Between Inconsistent Occurrences to Resolve an Inconsistency

```
1  Procedure ReorderThird(χ = ⟨O, D, R, C⟩, M_Σ, n = ⟨i, o, o′, v, c⟩)
2  begin
3      X ← ∅
4      for o″ ∈ O do
5          if ¬(o″ ≺_χ o) ∧ ¬(o′ ≺_χ o″) then
6              for e ∈ eff(o″) : MGS(e, c, i, v) ≠ ⊥ do
7                  ⟨θ, con⟩ ← MGS(e, c, i, v)
8                  R′ ← apply(θ, R) ∪ {⟨o, o″⟩, ⟨o″, o′⟩}
9                  C′ ← apply(θ, C) ∪ con
10                 χ′ ← ⟨apply(θ, O), apply(θ, D), R′, C′⟩
11                 X ← X ∪ {χ′}
12     return X
```

Note that in some cases, the outside occurrence $o''$ does not actually need reordering to come in between; the function is named for the more general case in which it will reorder $o''$ if necessary.

**Introduce Transition Discontinuity**

Any inconsistency can be resolved through the addition of a transition discontinuity assumption. Transition discontinuity assumptions are extremely powerful because they allow an occurrence with an arbitrary effect to be placed anywhere in the explanation. It's also dangerous, because every discontinuity makes the explanation less accurate.

The refinement method *IntroduceDiscontinuity* is responsible for creating an appropriate discontinuity and ordering it with respect to the inconsistent occurrences. See Algorithm 9.

**Algorithm 9:** Introduces Transition Discontinuity

```
1  Procedure IntroduceDiscontinuity(χ = ⟨O, D, R, C⟩, M_Σ, n = ⟨i, o, o′, v, c⟩)
2  begin
3      id ← new identifier
4      C′ ← C
5      if c ∈ Comp then
6          var ← new variable
7          type(var) ← itype(i)
8          C′ ← C ∪ {⟨c, var, v⟩}
9      tda ← ⟨discontinuity, i, v, id⟩
10     return {⟨O ∪ {tda}, D, R ∪ {⟨o, tda⟩, ⟨tda, o′⟩}, C′⟩}
```

### 4.4.3 The FindExtraEvents Subroutine

Requirement 4 of a plausible explanation (see Section 4.3.2) states that all events that would be projected to occur by the explanation must occur. This ensures that deterministic events that must happen are inferred, even when the observations provide no evidence of them. The function FindExtraEvents adds these to an explanation, by examining the difference between $occurs(t)$, the events in an explanation that happen at time $t$, and $\mathsf{projectedEvents}(t)$, the set of events whose preconditions are met at time $t$. New events added by FindExtraEvents are ordered with respect to occurrences at $t-1$ to ensure

that each has a proximate cause. To support ambiguous explanations, no event is added when an existing partially unbound event could be further bound to arrive at it. We detail this in Algorithm 10.

As some events predicted by the current assumptions might contradict available observations, and DiscoverHistory introduces no alternate search path when extra events are added, all events added by FindExtraEvents are made *defeasible* by adding them to $D$. This allows the event to be later removed, with the caveat that its removal must be justified (see description of the *RemoveOcc* operator).

**Algorithm 10:** The FindExtraEvents Subroutine

```
1  Procedure FindExtraEvents (χ = ⟨O, D, R, C⟩)
2  begin
3      O' ← O
4      D' ← D
5      R' ← R
6      n ← 0
7      while |occurs(χ, n)| > 0 do
8          n ← n + 1
9          χ' ← ⟨O', D', R', C⟩
10         for e = ⟨em, θ⟩ ∈ {e|e ∈ projectedEvents(χ', n) ∧ e is an event occurrence}
            do
11             if ∄e' = ⟨em, θ'⟩ ∈ occurs(χ', n) : θ' ⊂ θ then
12                 O' ← O' ∪ {e}
13                 D' ← D' ∪ {e}
14                 R' ← R' ∪ {⟨o, e⟩|o ∈ occurs(χ', n − 1) ∧ ∃p ∈ pre(e) : eff(o) ⊨ p}
15     return ⟨O', D', R', C⟩
```

## 4.5 DiscoverHistory Search Properties

### 4.5.1 Soundness of DiscoverHistory Search

To be sound, we must show that a solution to the explanation generation problem based on DiscoverHistory only returns correct results. We showed in sections 4.3.2 and 4.3.3 that a strictly plausible explanation corresponds to a hypothesis that meets the definition found in the explanation generation problem. We can therefore construct a sound solution by specifying that a search returns all strictly plausible explanations found, and subsequently transforms them to hypotheses from which the observations can be unambiguously derived.

For each ambiguously plausible explanation $\chi = \langle O, D, R, C \rangle$, there are one or more substitutions $\theta$ that bind all variables to a legal value and resolves all inconsistencies without adding ordering constraints, by condition 9. Finding a strictly plausible explanation from an ambiguously plausible explanation $\chi$ is possible by performing repeated applications of the refinement operators $UnifyInconsistent$ and $ReorderThird$. Therefore, we will use an ambiguously plausible explanation to represent the set of strictly plausible explanations reachable by applying $UnifyInconsistent$ and $ReorderThird$ until no variables remain, as well as the hypotheses found by transforming those strictly plausible explanations. Therefore, by representational equivalence, a solution to the explanation generation problem is also sound if it is defined as returning all ambiguously plausible explanations found.

### 4.5.2 Completeness of DiscoverHistory Search

For explanation generation, we define completeness to mean that if at least one hypothesis exists that could be returned for an explanation generation problem, at least one hypothesis is returned that is correct.

For the most general case, we can construct a DiscoverHistory search technique whose refinement methods include $OrderInconsistent$ and $IntroduceDiscontinuity$. Any

inconsistency can be resolved by this refinement method without introducing new inconsistencies. To do so, we first construct the **base explanation**, an explanation with inconsistencies corresponding to what is unknown in the initial problem. The base explanation is denoted $\chi_0 = (\langle \boldsymbol{o} \cup \mathbf{a_h}, \emptyset, \{\langle occ_i, occ_j \rangle | occ_i, occ_j \in \boldsymbol{o} \cup \mathbf{a_h} \wedge occ_i \prec occ_j\}, \emptyset \rangle)$. The base explanation with events is $\chi_1 = \textsc{FindExtraEvents}\ (\chi_0)$. Given $|\mathsf{Inconsistencies}(\chi_1)| = n$ and $|O| = s$, a strictly plausible explanation $\chi^*$ will be found within $s \times (n+1)$ applications of DiscoverHistory to child explanations generated.

To demonstrate why, first note that all inconsistencies in $\mathsf{Inconsistencies}(\chi_1)$ will be of the form $n = \langle i, o, o', v, c \rangle$ where $o'$ is an observation. Event preconditions will not be inconsistent with other occurrences, because they were generated based on the projectedState function, which agrees with prior occurrences by definition. Action preconditions are based only on static, observable functions and relations, and are not executable when their preconditions are not met. Therefore, action preconditions are never inconsistent with earlier occurrences. Next, we note that *IntroduceDiscontinuity* can be applied to avoid any inconsistency in $\mathsf{Inconsistencies}(\chi_1)$, introducing a new transition discontinuity assumption $tda$ and explanation $\chi_2$. By definition, $tda$ affects only one instance, so there can be at most one new inconsistency in $\mathsf{Inconsistencies}(\chi_2)$ between any other occurrence $o \in O$ and $tda = \langle \mathsf{discontinuity}, i, v, id \rangle$, for a total of $s$ inconsistencies.

Each new inconsistency can be resolved in one refinement, creating no other new inconsistencies. No event after $o'$ will be inconsistent with $tda$, because the existence of $n$ indicates that $o'$ is relevant to $i$. Therefore, any resulting inconsistency $n_1 = \langle i, tda, o'', w, c$ can be resolved by the *OrderInconsistent* method, which will create a new explanation $\chi_3$ with $o'' \prec tda$. The explanation $\chi_3$ will have no new inconsistencies, because *OrderInconsistent* creates none. A maximum of $s$ calls to *OrderInconsistent* are therefore necessary for each inconsistency $n \in \mathsf{Inconsistencies}(\chi_1)$.

*IntroduceDiscontinuity* will be called at most $n$ times, introducing at most $s \times n$ new inconsistencies. *OrderInconsistent* is called once for each new inconsistency. In total, DiscoverHistory must be called a maximum of $s \times (n+1)$ times, resulting in an explanation

with no inconsistencies, $\chi^*$.

The plausibility conditions are met by $\chi^*$ for the following reasons:

1. By the prior argument, there are no inconsistencies remaining in $\chi^*$.

2. All events in $\chi_1$ are added by FINDEXTRAEVENTS. As FINDEXTRAEVENTS adds only events in PROJECTEDEVENTS($t$), these are all events with proximate causes. No event occurrences are added or removed by *IntroduceDiscontinuity* or *OrderInconsistent*, and no ordering constraints $\langle o, e\rangle$ are added where $e$ is an event, so all events in $\chi^*$ come from $\chi_1$, and each still has a proximate cause in $\chi^*$.

3. The explanation $\chi_1$ contains only actions and events that would occur in a possible world according to $M_\Sigma$. We know that the effects of these actions and events will not contradict, because otherwise the environment model $M_\Sigma$ would be illegal (as described in Section 3.1.3). We further know that the preconditions of events can not disagree, because the preconditions of all simultaneous events generated by FINDEXTRAEVENTS are implied by a single state which maps each instance to a single value. All discontinuities created by *IntroduceDiscontinuity* set the value of an instance $i$ to the value required by the next action or observation that occurs later. Two simultaneous discontinuities $tda_1, tda_2 \in occurs(n)$ for an instance $i$ would have the same next action or observation, so by definition, $\mathsf{knownafter}(i, tda_1, v) \equiv \mathsf{knownafter}(i, tda_2, v)$.

4. All events in $\mathsf{projectedEvents}(\chi_1, t)$ are in $\chi_1$ due to the application of FINDEXTRAEVENTS. The same events are in $\chi^*$. The discontinuities in $\chi^*$ are order constrained such that they do not affect the state prior to occurrences other than the following observation. Therefore, no new events are in $\mathsf{projectedEvents}(\chi^*, t)$ that are not also in $\mathsf{projectedEvents}(\chi_1, t)$.

5. The base explanation $\chi_0$ is defined as including the occurrences in $\boldsymbol{o} \cup \mathbf{a_h}$. Neither FINDEXTRAEVENTS, *IntroduceDiscontinuity*, nor *OrderInconsistent* removes any

occurrences from an explanation, so the occurrences of $\chi_0$ are a subset of the occurrences in $\chi^*$.

6. The base explanation $\chi_0 = \langle O, D, R, C \rangle$ is defined as including in $R$ all constraints in $\prec$. Therefore, as neither FindExtraEvents nor any refinement method can remove constraints from an explanation, $\chi^*$ includes the same constraints. Therefore, $\prec_{\chi^*}$ is consistent with $\prec$.

7. The explanation $\chi_1$ includes no actions aside from the action history. The refinement methods *IntroduceDiscontinuity* and *OrderInconsistent* cannot add actions. Therefore, the explanation $\chi^*$ contains no actions that are not in the action history.

8. The explanation $\chi_1$ contains only actions in $\mathbf{a_h}$, which are totally ordered by $\prec$. Neither *IntroduceDiscontinuity* nor *OrderInconsistent* adds actions. Therefore, all actions in $\chi_*$ are totally ordered by $\prec^*_\chi$.

9. The base explanation $\chi_0$ contains no variables, and neither FindExtraEvents nor *OrderInconsistent* adds variables. *IntroduceDiscontinuity* adds variables only to resolve a discontinuity with an inequality, which will not occur, as all inconsistencies in $\mathsf{Inconsistencies}(\chi_1)$ involve the preconditions of observations.

Any DiscoverHistory search that includes the refinement methods *OrderInconsistent* and *IntroduceDiscontinuity*, searches all nodes less than or equal to $n$ distance away, and returns all strictly plausible explanations, will return $\chi^*$. Therefore, such a search is a sound solution.

A solution found in this manner, without finding reasons within the model for surprising observations, may be of low quality, as it does not infer any action or initial state assumptions that might help with prediction. However, it does at least find the hidden impact of predictable events. We refer to $\chi^*$ as the **incurious explanation**, as it does not posit reasons for expectation failures. However, in some cases, when no exogenous events or hidden initial state features have caused an observable effect, the incurious explanation

is reasonable; in these cases, $\chi_1 = \chi^*$. In these cases, search will often be considered unnecessary.

It is also interesting to consider a related property, which we will call **dynamic-aware completeness**. An explanation generation solution is dynamic-aware complete when, for every explanation generation problem with at least one plausible hypothesis *without discontinuities*, the search returns at least one plausible hypothesis *without discontinuities*. The incurious explanation is not considered a solution under this definition, as it will in all interesting cases contain discontinuities. Unfortunately, a proof of this is beyond the scope of this work. While experimental results have shown that it is often possible, dynamic-aware completeness is not yet proven. We are also interested in the question of whether the order in which inconsistencies are resolved affects the reachability of individual explanations. Our intuition is that it does not, at least for certain refinement sets which encompass all possible one-step changes. However, this remains unproven as well.

## 4.6 DHAgent

DHAgent is an important contribution of this dissertation, which demonstrates how to use managed knowledge together with planning to improve execution performance of an agent. Algorithm 11 shows the DHAgent top-level algorithm, which inputs a list of goals to accomplish, then begins a loop of observing (line 11) and acting in (line 10) an environment. At any given time, DHAgent has a single maintained explanation, $\langle O, D, R, C \rangle$, that is modified by adding new actions it takes and observations it receives, as well as all projectedEvents that would be expected based on its current assumptions (lines 12-14). To manage its beliefs about the environment, DHAgent computes the newest projected state in every loop iteration (line 23) based on its current explanation. Whenever this explanation is found to have inconsistencies, a DiscoverHistory search is run (line 17) to find descendant explanations that are plausible. Whenever the explanation is modified, the plan is also recreated (line 24), as its assumptions have been violated. In order to maintain execution guarantees, the

search termination condition may sometimes cause the search to terminate without finding a solution. In recognition of this, on line 22, the algorithm reboots its explanation whenever search returns no results.

**Algorithm 11:** DHAgent

```
1  Procedure DHAgent(M_Σ, g)
2  begin
3      obs_0 ← ReceiveObservation()
4      O ← {obs_0}  D ← ∅  R ← ∅  C ← ∅
5      S ←projectedState(⟨O, D, R, C⟩, ∞)
6      π ← Plan(S, g)
7      i ← 1
8      while π ≠ ∅ do
9          a_i ←Pop(π)
10         ExecuteAction(a_i)
11         obs_i ←ReceiveObservation()
12         O ← O ∪ {a_i, obs_i}
13         R ← R ∪ {⟨obs_{i−1}, a_i⟩, ⟨a_i, obs_i⟩}
14         ⟨O, D, R, C⟩ ←FindExtraEvents(⟨O, D, R, C⟩)
15         if Inconsistencies(⟨O, D, R, C⟩) ≠ ∅ then
16             k ← 0  X_new ← ∅
17             X_new ← DiscoverHistory-Search()
18             if X_new ≠ ∅ then
19                 ⟨O, D, R, C⟩ ←First(X_new)
20                 D ← O \ ({obs_j | j ∈ 0 . . . i} ∪ {a_j | j ∈ 1 . . . i})
21             else
22                 ⟨O, D, R, C⟩ ← ({obs_i}, ∅, ∅, ∅)
23             S ←projectedState(⟨O, D, R, C⟩, ∞)
24             π ←Plan(S, g)
25         i ← i + 1
```

DHAgent is designed to start with a simple explanation, based on the closed world assumption, and add complexity as observations indicate its current assumptions are incorrect. This design is intended to produce reactive, resilient behavior in domains with many possible but unlikely worlds. In such domains, the high number of hidden facts causes

planning for all possible worlds to be intractable. For example, although sand pits may be ubiquitous on Mars, a Mars rover could not reason about all possible such pits. Indeed, by assuming sand pits are everywhere it might remain stuck in one place, never moving for fear of falling in. Instead, DHAgent would initially assume that the ground is safe, but be prepared to revise its assumptions when new evidence is observed.

The incremental nature of DHAGENT is implemented by using its current explanation as a basis for a new search (see line 17). So as to ensure that DHAGENT is able to recover from past mistakes, all occurrences found in a search are made defeasible before the next search starts (see line 20). Therefore, when DHAGENT has an existing explanation $\chi$ with correct assumptions based on prior searches, a new search will have a head start and complete faster. However, when the existing explanation $\chi$ has incorrect assumptions, the new search can retract them.

We have purposely left out a description of a specific DISCOVERHISTORY search used by DHAGENT; several will be discussed in the coming chapters.

