Cognitive Autonomous Robots and Agents: Specification and Implementation (Theories of Knowledge, Action, Change and Evolution)

Chitta Baral

Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85233, USA
chitta@asu.edu

Tran Cao Son

Department of Computer Science
New Mexico State University
PO Box 30001, MSC CS
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

October 13, 2008

Contents

	Term	Usage
Axiom	fluent effect axiom	a **causes** l **if** φ
	executability axiom	**executable** a **if** φ
	knowledge effect axiom	a **determines** φ
	static causal axiom	φ **s_causes** l
Action specification	effect axiom	fluent effect axiom
		knowledge effect axiom
	executability axiom	
Specification	domain specification	set of action specifications and static causal axioms
	(current) observation	**initially** l
	(past) observation	α **occurs_at** $\mathtt{s}$, **between** $\mathtt{s}_1, \mathtt{s}_2$, etc.
Action theory	(D, O)	D is a domain specification and O is a set of current observations
Narrative	(D, Γ)	D is a domain speicification and O is a set of past observations
Notation/Symbols	$[\ldots]$	list notation — denoting sequence of actions
	$\circ$	concatenation operator
	σ	set of fluents (state in the language $\mathcal{A}$)
	s	state (complete set of fluent literals sattisfying all static causal axioms, state in $\mathcal{B}$)
	$\langle\ldots\rangle$	combined state

Table 1: Frequently used terminologies

Chapter 1

Introduction

> Destiny is no matter of chance. It is a matter of choice. It is not a thing to be waited for, it is a thing to be achieved.
>
> William Jennings Bryan (1860 - 1925)

1.1 Introduction

Plato in Cratylus (402a) attributes Heracleitus (540 BC-480 BC) to the doctrine that all things are in constant flux. While constant change and the resulting evolution of the world is indeed universal, autonomous agents that have free will (such as humans) try to control certain aspects of this change to steer the evolution in a desired way.

In Computer Science, the evolving world manifests itself in many ways, such as the changing environment–values in memory locations and registers–of a running program, the data in a dynamic database, the physical environment of a robot, the environment of an autonomous software agent and the environment of the virtual model of a cell. Agents with autonomy and free will[1] also manifest in the above environments. A dynamic database with active components have active rules that are automatically triggered when certain updates are made so as to steer the change in the database in a particular way. Although these active rules are written by human beings who are outside of the environment, one could develop programs which would construct active rules for a given specification of the desired evolution of the database. Similarly, a robot or an agent may be given a goal to control the trajectory of their world in a particular way and the robot or agent acts and reacts (after first figuring out on its own what actions to take and when) to the environment to achieve the given goal. With respect to the virtual model of a cell, diseases often manifest as cells behaving in undesired ways (i.e., the environment inside the cells evolve in undesired ways) and designing a drug or therapy involves coming up with a plan of intervention that will make cells behave in a desired way.

The goal of this book is to present the necessary theories, architectures and algorithms that will allow us to design and develop agents with autonomy and free will. In particular, we would like to develop programs that can automatically generate active rules or verify the correctness of a given set of active rules corresponding to a particular specification of how the database should evolve; We would like to develop agents and robots that can themselves verify or construct their control so as to make the trajectory of their environment evolve in a particular way; and we would like to develop programs that can construct drugs and therapies that make the cell behave in a desired way.

[1] There are some restrictions to the free will and autonomy of these artificially created agents. Their set of actions is predesigned and they are given a goal. This is necessary to rule out the possibility that they might enslave humans.

Let us now analyze the above tasks to better understand the various topics that will be covered in this book. For an agent to come up with its own plan of action or to reason about the impact any action that it may execute, the agent must have the knowledge about what it can do (i.e., what actions it can execute) and when and what the impact of those actions would be on the world; the agent must also know how the world may evolve even without its interference. Thus we need to develop languages to express effect of actions on the environment. A challenging aspect of this is that the language should allow succinct representation, as a direct representation of change in the state of the world due to actions could be exponential in the size of the various properties of the world. Next, for an agent to take a directive we must develop languages that can express the directives that state which particular evolutions of the world are desired. Now for the agent to succeed in controlling the evolution of the world as desired it must be able to observe the world[2], and reason about its evolution in both past and future directions. Reasoning about the past is especially helpful when the agent can only partially observe and has to reason to make additional conclusions about the past which it could not directly observe. This is helpful because, the more the agent knows about the world the better the chances that it can come up with ways to control the evolution in the desired way.