# Chapter 5: Explanation-Based Belief Management

Having described DiscoverHistory and discussed performing belief management using DiscoverHistory as part of DHAgent, we seek to show that this is a reasonable strategy that yields performance improvements when solving the goal achievement problem. To focus on belief management, we consider a constrained version of the goal achievement problem, in which the environment model $M_\Sigma$ is assumed to be complete, and no external agents affect the world. The environments examined in this chapter are partially observable and dynamic, due to the occurrence of events, but the dynamics are fully known.

In this chapter, we describe and defend claims that DiscoverHistory search increases the performance of DHAgent when compared to a replanning agent that can not change its assumptions or infer any unobserved facts. The experimental study in this chapter was previously published with colleagues Kuter and Klenk (2012).

## 5.1 Claim

In this section, we compare DHAgent's performance and knowledge of the world to a replanning agent that does not perform explanation. Instead, this second agent, which we refer to as IncuriousAgent, bases its plans on the projected state as predicted by the incurious explanation, defined in Chapter 4. This is consistent with past work on belief revision operators, which make only the minimal changes necessary to be consistent with contradictory explanations. The only difference between IncuriousAgent and DHAgent is in the construction of a state for use in replanning; like DHAgent, it observes each time it takes an action, and replans whenever its expectations are violated by the newest observation.

We now repeat the first of the three claims presented in Chapter 1, which we provide experimental evidence for in this chapter.

**Claim 1**

*Belief management using* DiscoverHistory *search causes the goal achievement performance (number of goals achieved successfully) of* DHAgent *in a partially-observable, single-agent context to improve relative to a traditional replanning agent,* IncuriousAgent*, which uses a more naive belief management strategy.*

The goal achievement performance of an agent is defined as the percentage of requested goals it is able to achieve in a given scenario. This metric is chosen as the ability to accomplish goals is typically important to an agent's user, and it shows the downstream consequences of failure to manage information wisely. Agents that don't recognize their mistakes will repeat them, and eventually be trapped by them. IncuriousAgent is designed to be typical of agents that take an ad-hoc approach to belief management, and serves as a proxy for a family of online replanning agents that are not made available for scientific comparison.

## 5.2 Related Work

For the convenience of the reader, we briefly summarize some of the most relevant related work introduced in Chapter 2, and differentiate from it.

DHAgent's revision of its beliefs based on an explanation is nearly consistent with Dupin de Saint-Cyr and Lang's (2011) description of a belief extrapolation operator using the *event penalty* ordering. However, the belief extrapolation operator is defined only for propositional environments, and has not, to our knowledge, been implemented and studied experimentally.

Prior research in diagnosis or explanation of action sequences has examined an analogous problem, sometimes referred to as diagnosis of discrete-event systems. While this diagnostic work represents both environments and actions, it is divorced from the decision making

of an agent. This removes the need to examine several important issues. Due to this agent connection, the DiscoverHistory solution considers the need for incrementality, recognition of exogenous events, and inference of the environment state.

Sohrabi et al (2010) suggested that many diagnostic problems can be accomplished by using a planner to construct a series of actions that match the events. While no existing systems have been shown to do this for general explanation procedures, this strategy can be extended to work with deterministic events and a stream of observations, for purposes of comparison to DiscoverHistory, when the number of possible initial states is low (see Chapter 7 for further discussion).

## 5.3 Comparison with Other Agent Approaches

Compared to planning algorithms for partially observable worlds, DHAgent is very flexible, as it tries to achieve success in worlds where success may be impossible, and is able to reason about domains with many possible worlds, unlike modern planners.

Gspandl et al (2011) described an agent, IndiGolog, capable of managing its beliefs based on history-based diagnoses similar to the explanations generated by DiscoverHistory search. IndiGolog executes high-level robotic programs rather than planning as DHAgent does; it's an online agent framework that represents the environment model, agent, and reasoning techniques as part of a common code base. Additionally, the authors state that its applicability to domains of higher complexity is limited. While the work is similar in spirit, the specific problems addressed are different.

Unlike typical contingent and conformant planning agents (e.g., Hoffmann and Brafman 2005; Albore, Palacios, and Geffner 2009; Bertoli and Cimatti 2002; Bryce, Kambhampati, and D. Smith 2004), DHAgent does not project a belief state consisting of all possible futures. Instead, it makes a set of reasonable assumptions about the initial state; effectively, it chooses to believe in the one maximally plausible world, and project only the consequences of that world. We represent transitions as deterministic, so all projections created by a planner result in one possible world after each set of actions. This means that plans

generated by DHAgent are not guaranteed to reach a goal state. However, this is not necessarily a handicap: in the domains we examine, there is no correct plan (contingent or conformant) that achieves a goal state in all possible worlds. The role of DiscoverHistory is to revise the belief set such that the belief state always contains one maximally plausible world, which is a highly efficient strategy compared to full projection.

The SDR planner (Shani and Brafman 2011), based on contingent planning techniques, is similar to DHAgent in that is considers only a small number of possible worlds at a time. However, it assumes environment models that represent sensing actions and contingent effects; this limits its ability to address a large number of initial states.

## 5.4 Experimental Design

We examined the performance of DHAgent in the context of planning and execution in two hazardous partially-observable domains. Unlike standard domains in common use, these domains have been engineered to have a very large number of possible worlds and include events that will happen in some possible worlds and not others. Thus, events will occur at execution time that can not be predicted in advance without considering all possible worlds. We believe that this provides more realistic difficulties than existing domains. These domains are also hazardous, in the sense that states can be reached from which a goal can not be achieved. The initial observation of each problem in these domains is consistent with many possible worlds, some of which are unsolvable. This is important because traditional techniques have trouble dealing with large numbers of possible worlds, or rely on being able to find a complete advance plan that is guaranteed to succeed. Our technique is resilient to these problems. However, no modern planner intended for of reasoning about uncertain environments was able to successfully plan for any of these problems despite extensive testing, including SDR (Shani and Brafman 2011), Contingent FF (Hoffmann and Brafman 2005), CLG (Albore, Palacios, and Geffner 2009), MBP (Bertoli and Cimatti 2002) and POND (Bryce, Kambhampati, and D. Smith 2004).

Hazardous Rovers is a navigation domain with hidden obstacles inspired by the difficulties encountered by the Mars Rovers, and based on the Rovers domain from the International Planning Competition (IPC) 2002 Long and Fox 2003. Specifically, individual locations may be windy, sandy, and/or contain sand pits, which the rover cannot observe directly. Sandy locations cause the rover to be covered in sand; while covered in sand, the rover cannot perceive its location or recharge. Sand pits stop the rover from moving; the rover can dig itself out at a high energy cost. Windy locations clear the sand off of the rover, but due to a malfunction, may confuse the rover's compass, causing it to move in the wrong direction. Rovers must maintain their energy level. Each movement action costs energy, and rovers can no longer accomplish goals if their energy runs out. Recharging can only be performed when the rover is in the sun. Goals in this domain are based on simply navigating to a target location at least four actions away. However, based on an initial observation, it is always possible that the rover is surrounded by overshadowed sand pits, so no sequence of actions is ever guaranteed to succeed.

Hazardous Satellites is based on the Satellite domain from IPC 2002. The objective in each scenario is to acquire images of various phenomena using various specialized instruments and transmit them to Earth. Our additions to this domain include various causes of satellite malfunction: supernovae, which can damage sensitive instruments that are pointed toward them; fuel leaks, which cause fuel reserves to diminish rapidly; and motor malfunctions, which delay a satellite's turn to a new perspective. When fuel reserves are depleted, no further goals can be accomplished. It will sometimes be impossible to obtain a clear target image due to a supernova, so no sequence of actions is guaranteed to succeed.

### 5.4.1 Problem Generation

We wrote a problem generator for each domain that randomly creates an initial state including both observable and hidden facts and goals. For the Hazardous Rovers domain, each starting state contained 3 rovers, and a goal for each rover that required it to move to a new destination. Goal destinations were generated randomly such that the rover must

cross at least 3 distinct locations to accomplish its goal. Each scenario took place on a 6 $\times$ 6 grid of locations connected in the four compass directions. Hidden state was assigned independently for each location and condition with frequency $\lambda$. In all, there are 108 non-observable binary fluents in this domain, which means that a total of $2^{108}$ possible states are consistent with each observation.

Each randomly-generated Hazardous Satellites problem included 3 satellites and required the attainment of 8 image acquisition goals. Each image target was chosen randomly from a set of 20. Targets were associated with supernovae, fuel leaks, and motor malfunctions based on a frequency $\lambda$. In all, there are 60 non-observable binary fluents in this domain, so $2^{60}$ possible states correspond to each observation.

### 5.4.2 Search Configuration

For this experiment, DHAgent searched the space of available explanations for a **closest explanation** which is minimally distant, among all strictly plausible explanations, from a root explanation. This is done by using an iterative deepening search whose increasing depth bound $k$, is based on distance from the root explanation, which is DHAgent's prior explanation. That is to say, at each step of the iterative deepening search, a depth-first search is conducted that returns all strictly plausible explanations with distance $d$ such that $d < k$. "Distance" here is defined as the number of times a refinement operator has been applied to an explanation; an explanation of depth $d$ is therefore an explanation resulting from $d$ recursive applications of refinement operators to the prior explanation found by DHAgent. The first explanation found by search is therefore the closest explanation, as no explanation with depth greater than $d$ can have been found before an explanation of depth $d$ in an iterative deepening search. This definition of distance, based on refinements from a root explanation is used for three reasons: (1) basing on the current explanation allows DiscoverHistory search to be used for incremental updates, which is desirable for continual planning systems. (2) The definition permits an efficient search. (3) Considered as a belief management operator, it makes sense for a DiscoverHistory search to find

the minimum necessary changes to its current belief structure.

Search terminates when $k$ reaches a maximum depth bound $k_{MAX}$ if a strictly plausible explanation has not been found. This ensures that search always returns within a reasonable amount of time (the resulting search finishes in polynomial time), but also results in a search that is not a complete solution of the explanation generation problem. Allowing for this kind of failure, we believe, is a reasonable tradeoff for the execution time savings, if the failure is not too frequent. As the dynamics of the environment are known, for this experiment DHAgent used only the refinement operators *AddGround*, *RemoveOcc* and *AssumeInitial* described in Chapter 4. Selection of an inconsistency to resolve at each search node was done randomly. For this experiment, the value of $k_{MAX}$ was 6.

### 5.4.3 Setup

We randomly generated 25 problems in each domain, and recorded the percentage of goals achieved by IncuriousAgent and DHAgent on each problem. A simulator executed the actions requested by each agent based on the domain's transition function and the ground truth initial state generated for each problem. Observations were generated and provided to each agent based on an observation model. The correct transition model and observation model were provided to each agent, as well as the generated goals, but the ground truth initial state was withheld.

For the Hazardous Rovers domain, we used four values for $\lambda$, the frequency of hidden state information: $0.0, 0.1, 0.2$, and $0.3$; in the Hazardous Satellites domain, we used one value: $\lambda = 0.3$. A search depth bound of 7 was used in all of our experiments, i.e., DHAgent's search for explanations returned no results if all plausible explanations were more distant than 7 recursive refinements from the prior explanation.

## 5.5 Results

Table 5.1 shows a comparison of the performance of DHAgent and IncuriousAgent. Statistical results are based on a two-tailed t-test with paired samples, which showed that

Table 5.1: Statistical $t$-test results, comparing the percentage of goals accomplished by IncuriousAgent and DHAgent in the Hazardous Rovers and Satellites domains.

| **Domain** | IncuriousAgent | DHAgent | $t$-**test** |
|---|---|---|---|
| **Hazardous Rovers** ($\lambda = 0.1$) | 65.3% | 78.7% | 0.001 |
| **Hazardous Satellites** ($\lambda = 0.3$) | 52.5% | 76.0% | $< 0.001$ |

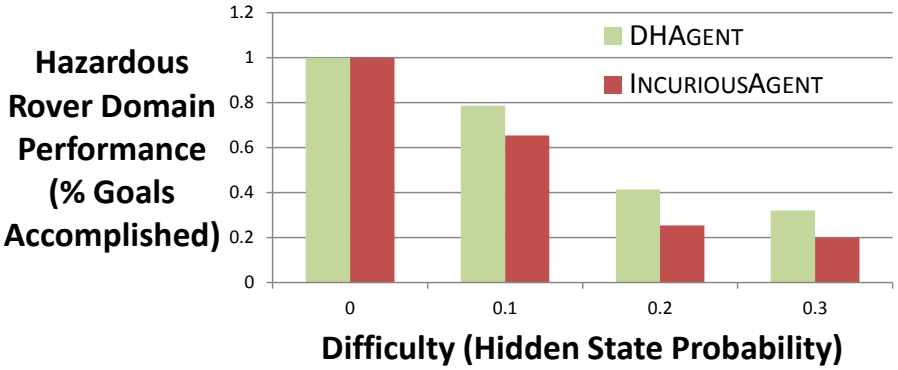


Figure 5.1: Comparison of percentage goals accomplished at various difficulty levels.

DHAgent statistically outperformed IncuriousAgent in both domains. As the only difference between the two agents is the use of explanation, it's clear that the use of DiscoverHistory search improved performance. This shows that abductive explanation of state events can improve performance over replanning alone in partially-observable dynamic environments.

To further examine the impact of hidden state on performance, we increased the difficulty of the Hazardous Rovers domain by varying the probability of hidden states. Figure 5.1 compares the performance of the two agents at 4 difficulty levels: $\lambda = 0.0, 0.1, 0.2$, and 0.3. At $\lambda = 0.0$, the closed world assumption is correct; as expected, we see perfect performance from both agents since no explanation is necessary. As the probability of the closed world assumption being inaccurate increases, both agents perform more poorly, but DHAgent continued to significantly outperform IncuriousAgent. At $\lambda = 0.3$, DHAgent accomplished goals 50% more often than IncuriousAgent. Differences between them were statistically significant for all $\lambda > 0.0$.

## 5.6 Discussion

Results of our experiments show that DHAgent does indeed achieve more goals than IncuriousAgent, which supports our claim. Furthermore, our results in the Hazardous Rovers domain show that a higher percentage of fluents for which the closed world assumptions are incorrect cause a higher gap between DHAgent and IncuriousAgent's performance. This supports the idea that the more correct an agent is, the less likely it is to get stuck in a dead end. Based on this work, we expect similar results to hold in other partially observable domains where hidden state can affect the outcome of an agent's actions in ways that are *eventually* observable; DHAgent's strategy is appropriate when its observations increase its information about the environment, and is conversely not useful when the effects of its actions are themselves hidden. Other strategies, such as constructing and pursuing a contingent plan from the start, will likely yield better performance when the number of possible initial states is limited.

# Chapter 6: Efficiency Issues in Explanation

In Chapter 5, we demonstrated that DiscoverHistory search can improve performance. However, this improvement must come at a cost; search can be computationally expensive, and will always be more expensive than *not* doing it. However, we take the position that performance improvements can be worth the price. While search is well-known to have an exponential worst-case complexity, heuristics and bounds can lower these computational demands in practice. Here, we analyze the execution time of DiscoverHistory in our sample domains to get a sample of practical costs.

In this chapter, we examine the practical computational requirements of two implementations of DiscoverHistory with different performance characteristics, and analyze the computational efficiency of each for explanation generation in the Hazardous Rovers and Hazardous Satellites domains. We also investigate the performance impact of a second distance metric on DiscoverHistory search.

## 6.1 Differences Between Implementations

In this section, we describe in more detail two implementations of DiscoverHistory, which we refer to as DiscoverHistory$_1$ and DiscoverHistory$_2$. When we wish to refer to characteristics of the family, we will simply say DiscoverHistory.

DiscoverHistory$_1$ and DiscoverHistory$_2$ differ in how they maintain the structure of the partial ordering $R$ over occurrences. DiscoverHistory$_1$ search maintains a data structure as part of an explanation that maps each relation and function instance to a partially ordered list of all occurrences relevant to it. When events are added, they must be ordered by this structure, and the list allows DiscoverHistory$_1$ to determine compatible

orderings. Using this instance map, DiscoverHistory$_1$ can quickly determine all inconsistencies, by iterating over pairs of occurrences that differ in value. However, requiring that $R$ be ordered increases the branching factor when adding events.

DiscoverHistory$_2$ search, in contrast, enumerates the set of all possible events as a preprocessing step, and determines when they occur, assigning a time to each one. The ordering $R$ is defined as a comparison of the values of this function, $time : O \to \mathbb{N}$, between occurrences. Computation of the list of inconsistencies is slower for DiscoverHistory$_2$ than for DiscoverHistory$_1$, because the state must be projected. DiscoverHistory$_2$, however, does not spend time maintaining the occurrence ordering, and the branching factor of DiscoverHistory$_2$'s search space is greatly reduced: DiscoverHistory$_2$ considers only the set of possible event occurrences $E_{poss}$ during search, instead of the larger set $E$ containing all occurrences that can be described. Finally, DiscoverHistory$_2$ incurs extra overhead in the pre-enumeration of the set $E_{poss}$.

We previously defined distance between explanations (see Section 5.4.2) based on the number of changes (i.e., refinements) made to reach a new explanation from a root explanation. The search method used here, which is the same as described in the prior experiment, employed a search bound based on distance so as to find a closest explanation first. In this experiment, we consider a second definition of distance between explanations based on the number of *decisions* made (i.e., when one branch is selected from among several) in finding a new explanation. This new distance metric "fast tracks" explanations that are refined unambiguously, because only one possible refinement leads to a legal explanation. Both metrics make use of the search tree, in that the first explanation found in an iterative-deepening or breadth-first search is guaranteed to be a closest explanation.

## 6.2 Experimental Evaluation

In this evaluation, we compare DHAgent using DiscoverHistory$_1$, DiscoverHistory$_2$ using the change-based metric, and DiscoverHistory$_2$ using the decision-based metric, as well as IncuriousAgent on the problems described in our last evaluation. We report

results for Hazardous Rovers with $\lambda = 0.1$, 0.2, and 0.3, and for Hazardous Satellites with $\lambda = 0.3$. Metrics reported in this study include goal achievement performance and computational efficiency. Search using DiscoverHistory$_1$ and DiscoverHistory$_2$ are identical to the search description from Section 5.4.2, i.e., an iterative deepening search with random inconsistency selection that terminates when the first strictly plausible explanation is found or after a maximum depth bound $k_{MAX}$ is reached. The depth of an explanation is measured using either a change-based or decision-based metric for determining distance from a root explanation, as described earlier.

To examine the behavior of the DiscoverHistory implementations in these domains, we ran the four agents on each of the four problem sets using 5 different values for the maximum search depth $k_{MAX}$. For each problem, we recorded both execution time and number of goals achieved. The results of these evaluations are depicted in Figure 6.1. A quick glance at these graphs shows two important behaviors: DiscoverHistory$_2$ achieves the highest performance at every maximum plausibility bound, and the time required by DiscoverHistory$_1$ tends to increase dramatically as the maximum plausibility bound increases. Based on this alone, DiscoverHistory$_2$ appears to be a better choice. However, the two plausibility metrics confound the results slightly, since a search using the decision count plausibility metric may require a higher bound to reach optimal performance than one using the change count metric. Therefore, we compared the algorithms by first finding the minimum value of maximum search depth at which each agent reached maximum goal achievement performance. Whether a particular agent reached maximum performance was determined by matching it against the highest absolute performer on that set of scenarios using a 1-tailed paired sample t-test with a 95% confidence threshold. When the null hypothesis of equal means could not be rejected, that agent was said to have reached maximum goal achievement performance. Then we compared the time required for that achievement across all agents.

Table 6.1 shows the results of this comparison. DiscoverHistory$_2$ shows a clear, large advantage in the Hazardous Rover domain. On the Hazardous Satellites benchmark,

Table 6.1: Performance of each of the 3 agents at the lowest maximum search depth with the highest goal achievement rate, along with statistical results. Search depth indicated by $d$, execution time by $t$, and statistical confidence level by $p$.

| **Domain** | DiscoverHistory$_1$ | DiscoverHistory$_2$ **using change metric** | DiscoverHistory$_2$ **using decision metric** |
|---|---|---|---|
| **Hazardous Rovers** $\lambda = 0.1$ | $d = 7$<br>$t = 157$<br>Loses, $p = .0002$ | $d = 7$<br>$t = 15.1$<br>Loses, $p = .003$ | $d = 3$<br>$t = 13.5$<br>Wins |
| **Hazardous Rovers** $\lambda = 0.2$ | $d = 7$<br>$t = 213$<br>Loses, $p = .0002$ | $d = 7$<br>$t = 19.2$<br>Tied | $d = 5$<br>$t = 19.6$<br>Tied |
| **Hazardous Rovers** $\lambda = 0.3$ | $d = 7$<br>$t = 269$<br>Loses, $p = .0003$ | $d = 9$<br>$t = 19.3$<br>Tied | $d = 5$<br>$t = 19.0$<br>Tied |
| **Hazardous Satellites** $\lambda = 0.3$ | $d = 5$<br>$t = 13.3$<br>Wins | $d = 7$<br>$t = 14.6$<br>Loses, $p < .0001$ | $d = 5$<br>$t = 14.5$<br>Loses, $p < .0001$ |

DiscoverHistory$_1$ statistically outperforms DiscoverHistory$_2$. However, the percentage difference in execution time is small. We see, therefore, that neither of the implementations considered has a clear performance advantage in all domains.

## 6.3 Discussion

We've shown that the DiscoverHistory$_2$ implementation based on pre-enumeration and a representation of ordering $R$ based on integers is, for at least one domain, much more computationally efficient than one based on DiscoverHistory$_1$, although both are able to achieve comparable levels of goal achievement performance. We believe that the greater efficiency of DiscoverHistory$_2$ in the Hazardous Rovers domain is primarily due to the reduced branching factor.

Significantly, we found that in both domains and for each DiscoverHistory version, a depth bound of 9 was sufficient to achieve optimal performance. In general, if such a reasonable depth bound can be found, it can be used to constrain the computational

requirements of DiscoverHistory.

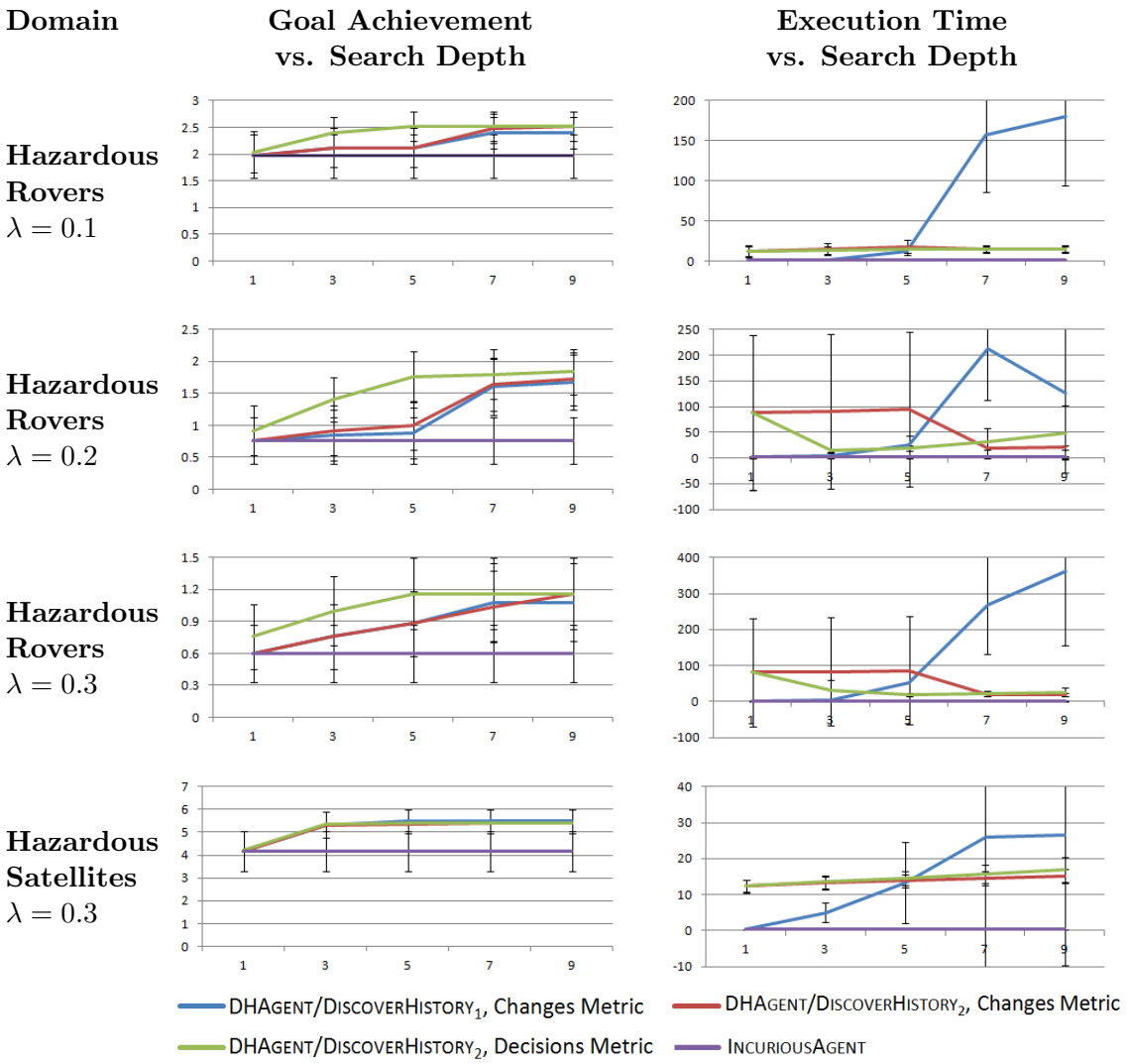


Figure 6.1: From top to bottom, the charts show performance versus search depth in the Hazardous Rovers domain with $\lambda = 0.1$, 0.2, and 0.3 and the Hazardous Satellites domain with $\lambda = 0.3$. Goal achievement performance is on the left, execution time on the right. Error bars show 95% confidence intervals. Data shown for DHAgent using DiscoverHistory$_1$ (blue), DiscoverHistory$_2$ with the change-based metric (red), DiscoverHistory$_2$ with the decision-based metric (green), and IncuriousAgent (purple).

# Chapter 7: Explanation in a Multi-Agent Domain

In this chapter, we describe a revised version of DiscoverHistory, Multi-Agent DiscoverHistory (MADH), and investigate the accuracy and efficiency with which it identifies actions of other actors in a multi-agent environment, which is a highly important capability in both collaborative and competitive settings. The study reported here were previously published with colleague David Aha (Molineaux and D. W. Aha 2015).

Efficiency of search for multi-agent explanations tends to be lower in practice than for a single-agent environment, because changes caused by other agents can happen at any time. The pre-enumeration strategy discussed in Chapter 6 for reducing branching factor and simplifying maintenance of the ordering $R$ does not work in a multi-agent setting, because the number of possible actions and events is too high to efficiently compute. Furthermore, there are more consistent explanations, because another agent can choose among many similar actions with similar effects. Therefore, in this chapter we explore techniques for speeding up search by using heuristics, inconsistency selection, and more frequent addition of projected events into an explanation.

This chapter is organized in the following fashion: first, we describe our motivation for examining explanation generation in a multi-agent domain, then we describe our claim that a search using MADH is faster and more accurate than other strategies. We follow this by discussing differences between DiscoverHistory and MADH that increase efficiency in a multi-agent domain. We then present the multi-agent environment Autonomous Squad Member, which we use for our investigation, and give an extended example of the operation of MADH in ASM. Finally,we then present an evaluation of MADH that supports our claim. For comparison, we show accuracy and efficiency results for a forward-chaining explainer called Deductive Explanation Generator (DEG) suggested by work in diagnosis.

## 7.1 Motivation

In many domains, it is desirable for a cognitive agent to collaborate or compete with other agents, especially humans. In general, this requires the agent to understand what other agents are doing. This task may be non-trivial, particularly in partially observable environments. Because many modern robotic sensors can only gather information at discrete intervals, a cognitive agent for real-time environments should be able to infer the occurrence of an action taken by another agent from observations that precede and follow it. This requires the agent to perform a diagnostic or explanatory task, inferring actions, events, and processes that explain its observations. For example, an agent performing alongside an army patrol would, when the patrol suddenly comes under fire, recognize that other team members are taking cover, and execute appropriate actions to help.

## 7.2 Claims

**Claim 2**

MADH *search is faster than a deductive strategy,* DEG, *for generating and maintaining explanations of exogenous actions in a dynamic, partially observable, multi-agent environment, and maintains comparable accuracy.*

We examine this claim in order to determine the efficacy of DiscoverHistory search in a multi-agent environment. The constraints of the environment forced us to redefine DiscoverHistory somewhat, resulting in MADH, but the basic principles remain the same. Unlike prior experiments, where we measured the effectiveness of the overall agent, this experiment examines the accuracy and efficiency of explanation generation itself. In order to measure accuracy, we assume the dynamics are fully known. Learned dynamics cannot be compared to true occurrences reliably, as different representations found in learning may prove equivalent in terms of prediction.

## 7.3 Definitions

For purposes of describing improvements to search, we now redefine an explanation to add an element corresponding to the history of the search path that led to it. Formally, the redefined explanation is of the form $\chi = \langle O, D, R, C, V \rangle$, where the elements $O, D, R$, and $C$ are defined as before. The new element is a list of revisions $V = [v_1, v_2, \ldots, v_n]$, where $v_1$ is the oldest revision and $v_n$ the newest. Each revision is a tuple $v_n = \langle f_n, O_n^-, O_n^+, \theta_n, R_n^+, C_n^+ \rangle$, that describe the difference between parent and child explanations. Semantically, a revision describes the changes made by a refinement method or other explanation-modifying function, such as FindExtraEvents, with respect to a parent explanation. Individual items are as follows:

1. $f$ is the specific function that performed this modification
2. $O_n^-$ is a list of occurrences removed
3. $O_n^+$ is a list of occurrences added
4. $\theta_n$ is a substitution performed to all occurrences
5. $R_n^+$ is a list of ordering constraints added
6. $C_n^+$ is a list of constraints added

Formally, given an explanation $\chi_n = \langle O_n, R_n, D_n, C_n, V_n \rangle$ and a child created from it $\chi_{n+1} = \langle O_{n+1}, R_{n+1}, D_{n+1}, C_{n+1}, V_{n+1} \rangle$, the following relationships hold in the final revision of $V_{n+1}$:

$$O_{n+1}^- = O_{n+1} / apply(\theta_{n+1}, O_n)$$
$$O_{n+1}^+ = apply(\theta_{n+1}, O_n) / O_{n+1}$$
$$R_{n+1}^+ = R_{n+1} / R_n$$
$$C_{n+1}^+ = C_{n+1} / C_n$$

For purposes of maintaining this history, all refinement methods, as well as FindExtraEvents, must add an entry to the revision list of a child explanation corresponding to

the specific changes made by that method to the child with respect to its parent. From this point on, we assume that all refinement methods do this, as it is not difficult to construct modified refinement methods that maintain this history.

We can now define the newest assumption of an explanation $\chi = \langle O, D, R, C, V \rangle$, which is a unique assumption occurrence in the current explanation that was added more recently than any other assumption occurrence in the history, as well as more recently than any revision by FindExtraEvents. If FindExtraEvents is more recent than any addition of an assumption occurrence, there is no newest assumption. We shall now define this formally through induction. In this definition, we use the functions $last(L)$, which returns the last item of the list $L$, $first(S)$, which returns an arbitrary item in a set $S$ or the first value in a list $S$, and $butlast(L)$, which returns a copy of a list $L$ with its last item removed.

$$
\begin{aligned}
&newestAssumption(\chi = \langle O, D, R, C, V \rangle) \equiv \\
&\qquad mostRecentIn(butlast(V), last(V), assumptions(O)) \\
&mostRecentIn(V, v = \langle f, O^-, O^+, \theta, R^+, C^+ \rangle, U) \equiv \\
&\qquad \begin{cases}
\bot & \text{if } f = \text{FindExtraEvents} \\
apply(\theta, mostRecentIn(butlast(V), last(V), U) & \text{if } U \cap O^+ = \emptyset \\
first(U \cap O^+) & \text{otherwise}
\end{cases}
\end{aligned}
$$

This states that the newest assumption is the most recent occurrence according to the revision history that is found in the assumption list. The relation mostRecentIn is defined recursively, such that if no occurrence in the input set $U$ was added in the last revision, the substitution $\theta$ is applied to whatever occurrence was most recent before that revision. If the most recent revision was created by a call to FindExtraEvents, then $mostRecentIn$ has no solution, which is indicated by the return value $\bot$. If an occurrence in $U$ is also in the list of added occurrences, however, it is returned. Note that the call to the function $first$ is actually unambiguous for all described refinements, as no refinement adds more than one assumption at a time. For example, suppose an assumption $o$ was added in revision $v_{n-3}$ of an explanation $\chi = \langle O, D, R, C, V \rangle$, and no further assumptions were added in $v_{n-2}$, $v_{n-1}$ or $v_n$. The function $newestAssumption(\chi)$ would return $apply(\theta_n, apply(\theta_{n-1}, apply(\theta_{n-2}, o)))$.