We now further elaborate on various technical formulations, and algorithms that are needed to be able to characterize and correctly implement the various abilities mentioned in the previous paragraph.

1.2 Specification and Implementation: the big picture

We start with specifying the actions of an agent its impact on the world.

1.2.1 Specifying actions and their effects

To start with we will need to specify what actions an agent can execute in what state of the world, and what its impact on the world would be. Since the number of states of the world is often exponential in the size of the properties of the world, we can not just explicitly specify a table that will tell us the transition between states of the world due to actions. A possible candidate for an implicit way of doing that would be to specify the impact of actions on properties of the world (also referred to as a "fluent") and from that derive the transition between states due to actions. This is not as easy as it may appear. Several issues make it difficult. For example:

1. Do we explicitly specify the impact of actions on each fluent regardless of whether the action 'changes' the truth of the fluent or not? Perhaps its more succinct to explicitly specify when an action "changes" a fluent and infer the non-change by some kind of default reasoning.

2. How do we specify when the impact of an action on a fluent depends on the state where it is executed? How do we express such conditional effects? Suppose an action has the same impact on a fluent when it is executed in a particular set of states. Do we resort to a logical specification of those set of states?

3. How do we specify actions with non-deterministic impacts on fluents? For example, suppose an action a may either make f true or make g true. In such a case how do we make correct reasoning about the effect of action a. Suppose initial f is true and a is executed. Could g be true in the resulting state?

4. Often actions have direct impact on some particular fluents, while because of certain connectedness between fluents, the action may have indirect impact on other fluents. For example, the action of

[2]In case of robots in physical worlds and nano agents monitoring cells, observation corresponds to sensing or perception leading us to cognitive autonomous agents.

lifting a cup has the direct impact that the cup gets lifted. But at the same time, any object inside the cup also gets lifted. Should we analyze all such connections and explicitly specify the impact of an action on all fluents? Or should we only explicitly specify the direct impacts, and the connection between the fluents, and have mechanisms to infer the indirect effects? If the later, what kind of inference mechanisms are necessary for inferring indirect effects?

5. If we start using logical specifications to club sets of states together, what kind of overall specification will we have. If we use a complicated logical language as our specification language then it may become difficult for domain experts (who may not be experienced in logic) to specify actions and their effects.

6. Domain experts will have similar difficulty if one were to use non-straight-forward data structures (that may lead to efficient reasoning) for the representation.

To recap, the main concern above is to be able to succinctly specify the transition between states due to actions, without needing specialized training, and have reasoning mechanisms that allows us to reason about the transition graph without explicitly constructing it.

1.2.2 Specifying the evolution of the world

The transition function discussed in the previous section only tells us the transition between states due to a single action. Although this information is important, it is not always sufficient to predict or analyze how the world will evolve. In particular, when actions, or a particular state of the world may trigger another action, when other agents may react to actions executed by our agent, and when natural actions may happen. We need to be able to specify such reactions and triggering and use it in reasoning about the evolution of the world.

As before we can not explicitly specify all possible ways the world will evolve. Some of the non-trivial issues encountered in a succinct specification of evolution through triggers and reactions involved are as follows:

1. Capturing the notion of triggers where the triggered actions do not have to necessarily happen immediately after the trigger conditions become true.

2. When multiple actions are triggered there may be ordering conditions between them.

3. Certain triggered actions may be inhibited by other conditions.

1.2.3 Specifying observations and figuring out the current state of the world

The specifications in the previous two sub-sections tell us the transition between states due to actions and the possible evolution of the world. When an agent is situated in a world, it may have complete information (in terms of which fluents[3] are true and which are false) about a particular initial state. But after that the agent executes some actions, other actions are executed by other agents or the environment, and the agent may observe only some of the happenings and may have sensed some of the fluent values at some time points. Based on its partial knowledge the agent needs to periodically figure out its current state so as to evaluate if

[3]By fluents we mean dynamic properties of the world.

the world is evolving the way the agent desired it to and if not the agent can then try to change its course of action to steer the evolution to a desired path.