We also formally define the concept of an ambiguous inconsistency. As we said before, an ambiguous inconsistency is one that can be resolved through multiple legal substitutions $\theta$ to an explanation $\chi$. The formal statement of this is as follows:

$$\begin{aligned}
&resolutionsBySubstitution(\chi, n) \equiv \\
&\quad UnifyInconsistent(\chi, n) \\
&\quad \cup \{\chi' = \langle O, D, R, C, V\rangle | \chi' \in ReorderThird(\chi) \\
&\qquad\qquad \wedge last(V) = \langle f, O^-, O^+, \theta, R^+, C^+\rangle \wedge R^+ = \emptyset\} \\
&ambiguous(n, \chi) \equiv |resolutionsBySubstitution(\chi, n)| > 2
\end{aligned}$$

We can also now formally define the set of unambiguous inconsistencies (abbreviated $\mathsf{UI}$) in an explanation:

$$\mathsf{UI}(\chi) \equiv \{n | n \in \mathsf{Inconsistencies}(\chi) \wedge \neg ambiguous(n, \chi)\}$$

We define the function $totallyOrdered : \mathbb{S}, \{I \times I \to \{T, F\}\} \to \{T, F\}$ over sets and ordering functions to return true iff the ordering function orders every element of the set with respect to every other. Formally,

$$totallyOrdered(S, \prec) \equiv \forall i, i' \in S, i \prec i' \vee i' \prec i.$$

## 7.4 MADH

Algorithm 12 describes MADH. There are three differences between the definition of MADH here and the definition of DiscoverHistory in algorithm 1. The first two are new conditions for adding projected events to the explanation. On line 3, we see the first new condition, which requires that the newest assumption be totally ordered with respect to observations and participate in no unambiguous inconsistencies. This causes projected events to be added every time a new assumption is "ready". When this condition is met, the newest assumption has reached its peak predictive power; if that assumption is correct, events derived from it are likely to be in the explanation. Adding those events through FindExtraEvents is easier and faster than inferring them through search. The reason why this is done for the newest assumption only should become clear when we discuss inconsistency

selection.

The second condition, on line 5, requires that there be no unambiguous inconsistencies in $\chi$. This is the case because we shall expect search using MADH to return ambiguously plausible explanations rather than strictly plausible explanations.

The third difference from DiscoverHistory is the reference to a selection function MASelect, which specifies a particular order for resolution of inconsistencies. We will detail this function in Section 7.4.2.

**Algorithm 12:** Multi-Agent DiscoverHistory

```
1  Procedure MADH (χ, M_Σ, Π)
2  begin
3      if a = newestAssumption(χ) ∧ ∄n ∈ UI(χ) : prior(n) = a ∨ next(n) = a then
4          χ ←FindExtraEvents (χ)
5      else if UI(χ) = ∅ then
6          χ ←FindExtraEvents (χ)
7      if UI(χ) = ∅ then
8          return {χ}
9      i ←MASelect(Inconsistencies(χ))
10     X ← ∅
11     for π ∈ Π do
12         X ← X ∪ π(χ, M_Σ, i)
13     return {χ' ∈ X|InvariantsMet(χ')}
```

### 7.4.1 Event Projection

New conditions on calls to FindExtraEvents are useful for two reasons: they enable a search for ambiguously plausible explanations, and speed up search. As described in Chapter 4, ambiguously plausible explanations can represent a set of strictly plausible explanations. Since all members of that set are solutions to the explanation generation problem, it often does not make sense to choose among them when the set can be compactly represented.

Search speedup from calling FINDEXTRAEVENTS results from a reduction in the number of inconsistencies that must be resolved. Suppose an assumption $a$ causes a large number of events, several of which have direct effects on observations in $\boldsymbol{o}$. If FINDEXTRAEVENTS is called only after all inconsistencies have been worked out, each causal chain of events from $a$ to $o$ must be found through search to remove all inconsistencies. However, if FINDEXTRAEVENTS is called as soon as $a$ is fully bound, only a single causal chain of events from $a$ to $o$ must be found. In order to take advantage of this, we also modify the inconsistency selection process to bias it toward resolving inconsistencies in one particular event chain.

### 7.4.2 Inconsistency Selection

Selection of inconsistencies can have a large impact on search efficiency. We assume that the order of resolution for inconsistencies does not impact search completeness. If this is true, efficiency should be the main criterion for choosing which inconsistency to resolve in a particular node. Choice of inconsistency can affect both the branching factor and depth of search, as some resolutions cause more commitments to be added to an explanation (potentially reducing search depth), and some inconsistencies have more possible resolutions than others (affecting branching). To intelligently select among these inconsistencies in the large explanation space caused by multiple agents, we consider a specific inconsistency selection method, MASELECT. MASELECT considers inconsistencies in the following order, stopping with the first non-empty set:

1. Inconsistencies with a single resolution (causing no immediate branching) or none (causing termination of that search path).
2. Unambiguous inconsistencies involving the newest assumption.
3. Unambiguous inconsistencies for which the most recently added event is the next occurrence.
4. Unambiguous inconsistencies.

5. Ambiguous inconsistencies.

Whenever multiple inconsistencies are considered, an inconsistency with the fewest refinements (i.e., which causes the least branching in search) is selected for refinement.

Inconsistencies with a single resolution are practically free in search, as they cause no branching. When there are no such inconsistencies, the selecting process drives toward identifying a consistent assumption that is ordered with respect to all observations. If an assumption has already been found, refining inconsistencies involving that assumption will help constrain it, reducing the possibility of incorrect events. If it has not, choosing an inconsistency that has the most recently added event as its next occurrence pushes up a chain toward an assumption. Once such an assumption is found, event projection can be used to deduce all events that result from it, which quickly reduces the remaining inconsistencies.

Unambiguous inconsistencies are ordered before ambiguous inconsistencies because of the larger branching factor of ambiguous inconsistencies and in order to preserve ambiguity where it does not affect the set of occurrences in an explanation. This allows an ambiguously plausible explanation to stay uncommitted with respect to values when there is no information to base that commitment on.

### 7.4.3 Refinement Methods

In order to reduce the branching factor, MADH search uses $AddMinimal$ rather than $AddGround$, and does not maintain a total ordering among related occurrences. To support creating orderings as necessary, the $OrderInconsistent$ refinement method is used. Due to the introduction of variables by $AddMinimal$, the refinement methods $UnifyInconsistent$ and $ReorderThird$ are necessary to create substitutions where required. Finally, the refinement methods used in prior experiments, $AssumeInitial$ and $RemoveOcc$, are also included.

### 7.4.4 Search Configuration

In order to maximize efficiency, we use best-first search with MADH with a domain-independent heuristic function. Search terminates when the first ambiguously plausible explanation is found, returning that single explanation. We examine a single heuristic function, in which the cost of an explanation is calculated based on plausibility and efficiency. Plausibility is measured as the sum of three metrics:

- *Age* is calculated as the number of observations between the earliest occurrence added during the current search and the current time. It measures how long something must have gone unnoticed for this explanation to be correct. This reflects a bias that a more recent mistake is more likely than an older one, which might have been noticed earlier.

- *Precedence ambiguity* is calculated, for each event and action, as the number of observations that have no precedence relationship with that occurrence. Formally, $\mathsf{precedence\text{-}ambiguity}(occ) = |\{\mathsf{obs} | \mathsf{obs} \in \boldsymbol{o} \wedge \mathsf{obs} \not\prec occ \wedge occ \not\prec \mathsf{obs}|$. MADH requires that this be 0 in an ambiguously plausible explanation, as each event must have a proximate cause and actions and observations are totally ordered. Therefore, an explanation with a lower precedence ambiguity is less distant from a solution.

- *Assumption cost* is counted as $\mathsf{ASSUMPTION\_COST}$ for each exogenous action and each initial state assumption in the current revision history. It measures how many distinct factors were unknown prior to search. This component rewards parsimony.

In combination, these three metrics guide the search toward more plausible explanations, but there is no guarantee of optimality.

The efficiency component of the explanation cost function includes two factors:

- *Search depth* is equal to the depth of an explanation in a search tree. Use of search depth prevents recursive applications of refinement operators that do not change plausibility from dominating the search space. While incorporating search depth prevents an infinite recursion, it is not intended as a significant heuristic component.

- *Event load* penalizes events added by the *AddMinimal* refinement. This is counted as EVENT_COST for each event in the revision history. This biases the search toward explanations with fewer abductively inferred events, which reduce search depth with earlier application of event projection.

Use of this heuristic function is intended to decrease search times in a large environment for which possible occurrences cannot be enumerated. In the future, we would like to examine multiple possible weightings for each of these metrics, to determine how such weightings affect search time in different domains.

### 7.4.5 Autonomous Squad Member Domain

The Hazardous Rovers and Hazardous Satellites domains discussed in Chapter 5 are single agent, and are not easily modified to include relations and functions that can be affected by an external agent. To investigate the accuracy and speed of MADH, therefore, we consider a new environment called Autonomous Squad Member (ASM).

In this environment, a robot assists an army patrol by following and carrying equipment, which requires the robot to monitor what patrol members are doing. In this domain, actions include team member movement, change of team member postures, firing of weapons, communication by gesture, and directions to subordinates. Modeled events include change of GPS coordinates, arrival at a destination, completion of a directive, auditory and visual observation of activities by humans, and injury and death due to gunfire. This domain is complex enough to be fairly challenging, requiring frequent explanation to understand what actions underlie environment changes caused by team members and enemies.

### 7.4.6 Extended Example

In order to give a comprehensive picture of how MADH works, this extended example gives detail on the knowledge maintained by the agent, the output of refinement methods, selection of an inconsistency, and computation of heuristic values.

We start near the beginning of a scenario, where one team member of the patrol has just

started to walk away from the starting location. So far, the robot has started to follow the team leader, labelled as MEMBER1, and made three successive observations of the environment. When the fourth observation arrives, it yields two surprising observations: (1) a team member, labelled MEMBER2, is reported to be at an unnamed location (generic label UNK-LOCATION) rather than the starting location (labelled LOCSTART), as the robot expected; and (2) team member MEMBER2 is reported to be somewhere along the route ROUTE1. Using MADH, the agent attempts to modify its explanation to explain the unexpected observations. Figure 7.1 lists parts of the robot's memory before MADH begins, including the existing explanation that the robot has been maintaining, partial observations, and the discovered inconsistencies.

**Initial Explanation**

*Occurrences*

$\mathsf{obs}_0$: ... (OBJECT-LOCATION MEMBER2 LOCSTART) ...
$a_{h,1}$: (FOLLOW ROBOT1 MEMBER1)
$\mathsf{obs}_1$: ...
$\mathsf{obs}_2$: ... (REPORTED-LOCATION MEMBER2 LOCSTART), (REPORTED-ROUTE MEMBER2 NO-ROUTE) ...
$\mathsf{obs}_3$: ... (REPORTED-LOCATION MEMBER2 UNK-LOCATION), (REPORTED-ROUTE MEMBER2 ROUTE1) ...

*Ordering*

$\mathsf{obs}_0 \prec a_{h,1}$.
$a_{h,1} \prec \mathsf{obs}_1$.
$\mathsf{obs}_1 \prec \mathsf{obs}_2$.
$\mathsf{obs}_2 \prec \mathsf{obs}_3$.

*Constraints*: None.

**Inconsistencies**

$\langle$(REPORTED-LOCATION MEMBER2 UNK-LOCATION), $\mathsf{obs}_2$, $\mathsf{obs}_3$,TRUE,=$\rangle$
$\langle$(REPORTED-ROUTE MEMBER2 ROUTE1), $\mathsf{obs}_2$, $\mathsf{obs}_3$,TRUE,=$\rangle$

Figure 7.1: Robot's memory near the beginning of an ASM scenario

The first inconsistency is selected, because it leads to only one child explanation. This child is generated by *AddMinimal*, which adds a new event of type GPS-OBSERVE-LOCATION, representing an update received by the robot from a patrol member's GPS transponder. The event reports a label for a known location a patrol member is near, and a label for any named route the patrol member may be following. The event is partially bound to match the inconsistent literal as well as static literals. Figure 7.2 gives a complete representation of the event model and event.

**Event Model**

```
(:EVENT GPS−OBSERVE−LOCATION
 :PRECONDITION
    (AND (ON−TEAM ?TEAMMATE ?TEAM)
         (IS−ROBOT ?SELF)
         (ON−TEAM ?SELF ?TEAM)
         (EQ (OBJECT−LOCATION ?TEAMMATE) ?LOC)
         (EQ (PERSON−ROUTE ?TEAMMATE) ?ROUTE)
         (OR (NEQ (REPORTED−LOCATION ?TEAMMATE) ?LOC)
             (NEQ (REPORTED−ROUTE ?TEAMMATE) ?ROUTE)))
 :EFFECT
    (AND (SET (REPORTED−LOCATION ?TEAMMATE) ?LOC)
         (SET (REPORTED−ROUTE ?TEAMMATE) ?ROUTE)))
```

**Event**

| | |
|---|---|
| *Type*: | GPS-OBSERVE-LOCATION |
| *Preconds*: | (ON-TEAM MEMBER2 TEAM1) (IS-ROBOT ROBOT1) |
| | (ON-TEAM ROBOT1 TEAM1) |
| | (OBJECT-LOCATION MEMBER2 UNK-LOCATION) |
| | (PERSON-ROUTE MEMBER2 ?ROUTE56); (IS-ROUTE ?ROUTE56) |
| | (NOT (REPORTED-LOCATION MEMBER2 UNK-LOCATION)) |
| *Effects*: | (REPORTED-LOCATION MEMBER2 UNK-LOCATION) |
| | (PERSON-ROUTE MEMBER2 ?ROUTE56) |
| *Constraints*: | None. |
| *Signature Tuple*: | (GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 NO-LOCATION ?ROUTE56) |

Figure 7.2: Representation of GPS-OBSERVE-LOCATION event and model

This event is ordered after $\mathsf{obs}_2$ and before $\mathsf{obs}_3$ by *AddMinimal*. In addition to MEMBER2 and UNK-LOCATION, several other values are bound to model variables based on static observable function and relation instances. Several new inconsistencies are found in this new explanation (shown in Figure 7.3). Each corresponds to a literal from the preconditions or effects of the added event that does not match other occurrences:

1. The route being followed by MEMBER2 is referenced by the precondition of the GPS-OBSERVE-LOCATION event, and represented as the literal (PERSON-ROUTE MEMBER2 ?ROUTE56). The value of this literal has not been assigned by any previous occurrence, and the default value is NO-ROUTE.

2. Preconditions of the GPS-OBSERVE-LOCATION event state that the value ?ROUTE56 must be of type route. However, it is as yet unassigned to any value, so the precondition is not met. This inconsistency also contradicts the default assumption.

3. A precondition of the GPS-OBSERVE-LOCATION event indicates that the location of MEMBER2 is unknown, but it is known at the time of the initial observation, observation1.

4. The occurrence observation4 indicates that MEMBER2 is reported to be following ROUTE1, which contradicts the effect of the GPS-OBSERVE-LOCATION event which places it on the route ?ROUTE56. This information is represented by the literal (REPORTED-ROUTE MEMBER2 ROUTE1).

To select an inconsistency for refinement, MADH first considers inconsistencies with only one possible refinement; there are none. Second, it considers unambiguous inconsistencies relating to the most recently added assumption; none exists. Third, it considers unambiguous inconsistencies for which the most recently added event in the next occurrence; this includes inconsistencies 1 and 4. Among these, inconsistency 1 is selected because it has the fewest possible refinements.

MADH applies refinements to this inconsistency to obtain refined explanations, as follows: *AddMinimal* creates a child explanation with a move action, because the move action

⟨(PERSON-ROUTE MEMBER2 ?ROUTE56), $\mathsf{obs}_0$,
(GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ?ROUTE56)⟩
⟨(IS-ROUTE ?ROUTE56), $\mathsf{obs}_0$,
(GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ?ROUTE56)⟩
⟨(REPORTED-ROUTE MEMBER2 ROUTE1),
(GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ?ROUTE56),
$\mathsf{obs}_3$⟩
⟨(OBJECT-LOCATION MEMBER2 UNK-LOCATION), $\mathsf{obs}_0$,
(GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ?ROUTE56)⟩

Figure 7.3: Inconsistencies after addition of GPS-OBSERVE-LOCATION event

has a literal in its effects of type PERSON-ROUTE. *RemoveOcc* returns no explanations, because neither the prior occurrence nor the next occurrence is defeasible. *AssumeInitial* returns a child explanation containing an initial state assumption, because PERSON-ROUTE is not observable, and the prior occurrence is $\mathsf{obs}_0$. $UnifyInconsistent$ returns a child explanation containing the substitution ?ROUTE56 $\mapsto$ NO-ROUTE, as NO-ROUTE is the default value for PERSON-ROUTE. *OrderInconsistent* returns no explanations, because $\mathsf{obs}_1$ and the GPS-OBSERVE-LOCATION event are already ordered. *ReorderThird* also returns no explanations, as no existing event has an effect of type PERSON-ROUTE. In total, the refinement methods result in three modified explanations. A representation of these, omitting the information carried over from the initial explanation, is shown in Figure 7.4.

Further along one of these search paths, an explanation is found that is close to being ambiguously plausible, as it includes a move assumption that has no unambiguous inconsistencies, and all occurrences are ordered. At this point, event projection must be performed, to ensure that all events that should be caused by changes to the explanation are added. This causes two events to occur, of the type HUMAN-SEES. While these events have no observable effect on the state, they intuitively provide the information that the other patrol members must know what MEMBER2 is doing. Figure 7.5 shows the explanation before and after projection.

**Explanation 1**

*Occurrences*

$e_1$: (GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ?ROUTE56)

$a_2$: (MOVE MEMBER2 ?DEST57 ?ROUTE56 ?TM58 ?ORIGIN59 ?ACT60)

*Precedence*

$\mathsf{obs}_0 \prec a_2$.

$a_2 \prec e_1$.

$\mathsf{obs}_2 \prec e_1$.

$e_1 \prec \mathsf{obs}_3$.

*Constraints*: ?ROUTE56 != NO-ROUTE, ?DEST57 != LOCSTART

*Computed Cost*:

2 (precedence ambiguity) + 10 (assumptions) + 2 (depth) + 6 (event load) = 20

**Explanation 2**

*Occurrences*

$e_1$: (GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ?ROUTE56)

$isa_2$: ⟨initial, (PERSON-ROUTE MEMBER2 ?ROUTE56), TRUE⟩

*Precedence*

$isa_2 \prec a_{h,1}$. $isa_2 \prec \mathsf{obs}_1$.

$isa_2 \prec \mathsf{obs}_2$. $isa_2 \prec \mathsf{obs}_3$.

$isa_2 \prec e_1$. $\mathsf{obs}_2 \prec e_1$.

$e_1 \prec \mathsf{obs}_3$.

*Constraints*: None.

*Computed Cost*: 3 (age) + 10 (assumptions) + 2 (depth) + 6 (event load) = 21

**Explanation 3**

*Occurrences*

$e_1$: (GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ?ROUTE56)

*Precedence*

$\mathsf{obs}_2 \prec e_1$.

$e_1 \prec \mathsf{obs}_3$.

*Constraints*: None.

*Computed Cost*: 2 (depth) + 6 (event load) = 8

Figure 7.4: Resulting explanations with computed explanation costs (ASSUMPTION_COST = 10, EVENT_COST = 6)

After projection, no unambiguous inconsistencies remain, and search returns the explanation produced by projection. After more observations are received, new information may cause the destination of the move action, ?DEST57, seen in Figure 7.5, to be bound. However, the activity id, ?ACT60, is not present in any observable literals, so it will remain unbound indefinitely, with no consequence to the robot's understanding of what is happening.

## 7.5 Experiment

### 7.5.1 Design of the Deductive Explanation Generator

Efficiency results in the diagnosis of discrete-event systems indicate that use of automated planners to solve diagnosis problems is currently the most efficient solution (Grastien et al. 2007). To ensure that generated plans satisfy a sequence of observations rather than a distant goal, Sohrabi et al. (2010) demonstrated the addition of a special advance action to a planning problem; it forces a planner to generate plans that explain all observations. This approach is not directly applicable to the incremental problem, which requires modification of an existing action sequence rather than construction of a new one, as well as the insertion of deterministic events.

We follow the example of Sohrabi et al. with the Deductive Explanation Generator (DEG) (Algorithm 13), a system which uses the principle of an advance action in a forward search to maintain a set of strictly plausible explanations. In a single planning step, a forward state-space planner considers every possible action whose preconditions are met and projects its consequences. Analogously, DEG finds a set of possible actions that could have been performed after each new observation is received, and then generates new explanations corresponding to the combination of each action with each explanation it maintains. Then, consequences are projected for each explanation using FINDEXTRAEVENTS. DEG retains a subset of the resulting explanations that have no inconsistencies, and therefore explain the new observation, as the successor explanation set. Note that DEG does not posit any initial

state assumptions. However, this is not a handicap in the ASM domain, where the initial default assumptions are correct. If this were not the case, DEG would be at an extreme disadvantage due to needing to guess the entire initial state before applying a forward search technique.

Because the branching factor becomes large, the full set of explanations generated by DEG can exceed the available memory space. To avoid this, DEG retains only a subset $XMAX$ of the possible explanations found. These are selected uniformly at random among the plausible explanations found after each observation, and the rest are discarded. Thus, there is no guarantee that DEG will continue to find plausible explanations indefinitely. If no plausible explanation can be found, DEG stops trying to explain and thereafter returns the last non-empty explanation set found. The problem DEG has here is analogous to the memory space problems that plague typical planners in large domains.

DEG is designed to consider all reasonable assumptions, and unless it drops explanations to save memory space, it always finds an explanation with no false positives.

**Algorithm 13:** DEDUCTIVE EXPLANATION GENERATOR

```
1  Procedure DEG (X, M_Σ, i)
2  begin
3      X' ← ∅
4      for χ = ⟨O, D, R, C, V⟩ ∈ X do
5          t ← t|obs_{i-1} ∈ occurs(χ, t)
6          for a ∈ 𝔸 do
7              if projectedState(χ, t) ⊨ pre(a) then
8                  χ' ← ⟨O ∪ {a}, D, R ∪ {⟨obs_{i-1}, a⟩, ⟨a, obs_i⟩}, C⟩
9                  X' ← X' ∪ {FINDEXTRAEVENTS (χ')}
10         X' ← X' ∪ {FINDEXTRAEVENTS (χ)}
11     X' ← {χ' ∈ X'|χ' is strictly plausible}
12     if |X'| > XMAX then
13         X' ←randomSubset(X', XMAX)
14     return X'
```

To interoperate with the environment, DEG uses a minimally modified version of DHAGENT that maintains a set of explanations $X$. On each iteration, this modified DHAGENT replaces its existing explanation set $X$ by the value returned in a call to DEG with the current set $X$, the environment model $M_\Sigma$, and the variable $i$ indicating the index of the most recent observation.

### 7.5.2 Experiment Description

Performing frequent explanations in a real-time environment is a challenging goal, due to the expected computational expense of the task. Typically, a deductive strategy is assumed to be faster than an abductive one. We therefore present an investigation of MADH's efficiency. To assess our claim, we examine the efficiency of DEG and MADH on a series of random runs from scenarios defined in the ASM domain. Variation across these runs primarily comes from the choices made by the simulated human patrol members, who make decisions based on frequent replanning in a nondeterministic hierarchical task network, and act in a nondeterministic order when executing actions.

We measure accuracy with respect to a ground truth explanation generated with knowledge of the team members' actions. Each event and exogenous action in an inferred explanation is paired with a matching occurrence in the ground truth explanation, if any exists. A generated occurrence that matches no occurrence in the true explanation is a *false positive*. Conversely, an occurrence from the true explanation that matches no occurrence in the true explanation is a *false negative*. An event or action in the generated explanation that matches an event or action from the true explanation is a *true positive*. Two events or actions *match* iff some interpretation of each is identical in all preconditions, effects, and its performer, and each is ordered in the same way with respect to observations and other shared occurrences. A match is *partial* if some variables in the generated action or event must be bound to achieve equality.

Based on these definitions, we measure accuracy using a modified version of precision and recall. Under this definition, each true positive resulting from a partial match is discounted

by the ratio of (1) the number of variable substitutions necessary to achieve equality to (2) the number of variables in the original action or event model. We call this the *match ratio*, and define a *true positive ratio* that sums this for all matches. The true positive ratio is similar to the partial precision and recall scoring used in Meadows, Langley, and Emery 2013. Using this definition, our accuracy metrics are:

$$partial\ precision = \frac{true\ positive\ ratio}{\#\ true\ positives + \#\ false\ positives}$$

$$partial\ recall = \frac{true\ positive\ ratio}{\#\ true\ positives + \#\ false\ negatives}$$

We define efficiency as the number of seconds required to perform explanation on the test machine, which is a virtual machine using 4 Xeon X5650 CPUs and 24GB of physical memory. Each iteration was allocated 4GB of process space and 1 CPU.

## 7.6 Results

A typical state in the ASM domain is described by 400 literals; 3 external agents (i.e., squad members) were present in the scenario, and by the end of a run, MADH's explanation typically included more than 100 actions and 400 events. We ran each experimental condition 10 times with the same initial state in the ASM domain, with a different random seed causing distinct behaviors. We used parameter values of EVENT_COST = 6 and ASSUMPTION_COST = 10 in our experiments, which for the ASM domain results in similar ranges for the age, assumption cost, precedence ambiguity, event load, and search depth metrics; no one metric strictly dominates.

DEG is time and memory-intensive at some XMAX values and achieves lower precision and recall at others. With an infinite value for XMAX, DEG would necessarily achieve a higher precision and recall than MADH due to its exhaustive strategy; however, testing has shown that high memory usage at this level inevitably causes failure. Therefore, we instead report on the following conditions:

- DEG-10, using DHAgent and DEG with XMAX set to 10
- DEG-30, using DHAgent and DEG with XMAX set to 30
- MADH, using DHAgent and MADH

Figure 7.6 plots the average partial precision and partial recall of the most accurate explanation found by each agent as explanations change over time. The x-axis of each plot describes the number of observations explained so far. MADH, represented by the solid blue line, achieves similar partial precision and recall to DEG-30. The accuracy of DEG-10 decreases quickly (relative to the other conditions). In some runs, DEG-10 failed to maintain a plausible explanation; its random sample is too small to provide sufficient generality to always find a plausible explanation. Subsequently, it repeatedly reported previously found explanations, which decreased in accuracy as more occurrences accumulated.

These graphs also show that no experimental condition achieves a partial recall value far above 0.8; this is because some of the true events and actions in the world occur away from the robot, where their effects cannot be directly observed. Although these events might be inferred based on later observations, achieving near-perfect recall is highly unrealistic.

To perform a statistical comparison, we compared the ranges of the 95% confidence interval for mean partial precision and recall between conditions at each point on the curve. In each comparison, MADH eventually reaches a turning point, after which it always outperforms DEG, maintaining a lower bound for mean partial precision or recall greater than the upper bound of DEG-10 and DEG-30 for all later observations. For partial precision and DEG-30, that turning point occurs at 36 observations; for partial precision and DEG-10, 12 observations; for partial recall and DEG-30, 63 observations; for partial recall and DEG-10, 42 observations. As MADH significantly outperforms DEG with a small margin, it seems fair to say that MADH's accuracy is at least comparable to that of DEG.

Comparing the efficiency of these conditions (Table 7.1) highlights the major advantage to using MADH. The differences shown are highly significant, with $p < .001$. While a large enough body of unambiguous explanations can maintain reasonable accuracy, it is highly

Table 7.1: Efficiency results for the ASM Domain

| **Experimental Condition** | **Average Time Spent Generating Explanations (minutes)** | **Average Time Spent per Observation (seconds)** | **Average Time Between Novel Observations** |
|---|---|---|---|
| DEG-10 | 94.0[1] | 45.2 | |
| DEG-30 | 425.6 | 176.1 | 45.5 |
| MADH | 5.4 | 2.2 | |

inefficient. In contrast, MADH's intelligent search techniques reduce explanation time to a relatively short interval. However, even when maintaining relatively few explanations, DEG is too slow for realistic use. The average interval between novel observations in the ASM domain is 45 seconds. The DEG-10 condition would consume nearly all of that time, leaving no time for replanning and other activities. The DEG-30 condition takes even longer, meaning that several novel observations would be received while the agent was considering a previous observation.

In summation, our investigation has shown the effectiveness of MADH in a large, partially observable multi-agent domain. We have supported our claim that MADH is capable of maintaining a comparable level of accuracy to a deductive explanation generator at far lower time requirements. We believe that the reason for higher performance here is the high number of actions, which increases the branching factor of a deductive search relative to an abductive search, which generally has a higher cost per search node to perform inference. While this data covers only one domain, we expect to see similar results in any domain with a large number of possible actions, or a large number of possible initial states.

## 7.7 Discussion

We introduced a revised version of DiscoverHistory, MADH, that can efficiently explain actions in the execution context we used in our study. Efficiency results indicate it may be fast enough for some real-world environments, and its accuracy is competitive with a

[1]Some runs stopped explaining early due to inability to maintain plausible explanations

deductive approach. However, its performance is to some degree dependent on a heuristic function that requires more investigation.

The results gathered support our claim of efficiency for MADH. Clearly, there is a computational tradeoff in DEG between accuracy and speed. Given that DEG is considerably slower even when highly inaccurate, MADH's accuracy and speed are a major improvement. While only one domain has so far been tested, it appears clear that a large number of possible actions slows down DEG unreasonably, and we expect to observe similar speed benefits in other partially observable multi-agent domains with large numbers of actions.

**Explanation Before Projection**

*Occurrences*

event1: (GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ROUTE1)

action2: (MOVE MEMBER2 ?DEST57 ROUTE1 600 LOCSTART ?ACT60)

*Precedence*

observation3 $\prec$ event2.

action2 $\prec$ event1.

event1 $\prec$ observation4.

*Constraints*: ?DEST57 != LOCSTART

*Computed Cost*: 10 (assumptions) + 4 (depth) + 6 (event load) = 20

**Explanation After Projection**

*Occurrences*

action2: (MOVE MEMBER2 ?DEST57 ROUTE1 600 LOCSTART ?ACT60)

event1: (GPS-OBSERVE-LOCATION MEMBER2 TEAM1 ROBOT1 UNK-LOCATION ROUTE1)

event2: (HUMAN-SEES MEMBER1 ?ACT60)

event3: (HUMAN-SEES MEMBER3 ?ACT60)

*Precedence*

observation3 $\prec$ action2.

action2 $\prec$ event1.

event1 $\prec$ observation4.

occ(event1) = occ(event2) = occ(event3)

*Constraints*: ?DEST57 != LOCSTART

*Computed Cost*: 10 (assumptions) + 5 (depth) + 6 (event load) = 21

Figure 7.5: Explanation before and after projection with computed explanation costs (ASSUMPTION_COST = 10, EVENT_COST = 6)

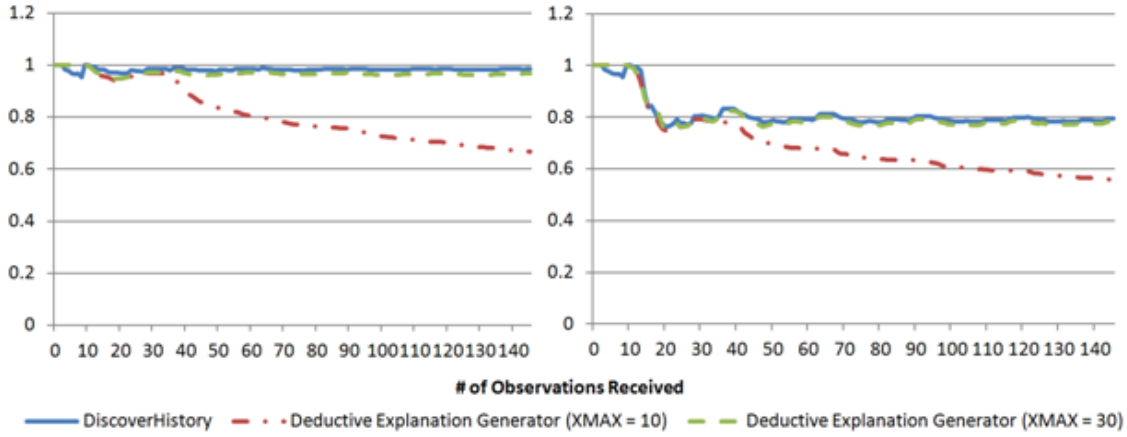


Figure 7.6: Partial Precision (left) and Partial Recall (right) vs. Observation Count (ASM Domain)

# Chapter 8: Learning Unknown Event Models

This chapter concerns the usage of DiscoverHistory in a learning context. We motivate research into explanation-based model learning. We describe the notion of surprise as it relates to an agent, and the implications of surprise for learning. We then describe use of DiscoverHistory search to generate examples for learning, and present a novel technique, Explanation-based Model Learning (EML), for learning models of unknown events that cause surprise. We investigate the performance of EML in a modified version of DHAgent through simulation studies, and report that executing plans based on learned environment models incurs significantly reduced execution cost in comparison to executing plans based on the original environment model. Material in this chapter was previously published at AAAI 2014 (Molineaux and D. W. Aha 2014).

To our knowledge, the problem of learning models for deterministic events has not been discussed by other researchers. Other agents have shown performance in learning transition models; for example, the entire body of reinforcement learning research is based on this idea (Sutton and Barto 1998). In planning research, some systems have been investigated that learn models of actions (e.g., LAMP (Zhuo et al. 2010)). The unique factorization of the transition function we use supports partial learning of exogenous dynamics in a large domain. Other agents that learn a transition model either do not support learning in a relational representation (e.g., reinforcement learning) or do not support domains with hundreds of fluents (e.g., LAMP).

## 8.1 Motivation

Most studies on planning and reasoning assume the availability of a complete and correct domain model, which describes how the environment changes. In this chapter, we relax the

completeness assumption; events occur in our environments that the agent cannot predict or recognize because its model does not describe them. For example, surprises can occur due to incomplete knowledge of events and their locations. In the fictional "The Princess Bride" (W. Goldman 1973), the main characters entered a fire swamp with three types of threats (i.e., flame spurts, lightning sand, and rodents of unusual size) for which they had no prior model. They learned models of each from experience, which they used to predict and defeat future examples. Surprising realistic events can also occur while an agent monitors an environment's changing dynamics. Consider an autonomous underwater vehicle (AUV) that detects an unexpected underwater oil plume for which it has no model. A default response might be to immediately surface (requiring hours) and report it. However, if the AUV first learns a model of the spreading plume, it could react to the projected effects (e.g., by identifying the plume's source).

Surprises in real-world environments can cause failures in robots. Autonomous response to failures would allow them to act for longer periods without oversight. Some surprises can be avoided by increased knowledge engineering, but it is often impractical due to high environment variance, or events unknown to a designer. In these cases, an agent has enough knowledge to construct plans and achieve goals in ordinary circumstances, but surprising events may prevent an agent from succeeding. Therefore, we consider how to learn in response to such surprises. In particular, we employ a novel variant of FOIL (Quinlan 1990), FOIL-PS, to learn models of unknown exogenous events in partially observable, deterministic environments. We employ this learning mechanism in DHAgent to reduce execution cost (a measure of resource expenditure) in two partially observable, partially known domains (see Section 8.5.2.

## 8.2 Claims

In this chapter, we investigate Claim 3 introduced in Chapter 1, repeated here:

**Claim 3**

*Our novel explanation-based model learning algorithm,* EML*, infers successively improved environment models that reduce the execution cost incurred by* DHAGENT *in accomplishing goals in dynamic, partially observable environments.*

EML is designed to solve the hypothesis-based model learning problem introduced in Chapter 1:

- ***Given***: a (possibly incomplete) environment model $M_\Sigma$, a set of transition discontinuities $\mathbf{d}$, observations of the environment $\mathbf{o}$, and an action history $\mathbf{a_h}$,
- ***Find***: a new model $M'_\Sigma$, such that for some hypothesis $h$ with no discontinuities, $h \cup o_0 \cup \mathbf{a_h} \cup M'_\Sigma \models \mathbf{o}$.

In other studies, we chose to measure explanation generation's impact on performance through goal achievement, which we believe is a primary metric of importance for goal-based agents. However, it proved insufficient for a learning study for a simple reason: agents need examples to learn from. Hazardous domains restrict goal achievement percentages by setting traps for the agent, and these same traps restrict the agent's experience of the environment. Therefore, to increase the agent's interaction with the environment, we choose domains and metrics that cause substantial interaction, even when the agent performs poorly. Specifically, execution cost is a general description of how many resources and how much effort are expended by the agent in accomplishing a goal. A poor quality solution requires more resources and effort, and thus requires more interaction with the environment.

As EML constitutes the first attempt at this type of learning, we make several assumptions about the nature of the learning problem which we would like to relax in later work:

1. We assume that the incomplete model omits only events, and is complete with respect to relations, functions, and actions, and the observation function.

2. We assume that the incomplete model is sufficient to create plans for all goals which the agent must satisfy.

Assumption 1 ensures that only events need be the target of learning. Assumption 2 ensures that the agent can learn while attempting to achieve goals, which we would expect to be the case for a realistic agent. The experiment presented in this chapter introduces domains that support these assumptions.

## 8.3 Modeling Surprise

We now define surprise as it affects DHAGENT, in order to clarify our approach. Informally, we say that a generic agent is surprised whenever an observation contradicts its expectations. This is constantly the case in DHAGENT, however; contradiction of expectations in DHAGENT leads to inconsistencies in its explanation, which leads DHAGENT to search for a new, improved explanation. As such, it's able to change its expectation, and understand why that observation occurred. However, in some cases, the only plausible explanations found may include transition discontinuities, because the model does not describe the events that take place. In such cases, DHAGENT does not truly understand what has happened, as the transition discontinuity is standing in for some missing transitions it doesn't know about. Therefore, we say that DHAGENT is surprised when the best explanation found includes transition discontinuities, and we consider the problem of inferring a new model that accounts for those surprises.

## 8.4 Recognizing Unknown Events

The first step in our approach is to recognize when an unknown event, not represented in $M_\Sigma$, has taken place. To do so, DHAGENT searches for a **minimally discontinuous explanation** $\chi_{md}$ that has fewer transition discontinuities than any other. Under the assumption that the simplest explanation is the best, the transition discontinuities in $\chi_{md}$ ought to correspond to unknown events. This search is a best-first search based on a cost function that weighs refinement steps that add transition discontinuities (cost 10) much more heavily than other refinements (cost 1). The first explanation found in a search

(which is, by definition, of minimal cost) is an approximation of the minimally discontinuous explanation $\chi_{md}$. To ensure manageable execution times, we employ a cost bound, set to 50 in our experiments. When this bound is reached, the search returns without solution.

### 8.4.1 Generalizing Event Preconditions

After determining when unknown events occur, creating a model of their preconditions requires generalizing over the states that trigger them. EML uses a novel technique, FOIL-PS (Projected States), which we adapted from the FOIL algorithm (Quinlan 1990) for this purpose. FOIL is useful for learning models because it outputs a set of logical clauses, which can be easily converted into the conditions of an event model. However, FOIL is designed to infer clauses that predict relations in a propositional database; instead, we wish to infer clauses that are satisfied in some projected states and not others. As described in Section 4.3.1, a projected state is a conjunction of literals which are true at a certain time, according to an explanation. These states are found by projecting the effects at each occurrence point of a minimally discontinuous explanation $\chi_{md}$. FOIL-PS infers the conditions of a hypothetical unknown event by finding clauses that are satisfied in projected states that *are* associated with a transition discontinuity (i.e., these projected states could have triggered the unknown event), and are *contradicted* in other projected states. The input to FOIL-PS consists of a set of positive example bags, and a set of negative examples. Each positive example bag contains a collection of states that are projected during a single discontinuity, and different bags are associated with different discontinuities. All other states encountered are negative examples.

EML maintains a set of minimally discontinuous explanations $X$, one for each completed training scenario. To infer events that cause $l$, where $eff(tda) = \{l\}$ for at least one transition discontinuity, it finds a set of bags of projected states that occur between the minimum and maximum times of each discontinuity, as each of those states may have triggered a hypothetical unknown event $e_u$. We define the minimum and maximum discontinuity time

for a transition discontinuity formally as:

$$minimumTime(\chi, tda = \langle \mathsf{discontinuity}, i, v, id \rangle) \equiv min(\{n | tda \in occurs(\chi, n)\})$$
$$maximumTime(\chi, tda = \langle \mathsf{discontinuity}, i, v, id \rangle) \equiv$$
$$min(\{n | n > minimumTime(tda) \wedge \exists o \in occurs(\chi, n) : \mathsf{knownbefore}(i, o, v)\}$$

These state that the minimum time of a transition discontinuity is the time at which it is projected in an explanation, and the maximum time is the time at which the next relevant occurrence is projected.

A positive example bag for a transition discontinuity assumption $tda$, then, is the set of projected states that fall in the interval between these times:

$$\mathsf{peb}(tda) = \{\mathsf{projectedState}(t) \mid minimumTime(tda) < t < maximumTime(tda)\}$$

To learn a model for events that cause $l$, we give FOIL-PS a set of positive example bags corresponding to each transition discontinuity that causes $l$ in all explanations in $X$. The negative examples are all the remaining projected states. Thus the positive example bags for some set of training explanations $X$ and surprising literal $l$ are:

$$\mathsf{peb}(l, X) = \left\{ \mathsf{peb}(tda) \;\middle|\; \begin{array}{r} \exists \chi = \langle O, D, R, C, V \rangle \in X : \\ \exists tda = \langle \mathsf{discontinuity}, i, v, id \rangle \in O : \\ v = target(l) \wedge i = value(l) \end{array} \right\}.$$

The negative examples consist of all other states:

$$\mathsf{ne}(l, X) = \{\mathsf{projectedState}(t) | occurs(t) \neq \emptyset\} \setminus \bigcup \mathsf{peb}(l, X).$$

To find the triggering conditions for an event causing $p$, FOIL-PS searches the space of possible clauses that satisfy zero states in $\mathsf{ne}(p, X)$ and at least one state in each bag in $\mathsf{peb}(p, X)$. The initial clause used is $\{\neg p\}$, and each node in the search tree adds one literal from its parent node.

As this search is costly, FOIL-PS does not consider literals that produce negative information gain according to FOIL's definition. Also, we restrict the number of zero information-gain literals to be added to a clause: FOIL-PS defines the search cost of a clause as the number of zero information gain additions made in the nodes leading to it,

and conducts an iterative deepening search to find only clauses with the minimal search cost. The first clause returned by each level of the search is one that covers the maximum number of positive example bags of any clause found within the search cost. Search repeats with the same cost until sufficient clauses are found to cover all positive example bags. If sufficient clauses cannot be found, search is repeated with an incremented cost.

### 8.4.2 Modifying the Environment Model

Each clause output by FOIL-PS is used to construct a learned event model whose condition is the clause output, and whose effect is the single ground literal believed to be inconsistent. To take an example from the Princess Bride, suppose DHAgent takes the point of the view of a hero, Westley, who observes a distressed damsel, Buttercup, sinking rapidly into the earth. FOIL-PS is given positive example bags including all projected states between the last time Buttercup was *not* sinking, and negative examples consisting of all other projected states Westley can remember. The inconsistent literal is (sinking-rapidly Buttercup). FOIL-PS outputs the clause:

```
(and (not (sinking-rapidly Buttercup))
     (location Buttercup ?loc)
     (sandy-location ?loc)),
```

EML then constructs a new learned event:

```
(:event       new-event51
 :precondition (and (not (sinking-rapidly Buttercup))
                    (location Buttercup ?loc)
                    (sandy-location ?loc))
 :effect      (sinking-rapidly Buttercup))
```

EML adds constructed events to the environment model, which is used for planning and explanation in future scenarios. Ideally, the set of events that are inferred to cause a literal $l$ will match the actual events that cause the condition modeled by $l$. However, FOIL-PS will not always initially find a correct set of models, so EML updates the model periodically, after each scenario is completed.

If the learned event models fail to cover all environment events causing $l$, then $l$ may be found to be inconsistent in a future $\chi_{md}$. When an inconsistent literal is found in the $\chi_{md}$ of the most recent scenario, all previously learned events that cause it are removed from the model and new models are learned from scratch. Conversely, if the learned event models cover situations that do not trigger any event causing $l$, an event will be erroneously predicted, likely resulting in an inconsistent explanation. Thus, we created a DiscoverHistory refinement method, $AbandonModel$ that retracts a previously learned event model, removing it from the inconsistent explanation and marking it as invalid. This refinement incurs less cost (5) than a transition discontinuity assumption, so that it's easier for DiscoverHistory to abandon an event than to add a discontinuity that resolves an inconsistency with a learned event. However, $AbandonModel$ still costs much more than other refinements, to discourage premature abandonment. After an event model is abandoned, the causes of that model's effect must be re-learned (based on all explanations in $X$).

Each of the failures discussed above will cause useful new examples to be added to $\mathsf{peb}(l, X)$ or $\mathsf{ne}(l, X)$ that are not compatible with the current event definition. Therefore, when new learned models cause poor performance, their performance is likely to subsequently improve.

As currently defined, EML cannot acquire exogenous event models with function literals or function effects, and cannot model inequalities or numeric relationships, which we leave for future work. Also, EML cannot acquire models of actions.

## 8.5 Evaluation

We evaluate EML's ability to learn improved environment models, as determined by reduced execution cost incurred by DHAgent using the learned models.

### 8.5.1 Search Configuration

This experiment uses DiscoverHistory as described in Chapter 5. As mentioned earlier, search attempts to find minimally discontinuous explanations by performing a best-first

search based on a cost metric, using DiscoverHistory for expansion. Refinement methods include *AddGround*, *RemoveOcc*, *AssumeInitial*, *IntroduceDiscontinuity*, and the new refinement, *AbandonModel*. Search terminates when a strictly plausible explanation is found or the cost bound (50) is reached. An inconsistency with the fewest child explanations was always chosen for refinement.

### 8.5.2 Environments

This experiment is designed to show that EML can improve the performance of DHAgent when starting from a good, but incomplete, environment model. This has implications for increased robustness in goal-based agents. For this reason, we assume that the agent's model is sufficient to create initial plans in each environment, despite incompleteness. Currently, DHAgent gains examples for learning only through interaction with the environment during the process of attempting to achieve a goal.

Unfortunately, this hampers DHAgent in certain domains, including those we used in prior experiments. In the Hazardous Rovers and Hazardous Satellites domains, DHAgent has only relatively short interactions with the environment when its model is incomplete, because it is unable to avoid trap states from which a goal is unreachable. Instead, for this experiment, we use domains in which an agent will never fail to achieve its goal, but unknown events can reduce the agent's performance. These unknown events increase the execution cost of a plan, and an agent with an incomplete model will not plan to avoid them. However, an improved environment model should allow the agent to generate plans with reduced execution cost. Our study will therefore show that learning is occurring through reduced execution cost.

We tested our EML/DiscoverHistory/DHAgent strategy in two new deterministic, partially observable, single actor domains. While in the future we intend to explore the efficacy of this strategy in larger multi-agent domains with more unknown dynamics, these suffice for a test of learning capability. Each domain contains one event that is not part of the agent's model, and is based on a world state that is not directly observed. While

no explicit learning goals exist, execution cost in each domain is lower when planning with knowledge of the unknown event.

The first domain, *Malfunctioning Satellites*, is based on an International Planning Competition (IPC) 2002 (Long and Fox 2003) domain in which a set of satellites have instruments that can obtain images in many spectra, and goals consist of taking various images. Performance is judged based on the time required to achieve all goals. One time unit is used to turn a satellite to a new position, and 10 to repair a satellite lens. The unknown event causes a satellite's lens to break when taking an image of an excessively bright object. The fact that the object is too bright for the camera lens is hidden to the agent, but bright objects cause an observable lens flare during calibration.

The second domain, *MudWorld*, based on the IPC 2002 Rovers domain, employs a discrete grid on which a simulated robot moves in the four cardinal directions. All goals describe a destination to which the robot must move; in generated test problems, all destinations were chosen such that the shortest path from starting point to destination required at least four steps. The robot observes its location, and its only obstacle is mud. Each location can be muddy or not; the robot cannot observe mud directly, but it deterministically receives a related observation when entering a location adjacent to one that is muddy. If it enters mud, its movement speed is halved until it leaves. However, its initial model does not describe this decrease in speed, so it is surprised when its speed decreases.

In both domains, execution cost is based on time taken in the simulated environment to achieve a goal. Our experiment tests the claim that DHAgent will incur smaller execution costs over time when using improved environment models learned by EML.

### 8.5.3 Experiment Description

For each domain, we randomly generated 50 training and 25 test scenarios. In Malfunctioning Satellites, the initial state of each scenario has 3 satellites with 12 instruments randomly apportioned among them. Each scenario includes 5 goals requiring that an image of a random target be obtained in a random spectrum. MudWorld scenarios consist of a 6x6

Table 8.1: Average execution time, learning time, and explanation failures found in test scenarios after 0-5 learning trials. (**Sat** = Malfunctioning Satellites **Mud** = Mudworld)

| **Learning Trials** | **Execution Time (Seconds)** | | **Learning Time (Seconds)** | | **Explanation Failures** | |
|---|---|---|---|---|---|---|
| | **Sat** | **Mud** | **Sat** | **Mud** | **Sat** | **Mud** |
| 0 | 43.9 | 3.5 | N/A | N/A | 1.80 | 2.24 |
| 1 | 183.3 | 3.8 | 0.16 | 51.8 | 0.93 | 0.78 |
| 2 | 186.5 | 3.5 | 0.28 | 40.5 | 0.57 | 0.23 |
| 3 | 171.8 | 3.5 | 0.24 | 2.3 | 0.41 | 0.14 |
| 4 | 171.5 | 3.3 | 0.18 | 13.9 | 0.41 | 0.03 |
| 5 | 160.1 | 3.3 | 0.21 | 0.7 | 0.36 | 0.03 |

grid with random start and destination locations. Each location may contain mud with 40% probability.

We ran 10 replications per domain, in each of which we used a random set of 5 training scenarios. Before the first training scenario was run, DHAgent's initial performance with the incomplete model was evaluated using each of the 25 test scenarios. Then, after each training scenario, the final explanation found by DHAgent was kept, and added to a set $X$ of known explanations. If any transition discontinuities were found in this explanation, EML was executed to infer a new environment model, which was used in each of 25 test scenarios to evaluate the performance of DHAgent with the new model.

### 8.5.4 Results

Figure 8.1 displays the average execution cost incurred in each domain by DHAgent (blue with triangle markers), an "optimal" version with a complete model (green with circle markers), and a non-learning baseline (red with square markers). The vertical axes depict the simulated time required to complete the test scenarios, while the horizontal axes depict the number of training scenarios provided.

In each domain, DHAgent's performance increased by more than 90% of the difference between its original performance and the optimal performance within 5 trials. After training on only one scenario in each domain, its average performance is significantly higher than

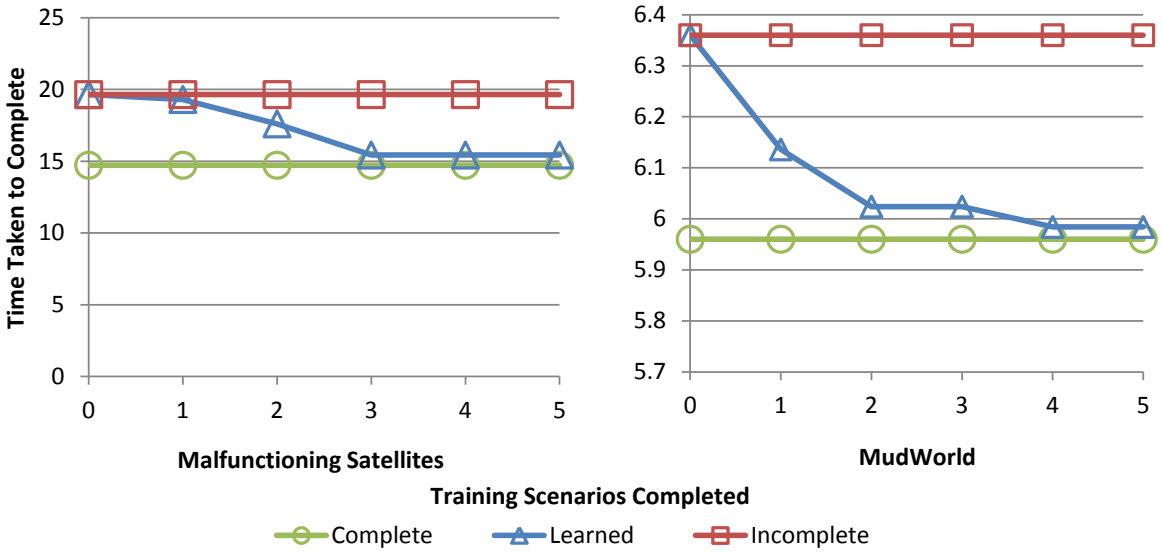


Figure 8.1: Average execution cost incurred by DHAGENT with a complete model (green curve with circles), learned model (blue curve with triangles), and incomplete model (red curve with squares). Lower is better.

when using the initial environment model ($p < .05$).

Table 8.1 shows average wall clock time spent during execution and learning per domain, and the number of explanation failures. The number of explanation failures trends downward, and learning time appears to decrease in MudWorld. However, execution time initially increases in Malfunctioning Satellites. Review of individual trials indicates that this is caused by learning initial models that are inefficient to reason with. We conclude that while explanation and planning clearly improve performance with learning, wall clock time can suffer and is an interesting subject for future study.

### 8.5.5 Discussion

We described a new model learning technique, EML, based on minimally discontinuous explanations found in DISCOVERHISTORY search. We also showed a new version of FOIL, FOIL-PS, which infers concepts based on bags of positive examples. We investigated improvements to execution cost incurred in this environment by DHAGENT when using EML,

and found that DHAgent's performance improved rapidly based on EML's learned environment models, given a small amount of example data.

This experiment supports our claim that EML infers successively improved environment models that reduce the execution cost incurred by DHAgent in dynamic, partially observable environments. While the MudWorld and Malfunctioning Satellites domains are not standard, they are based on typical domains used in planning research, and are supplemented with sufficient new events as to create the desired learning environment. We expect these results to generalize to other problems in which (1) sufficient knowledge is available in an initial incomplete model to construct plans for all goals, (2) all actions and states are represented by the initial incomplete model, and (3) all conditions of unknown events are observable or can be inferred through explanation. The speed of learning in such domains will likely be affected by the length of time between an unknown event and its impact on the agent's observations; our existing experiments have not investigated this effect. Furthermore, these experiments are not sufficient to demonstrate that EML will successfully learn in the presence of multiple unknown events that occur simultaneously.

These conditions are fairly restrictive, and further research is necessary to mitigate these problems. However, we believe that the initial framework and proof of concept for this learning method are promising. While complete knowledge is hard to provide, agents with a sound base model which they can extend safely and autonomously would greatly increase the robustness of modern agents.

# Chapter 9: Conclusions

In this final section, we detail the major novel contributions made, the status of our claims, and future research directions based on limitations of the current work.

## 9.1 Novel Contributions

In this dissertation, we described a number of novel scientific contributions, including:

1. a novel formulation of the goal achievement problem, and three subproblems: continuous planning, hypothesis generation, and hypothesis-based model learning,
2. the new algorithm DiscoverHistory, designed to be used in a search to generate hypotheses,
3. sufficient conditions under which DiscoverHistory search is sound and complete,
4. a new general method for goal achievement, DHAgent, that incorporates DiscoverHistory search,
5. a new general method for hypothesis-based model learning based on DiscoverHistory explanations,
6. experiments investigating claims of performance improvement based on DHAgent, DiscoverHistory, and EML
7. a parameteric analysis of computational efficiency of DiscoverHistory based on two differing implementations

## 9.2 Status of Claims

We presented three claims in Chapter 1, which we investigated in subsequent chapters.

Our first claim stated that belief management using DiscoverHistory search causes the goal achievement performance of DHAgent in a partially observable, single-agent context to improve relative to a replanning agent. In chapter 5, we defined IncuriousAgent, an agent that performs naive belief management, and compared it experimentally with DHAgent, showing that in two partially observable, single-agent domains with a high number of initial states, DHAgent achieved more goals than IncuriousAgent. We also found that the performance gap increased as the likelihood of default assumptions being correct decreased, which supports the notion that DHAgent's improved performance is due to correct explanations. We expect these results to generalize to other partially observable, single-agent domains with a large number of possible initial states, so long as unexpected action outcomes eventually impact the agent's observations.

We also reported on the inability of multiple modern goal-based agents to process problems at the complexity level of our test domains successfully.

Our second claim stated that MADH search, a faster version of DiscoverHistory search for multi-agent environments, is faster than DEG for generating explanations of exogenous actions in a dynamic, partially observable, multi-agent environment, while maintaining comparable accuracy. In Chapter 7, we introduced the deductive explanation generator (DEG), which performs forward search through the space of explanations. This system was designed not to run out of memory, as so many systems seem to do when faced with large relational states. We compared the accuracy and efficiency of explanation generation between DEG and MADH-based search. Results showed a tradeoff between accuracy and speed for DEG, but the MADH search proved faster than DEG, while maintaining equivalent accuracy. We expect to see similar speed improvements and accuracy in other partially observable multi-agent environments with large numbers of possible exogenous actions.

Our third claim stated that our novel explanation-based model learning algorithm,

EML, infers successively improved environment models that reduce the execution cost incurred by DHAgent in accomplishing goals in dynamic, partially observable environments. In Chapter 8, we described a concept of minimally discontinuous explanations that could be found by DiscoverHistory search, and a learning algorithm, EML, that infers new event models based on them. We then investigated a version of DHAgent that uses EML in two partially observable, dynamic environments. Results showed that execution cost incurred by DHAgent was greatly reduced in successive trials based on learning event models with EML, achieving 90% of the maximum possible improvement after only five learning trials. In general, we expect this learning technique to need only a small number of examples of a single surprising event to infer an improved model in a single-agent partially observable environment, provided that the effects of that event are observed quickly. We expect to see decreased performance related to a delay between the event and the observation of its (possibly downstream) effects. Furthermore, multiple simultaneous events might slow the learning process; this needs further investigation. This technique does not yet generalize to learning models depending on relationships and functions not present in its model, or exogenous actions. Furthermore, DHAgent will not generate sufficient learning examples for EML to infer an improved model if the model provided is not complete enough to plan for received goals. Despite all of these restrictions, initial experiments are promising, and we expect future work in this area to yield strong results.

## 9.3 Limitations and Future Work

There are several areas where we see this research continuing in the future. We list some of them here.

**Relaxing Additional Assumptions** An original goal of our work on DiscoverHistory and DHAgent was to be able to explain the effects of continuous processes on the environment based on time-series observations. As yet, the applicability of DiscoverHistory in continuous environments is limited. Noise is also a significant issue; all of our

environments are noise-free, and DiscoverHistory is not currently designed to tolerate noisy observations at all. Finally, DiscoverHistory currently has no means of describing or reasoning about nondeterministic or probabilistic events. These are important areas for general extension in the future.

**Theoretical Results** We have assumed that the order in which inconsistencies are refined does not affect reachability of explanations. While this seems intuitive, it has yet to be proven, and almost certainly depends on the specific refinement methods in use. A theoretical result describing what refinement methods allow inconsistencies to be resolved in any order would help in understanding the guarantees that can be made in explanation.

Furthermore, we have not yet conducted an analysis of the computational complexity of DHAgent, DiscoverHistory search, or EML. Such an analysis would help the larger community to understand the properties of the algorithm better.

**Experimentation with Search Heuristics** In Chapter 7, we introduced a complex heuristic based on five different metrics that could be explored further. There may be a large space of possible explanation cost metrics, and different weighted combinations may be superior in different environments. A large parameter study is needed to understand the impact of these metrics.

**Learning Extensions** Learning with EML assumes that an explanation failure is due to an unknown event(s); as a result, it may incorrectly infer a transition discontinuity when the true explanation is not found (e.g., due to computational constraints). Conversely, search may sometimes find an incorrect explanation when ambiguous observations are received, causing it to ignore an opportunity for learning. Reducing these false positives and false negatives is a future research topic.

Extending EML to learning general environment model components, including action models and process models, as well as inferring the presence of latent relationships, is a long-term goal.

**Narrative Generation** We anticipate that occurrence histories could form the basis of an agent capability to construct a narrative. Humans often communicate in this fashion, describing our experiences and justifying our actions in terms of sequences of events that are temporally and causally related. The capability to do so is called **narrative intelligence** by some in the artificial intelligence community (Blair and Meyer 1997; Mateas and Sengers 1999; Mateas and Sengers 2003; Riedl 2004), and **narrative dialogue** in the psychological literature, where the formation of the capability in young children has been identified and studied (Nelson 1993). While an occurrence history does not have all the qualities of narrative, it does meet the basic requirements of a narrative **fabula**, described by Riedl (2004) as the "events of the story world that make up the content of the narrated material". To make a true narrative, this must be layered with a **sjužet**, or "discourse that relates some of the events in the fabula", which we do not approach but believe is enabled by the construction of the fabula. By constructing occurrence histories, we describe the world in a form that humans naturally use and are comfortable with. Therefore, an agent that generates these should be better able to communicate with a human than one that does not. We have not yet studied this, but if successful, this would argue for centrality of explanation generation in many agent systems.

# Bibliography


Aberdeen, D. & Buffet, O. (2007). Concurrent probabilistic temporal planning with policy-gradients. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS'07)*. Providence, RI.

Albore, A., Palacios, H., & Geffner, H. (2009). A translation-based approach to contingent planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence* (pp. 1623–1628).

Alchourrón, C. E., Gärdenfors, P., & Makinson, D. (1985). On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 510–530.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, *23*, 123–154.

Ambros-Ingerson, J. & Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 83–88).

Amir, E. & Russell, S. (2003). Logical filtering. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vol. 18, pp. 75–82). Lawrence Erlbaum Associates Ltd.

Åström, K. J. (1965). Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, *10*(1), 174–205.

Ayan, F., Kuter, U., Yaman, F., & Goldman, R. P. (2007). HOTRiDE: hierarchical ordered task replanning in dynamic environments. In F. Ingrand & K. Rajan (Eds.), *ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems* (pp. 31–36).

Baral, C., Kreinovich, V., & Trejo, R. (2000). Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, *122*(1), 241–267.

Baral, C. & Son, T. C. (1997). Approximate reasoning about actions in presence of sensing and incomplete information. In *Proceedings of the International Logic Programming Symposium (ILPS 97)* (pp. 387–401).

Beetz, M. & McDermott, D. (1994). Improving robot plans during their execution. In *Proceedings of the Second International Conference on AI Planning Systems* (pp. 3–12).

Bernard, D., Dorais, G., Fry, C., Jr, E., Kanefsky, B., Kurien, J., ... Williams, B. (1998). Design of the remote agent experiment for spacecraft autonomy. In *Proceedings of the IEEE Aerospace Conference*.

Bertoli, P. & Cimatti, A. (2002). Improving heuristics for planning as search in belief space. In *Artificial Intelligence Planning Systems* (pp. 143–152).

Blair, D. & Meyer, T. (1997). Tools for an interactive virtual cinema. *Creating Personalities for Synthetic Actors*, 83–91.

Bonet, B. & Geffner, H. (2012). Width and complexity of belief tracking in non-deterministic conformant and contingent planning. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Boutilier, C. (1995). Generalized update: Belief change in dynamic settings. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vol. 14, pp. 1550–1556). Lawrence Erlbaum Associates Ltd.

Boutilier, C. (1998). A unified model of qualitative belief change: A dynamical systems perspective. *Artificial Intelligence*, *98*(1), 281–316.

Boutilier, C. & Becher, V. (1995). Abduction as belief revision. *Artificial Intelligence*, *77*(1), 43–94.

Bridewell, W., Langley, P., Todorovski, L., & Džeroski, S. (2008). Inductive process modeling. *Machine Learning*, *71*(1), 1–32.

Bridewell, W. & Langley, P. (2011). A computational account of everyday abductive inference. In *Proceedings of the Thirty-Third Annual Meeting of the Cognitive Science Society* (pp. 2289–2294).

Bruner, J. (1991). The narrative construction of reality. *Critical Inquiry*, *18*(1), 1–21.

Bryce, D., Kambhampati, S., & Smith, D. (2004). Planning in belief space with a labelled uncertainty graph. In *AAAI Workshop on Learning and Planning in Markov Decision Processes*.

Bryce, D., Kambhampati, S., & Smith, D. (2006). Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, *26*(1), 35–99.

Chernova, S., Crawford, E., & Veloso, M. (2005). Acquiring observation models through reverse plan monitoring. In *Progress in Artificial Intelligence* (pp. 410–421). Springer.

Choi, J., Guzman-Rivera, A., & Amir, E. (2011). Lifted relational kalman filtering. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence* (pp. 2092–2099). AAAI Press.

Console, L. & Torasso, P. (1990). Integrating models of the correct behavior into abductive diagnosis. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)* (pp. 160–166).

Cordier, M. & Thiébaux, S. (1994). Event-based diagnosis for evolutive systems. In *Proceedings of the Fifth International Workshop on Principles of Diagnosis (DX-94)* (pp. 64–69).

Darwiche, A. & Pearl, J. (1997). On the logic of iterated belief revision. *Artificial Intelligence*, *89*(1), 1–29.

Davis, R. (1984). Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, *24*(1), 347–410.

De Giacomo, G. & Levesque, H. J. (1999). Projection using regression and sensors. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)* (Vol. 1, p. 160). Morgan Kaufmann Publishers. Stockholm, Sweden.

De Kleer, J. (1976). *Local methods for localizing faults in electronic circuits*. Massachusetts Institute of Technology.

De Kleer, J. & Williams, B. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*(1), 97–130.

DeJong, G. & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, *1*(2), 145–176.

Del Val, A. & Shoham, Y. (1994). Deriving properties of belief update from theories of action. *Journal of Logic, Language and Information*, *3*(2), 81–119.

desJardins, M. E., Durfee, E. H., Ortiz Jr, C. L., & Wolverton, M. J. (1999). Survey of research in distributed, continual planning. *AI Magazine*, *20*(4), 13–22.

Dupin de Saint-Cyr, F. & Lang, J. (2002). Belief extrapolation (or how to reason about observations and unpredicted change). In *International Conference on Principles of Knowledge Representation and Reasoning* (pp. 497–508).

Dupin de Saint-Cyr, F. & Lang, J. (2011). Belief extrapolation (or how to reason about observations and unpredicted change). *Artificial Intelligence*, *175*(2), 760–790.

Dvorak, D. & Kuipers, B. (1989). Model-based monitoring of dynamic systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (Vol. 2, pp. 1238–1243). Morgan Kaufmann Publishers Inc.

Eco, U. (1983). Horns, hooves, insteps: Some hypotheses on three types of abduction. In *The Sign of Three: Dupin, Holmes, Peirce* (pp. 198–220). Indiana University Press.

Eiter, T., Erdem, E., Faber, W., & Senko, J. (2007). A logic-based approach to finding explanations for discrepancies in optimistic plan execution. *Fundamenta Informaticae*, *79*(1), 25–69.

Erol, K., Hendler, J., & Nau, D. S. (1994). HTN Planning: Complexity and Expressivity. In *Proceedings of the National Conference on Artificial Intelligence*. Retrieved from http://www.cs.umd.edu/~nau/papers/aaai94.ps

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*(4), 251–288.

Forbus, K. & Falkenhainer, B. (1990). Self-explanatory simulations: An integration of qualitative and quantitative knowledge. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 380–387).

Fox, M. & Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, *27*(1), 235–297.

Fox, M. & Long, D. (2002). PDDL+: Modelling continuous time-dependent effects. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*.

Fox, M. & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)*, *20*, 61–124.

Friedman, N. & Halpern, J. Y. (1994). A knowledge-based framework for belief change. Part II: revision and update. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR'94)* (pp. 190–201). Morgan Kaufmann Publishers. San Francisco, CA.

Friedrich, G. & Lackinger, F. (1991). Diagnosing temporal misbehavior. In *IJCAI* (pp. 1116–1122).

Gamper, J. (1996). *A temporal reasoning and abstraction framework for model-based diagnosis systems* (Doctoral dissertation, Rheinisch-Westfalsche Technische Hochschule Aachen).

Genesereth, M. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence*, *24*(1), 411–436.

Gerevini, A. & Long, D. (2006). Preferences and soft constraints in PDDL3. In *ICAPS Workshop on Planning with Preferences and Soft Constraints* (pp. 46–53).

Giacomo, G., Lespérance, Y., Levesque, H. J., & Sardina, S. (2009). IndiGolog: A high-level programming language for embedded reasoning agents. *Multi-Agent Programming:* 31–72.

Göbelbecker, M., Gretton, C., & Dearden, R. (2011). A switching planner for combined task and observation planning. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI-11)*.

Golden, K. & Weld, D. (1996). Representing sensing actions: The middle ground revisited. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)* (pp. 174–185). Morgan Kaufmann Publishers.

Goldman, R., Haigh, K., Musliner, D., & Pelican, M. (2002). MACBETH: a multi-agent constraint-based planner. In *Proceedings of the 21st Digital Avionics Systems Conference* (Vol. 2). IEEE.

Goldman, W. (1973). *The Princess Bride*. Pan Macmillan. Retrieved from https://books.google.com/books?id=IIx0PgAACAAJ

Grastien, A., Anbulagan, A., Rintanen, J., & Kelareva, E. (2007). Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 22, *1*, p. 305). Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Gspandl, S., Pill, I., Reip, M., Steinbauer, G., & Ferrein, A. (2011). Belief Management for High-Level Robot Programs. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*.

Hammond, K. J. (1990). Explaining and Repairing Plans That Fail. *Artificial Intelligence*, *45*, 173–228.

Hamscher, W. & Davis, R. (1984). Diagnosing circuits with state: An inherently underconstrained problem. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 84, pp. 142–147).

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, *173*(5-6), 503–535.

Hiatt, L. M., Khemlani, S. S., & Trafton, J. G. (2012). An explanatory reasoning framework for embodied agents. *Biologically Inspired Cognitive Architectures*, *1*, 23–31.

Hoffmann, J. & Brafman, R. (2005). Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)* (pp. 71–80).

Iwan, G. (2001). History-based diagnosis templates in the framework of the situation calculus. *KI 2001: Advances in Artificial Intelligence*, 244–259.

Josephson, J. & Josephson, S. (1996). *Abductive Inference: Computation, Philosophy, Technology*. Cambridge University Press.

Kabanza, F., Barbeau, M., & St-Denis, R. (1997). Planning control rules for reactive agents. *Artificial Intelligence*, *95*(1), 67–113.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1), 99–134.

Kambhampati, S. & A., J. H. (1992). A validation structure based theory of plan modification and reuse. *Artificial Intelligence*, *55*, 193–258.

Katsuno, H. & Mendelzon, A. O. (1991). On the difference between updating a knowledge base and revising it. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning* (pp. 387–394).

Klenk, M., Molineaux, M., & Aha, D. W. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, *29*(2), 187–206.

Knight, R., Rabideau, G., Chien, S., Engelhardt, B., & Sherwood, R. (2001). CASPER: Space exploration through continuous planning. *IEEE Intelligent System*, 70–75.

Kraft, D., Baseski, E., Popovic, M., Batog, A. M., Kjær-Nielsen, A., Krüger, N., ... Steedman, M. (2008). Exploration and planning in a three-level cognitive architecture. In *Proceedings of the International Conference on Cognitive Systems (CogSys)*.

Lang, J. (2007). Belief update revisited. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (pp. 2517–2522).

Leake, D. B. (1991). Goal-based explanation evaluation. *Cognitive Science*, *15*(4), 509–545.

Leake, D. B. (1995). Toward goal-driven integration of explanation and action. *Goal-Driven Learning*, 455.

Levesque, H. J. (1989). A knowledge-level account of abduction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (Vol. 2, pp. 1061–1067). Morgan Kaufmann Publishers.

Levesque, H. J. & Brachman, R. J. (1984). *A Fundamental Tradeoff in Knowledge Representation and Reasoning*. Laboratory for Artificial Intelligence Research, Fairchild, Schlumberger.

Liberatore, P. (2000). A framework for belief update. *Logics in Artificial Intelligence*, 361–375.

Liu, Y. & Levesque, H. J. (2005). Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vol. 19, p. 522).

Long, D. & Fox, M. (2003). The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res.(JAIR)*, *20*, 1–59.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, *28*(1), 47–65.

Mateas, M. & Sengers, P. (1999). Narrative intelligence. In *Proceedings of the AAAI Fall Symposium on Narrative Intelligence* (pp. 1–10).

Mateas, M. & Sengers, P. (Eds.). (2003). *Narrative Intelligence*. Amsterdam: J. Benjamins Pub.

McCarthy, J. & Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, *4*, 463–502.

McDermott, D. M. (2000). The 1998 AI planning systems competition. *AI Magazine*, *21*(2), 35.

McIlraith, S. A. (1998). Explanatory diagnosis: Conjecturing actions to explain observations. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* (pp. 167–179). Morgan Kaufmann Publishers.

Meadows, B. L., Langley, P., & Emery, M. J. (2013). Seeing beyond shadows: Incremental abductive reasoning for plan understanding. In *AAAI Workshop: Plan, Activity, and Intent Recognition* (Vol. 13, p. 13).

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

Molineaux, M., Klenk, M., & Aha, D. (2010a). Planning in dynamic environments: Extending HTNs with nonlinear continuous effects. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Molineaux, M., Klenk, M., & Aha, D. (2010b). Goal-driven autonomy in a Navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (pp. 1548–1554).

Molineaux, M. & Aha, D. W. (2014). Learning unknown event models. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press.

Molineaux, M. & Aha, D. W. (2015). Continuous explanation generation in a multi-agent domain. In *Proceedings of the Third Annual Conference on Advances in Cognitive Systems* (p. 1). Atlanta, Georgia.

Molineaux, M., Kuter, U., & Klenk, M. (2012). DiscoverHistory: Understanding the past in planning and execution. In *Proceedings of the 11th International Conference on*

*Autonomous Agents and Multiagent Systems, Volume 2* (pp. 989–996). International Foundation for Autonomous Agents and Multiagent Systems.

Morgenstern, L. & Stein, L. A. (1988). Why things go wrong: A formal theory of causal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence* (p. 518). AAAI Press.

Mourao, K., Zettlemoyer, L. S., Petrick, R., & Steedman, M. (2012). Learning strips operators from noisy and incomplete observations. *ArXiv Preprint ArXiv:1210.4889*.

Musliner, D. J., Durfee, E. H., & Shin, K. G. (1993). CIRCA: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, *23*(6), 1561–1574.

Musliner, D. J., Pelican, M. J. S., Goldman, R. P., Krebsbach, K. D., & Durfee, E. H. (2008). The evolution of CIRCA, a theory-based AI architecture with real-time performance guarantees. In *Proceedings of the AAAI Spring Symposium on Architectures for Intelligent Theory-Based Agents*.

Myers, K. L. (1996). Advisable planning systems. In A. Tate (Ed.), *Advanced Planning Technology*. AAAI Press.

Myers, K. L. (1999). A continuous planning and execution framework. *AI Magazine*, *20*(4), 63.

Nelson, K. (1993). Events, narratives, memory: What develops. *Memory and Affect in Development*, *26*, 1–24.

Nguyen, T.-H. D. & Leong, T.-Y. (2009). A surprise triggered adaptive and reactive (STAR) framework for online adaptation in non-stationary environments. In *AIIDE*.

Palacios, H. & Geffner, H. (2009). Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, *35*(1).

Pang, W. & Coghill, G. M. (2010). Learning qualitative differential equation models: a survey of algorithms and applications. *The Knowledge Engineering Review*, *25*(01), 69–107.

Pasula, H. M., Zettlemoyer, L. S., & Kaelbling, L. P. (2007). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 309–352.

Patkos, T. & Plexousakis, D. (2009). Reasoning with knowledge, action and time in dynamic and uncertain domains. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vol. 9, pp. 885–890).

Pednault, E. P. (1987). *Toward a mathematical theory of plan synthesis* (Doctoral dissertation, Stanford University, Stanford, CA, USA).

Pednault, E. P. (1988). Extending conventional planning techniques to handle actions with context-dependent effects. In *Proceedings of the National Conference on Artificial Intelligence*.

Pednault, E. P. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (pp. 324–332). Morgan Kaufmann Publishers.

Petrick, R. & Bacchus, F. (2002). A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*.

Pinto, J. A. (1994). *Temporal reasoning in the situation calculus* (Doctoral dissertation, University of Toronto).

Pinto, J. A. (1997). Integrating discrete and continuous change in a logical framework. *Computational Intelligence*, *14*(1), 2–13.

Pople, H. (1973). On the mechanization of abductive logic. In *Proceedings of the Third International Joint Conference on Artificial Intelligence* (pp. 147–152).

Powell, J., Molineaux, M., & Aha, D. (2011). Active and interactive learning of goal selection knowledge. In *Proceedings of the Twenty-Fourth Florida Artificial Intelligence Research Society Conference*.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Ram, A. (1993). Indexing, elaboration and refinement: Incremental learning of explanatory cases. In *Case-Based Learning* (pp. 7–54). Springer.

Ranasinghe, N. & Shen, W. (2008). Surprise-based learning for developmental robotics. In *Learning and Adaptive Behaviors for Robotic Systems, 2008. LAB-RS'08. ECSIS Symposium on* (pp. 65–70). IEEE.

Reggia, J. (1978). A production rule system for neurological localization. In *Proceedings of the Annual Symposium on Computer Application in Medical Care* (p. 254). American Medical Informatics Association.

Reggia, J., Nau, D., & Wang, P. (1983). Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, *19*(5), 437–460.

Reiter, R. (1987a). A theory of diagnosis from first principles. *Artificial Intelligence*, *32*(1), 57–95.

Reiter, R. (1987b). On closed world data bases. In *Readings in Nonmonotonic Reasoning* (pp. 300–310). Morgan Kaufmann Publishers.

Reiter, R. (1996). Natural actions, concurrency and continuous time in the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* (pp. 2–13). Morgan Kaufmann Publishers.

Riedl, M. O. (2004). *Narrative Planning: Balancing Plot and Character* (Doctoral dissertation, North Carolina State University).

Robust. (N.d.). https://www.merriam-webster.com/dictionary/robust. Retrieved March 26, 2017.

Rose, D. & Langley, P. (1986). Chemical discovery as belief revision. *Machine Learning*, *1*(4), 423–452. doi:10.1023/A:1022870800276

Russell, S. & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd). Upper Saddle River, NJ, USA: Prentice Hall Press.

Sanner, S. & Boutilier, C. (2009). Practical solution techniques for first-order MDPs. *Artificial Intelligence*, *173*(5-6), 748–788.

Schoppers, M. (1987). Universal plans for reactive robots in unpredictable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 1039–1046).

Schum, D. A. (2001). Species of abductive reasoning in fact investigation in law. *Cardozo Law Review*, 1645–1682.

Shahaf, D. (2007). *Logical filtering and learning in partially observable worlds* (Master's thesis, University of Illinois at Urbana-Champaign).

Shanahan, M. (1993). Explanation in the situation calculus. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vol. 13). Lawence Erlbaum Associates.

Shanahan, M. (1996). Robotics and the common sense informatic situation. In *Proceedings of the European Conference on Artificial Intelligence* (pp. 684–688). Pitman.

Shanahan, M. (2000). Reinventing shakey. In *Logic-Based Artificial Intelligence* (pp. 233–253). Springer.

Shanahan, M. & Witkowski, M. (2000). High-level robot control through logic. In *International Workshop on Agent Theories, Architectures, and Languages* (pp. 104–121). Springer.

Shani, G. & Brafman, R. (2011). Replanning in domains with partial information and sensing actions. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Simmons, R. & Davis, R. (1987). Generate, test and debug: Combining associational rules and causal models. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence* (pp. 1071–1078). Milan, Italy.

Sohrabi, S., Baier, J., & McIlraith, S. (2010). Diagnosis as planning revisited. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning* (pp. 26–36).

Son, T. C. & Baral, C. (2001). Formalizing sensing actions: a transition function based approach. *Artificial Intelligence*, *125*(1), 19–91.

Sooriamurthi, R. & Leake, D. B. (1995). An architecture for goal-driven explanation. In *Proceedings of the Eighth Florida Artificial Intelligence Research Symposium* (pp. 218–222). Eckerd College.

Sutton, R. & Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Talamadupula, K., Benton, J., Kambhampati, S., Schermerhorn, P., & Scheutz, M. (2010). Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *1*(2), 14.

Thagard, P. R. (1993). *Computational Philosophy of Science*. MIT Press.

Thielscher, M. (1997). A theory of dynamic diagnosis. *Linkoping Electronic Articles in Computer and Information Science*, *2*(11).

Vassos, S. & Levesque, H. (2007). Progression of situation calculus action theories with incomplete information. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vol. 7).

Verfaillie, G. & Schiex, T. (1994). Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence* (pp. 307–312).

Wang, X. & Chien, S. (1997). Replanning using hierarchical task network and operator-based planning. *Recent Advances in AI Planning*, 427–439.

Warfield, I., Hogg, C., Lee-Urban, S., & Munoz-Avila, H. (2007). Adaptation of hierarchical task network plans. In *Proceedings of the Twentieth International FLAIRS Conference (FLAIRS-07)*.

Weber, B. G., Mateas, M., & Jhala, A. (2012). Learning from demonstration for goal-driven autonomy. In *Proceedings of the National Conference on Artificial Intelligence*.

Webster, G. (2005). NASA's rovers continue martian missions. Retrieved from http://marsrover.nasa.gov/newsroom/pressreleases/20050524a.html

Weld, D. S., Anderson, C. R., & Smith, D. E. (1998). Extending graphplan to handle uncertainty & sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (pp. 897–904). AAAI Press.

Wilkins, D. E. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA: Morgan Kaufmann.

Wilson, M., Molineaux, M., & Aha, D. W. (2013). Domain-independent heuristics for goal formulation. In *Twenty-Sixth International Florida Artificial Intelligence Research Society Conference*. AAAI Press.

Wyatt, J. L., Aydemir, A., Brenner, M., Hanheide, M., Hawes, N., Jensfelt, P., ... Pronobis, A. (2010). Self-understanding and self-extension: A systems and representational approach. *IEEE Transactions on Autonomous Mental Development*, *2*(4), 282–303.

Yoon, S., Fern, A., Givan, R., & Kambhampati, S. (2008). Probabilistic planning via determinization in hindsight. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence* (Vol. 2).

Yoon, S., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling* (pp. 352–359).

Zhuo, H. H., Hu, D. H., Hogg, C., Yang, Q., & Munoz-Avila, H. (2009). Learning HTN method preconditions and action models from partial observations. In *IJCAI* (pp. 1804–1810).

Zhuo, H. H., Yang, Q., Hu, D. H., & Li, L. (2010). Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, *174*(18), 1540–1569.

# Biography

Matthew Molineaux received a Bachelor of Science from Eckerd College in St. Petersburg, Florida in 2001. He received a Master of Science in Computer Science from Depaul University in Chicago, Illinois in 2008. From 2003 to 2007, he worked for ITT as a research scientist, supporting the Naval Research Laboratory AI Center in Washington, DC. From 2007 to today, he has worked as a Senior Computer Scientist at Knexus Research Corporation. In this position, he has served as Team Lead and Principal Investigator on various internal projects, and also supported the Naval Research Laboratory AI Center.

**Selected Papers:**


Molineaux, M. and Aha, D. W. (2015). Continuous explanation generation in a multi-agent domain. In *Proceedings of the Third Annual Conference on Advances in Cognitive Systems* (p. 1). Atlanta, Georgia.

Molineaux, M. and Aha, D. W. (2014). Learning unknown event models. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press.

Klenk, M., Molineaux, M., and Aha, D. W. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, *29*(2), 187–206.

Wilson, M., Molineaux, M., and Aha, D. W. (2013). Domain-independent heuristics for goal formulation. *Twenty-Sixth International Florida Artificial Intelligence Research Society Conference*. AAAI Press.

Molineaux, M., Kuter, U., and Klenk, M. (2012). DiscoverHistory: Understanding the past in planning and execution. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, Volume 2* (pp. 989–996). International Foundation for Autonomous Agents and Multiagent Systems.

Powell, J., Molineaux, M., and Aha, D. (2011). Active and interactive learning of goal selection knowledge. *Proceedings of the Twenty-Fourth Florida Artificial Intelligence Research Society Conference*.

Molineaux, M., Klenk, M., and Aha, D. (2010b). Goal-driven autonomy in a Navy strategy simulation. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (pp. 1548–1554).

Molineaux, M., Klenk, M., and Aha, D. (2010a). Planning in dynamic environments: Extending HTNs with nonlinear continuous effects. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.