Here the specification of various kinds of observations is syntactically simple, but analyzing these observations to narrow down the current state, and how the world progressed to it (from the initial state) is challenging. Some of the challenging issues are:

1. In the absence of direct observations regarding action occurrences figuring out what might have happened between two time points.

2. Eliminating plausible but highly improbable trajectories by minimizing unforced (directly or indirectly via observations) action occurrences.

3. Figuring out what the current state could be.

4. Distinguishing between abducible (or explainable) observations and free-will observations and taking them into account in an appropriate manner.

1.2.4 Expressing directives and Goals

The previous three subsection were about the specification and analysis that an agent needs to make to figure out how the world has evolved and may evolve in the future. As we mentioned earlier, we envision our 'autonomous' agent to be ready to take directives from a human. To express such directives, as well as for the agent to express its own goals in a declarative manner we need a goal or directive specification language. Since specifying how the world should evolve means specifying the set of acceptable or desirable trajectories (of the world), a starting point for a goal specification language are temporal logics, such as LTL and CTL*, that are used in program specification. Although these languages have many useful connectives such as $\Diamond$, $\Box$, $\mathcal{U}$, A, and E meaning, *eventually*, *always*, *until*, *for all paths*, and *exist path* respectively, many important directives can not be expressed using these languages. Some of the challenges in this direction are:

1. Although the branching time logic CTL* has connectives A, and E meaning *for all paths*, and *exist path*, we may need multiple version of these connectives to specify paths with respect to various agents in the domain.

2. When actions have non-deterministic effects, it may be hard to find ways that fully guarantee the achievement of a desire, while finding ways that may possibly achieve the desire may be too weak. In such cases, often one is directed to 'try his best'. One of the challenges is to formally specify this notion that will allow us to distinguish efforts that do not try their best from the ones that do.

3. Expressing maintenance goals poses a challenge. While the simple LTL expression $\Box f$ is too strict, the expression $\Box\Diamond f$ does not take into account belligerent agents bent on disrupting the achievement of f, and also does not tell us how soon f will be achieved.

4. In presence of incompleteness, one may need goals such as "at all times the truth of a fluent f must be known". To express such goals one needs to use knowledge modalities together with temporal operators.

1.2.5 Sensing actions and partial observability

Often an agent does not have sensors that can sense all the fluents all the time. In fact certain fluents can not be sensed directly and their values need to be inferred from other sensor values. Certain sensors can only be operated when some specific conditions are satisfied. Finally, even if we can sense all the fluents all the time, that may be too time consuming as well as expensive. Thus often an agent will have incomplete knowledge about the world it is in. This leads to several challenges in representation and reasoning.

1. How do we view sensing activity that can be done only under certain conditions? If we view them as actions, how do we characterize their effect?

2. How do we distinguish the real world and and an agent's knowledge about the world?

3. How do we characterize transitions between an agent's knowledge of the world due to regular and sensing actions?

1.2.6 Control architectures and specification, synthesis and verification of control

Once we analyze an agent's ability (what actions it can do, when and what their impact would be), and are able to give a directive to an agent, the next issue is how the agent executes: what actions it executes and when. This is referred to as the control of an agent and the control program of an agent has several differences with standard computer programs.

Foremost among them is that when a basic action of an agent is performed it may change the truth value of several fluents in the world simultaneously, while similar basic steps in computer programs change the value of a single variable. This is because the variables in a program environment are not so tightly linked[4] that changing one of them simultaneously causes change in many others. On the other hand in many of the worlds an agent resides in the fluents are tightly linked; lifting the cup lifts the water in it; moving from A to B simultaneously makes $at(A)$ false and $at(B)$ true. Some of the other differences are as follows:

- The basic actions of an agent may have non-deterministic effects, while the basic statements of a computer program usually change the program in a deterministic way.

- In most programming environments the program in focus is the only one which can change the values of the variables. In the environment of an agent other agents can change the world. Thus an agent in a dynamic environment can not just assume that the value of a fluent remains unchanged, unless the agent itself changes it.

- Automatic synthesis of agent control is an important issue with autonomous agents, while programs are normally written by people. Because of this, an agent control language depends on the assumptions about the environment, and usually the simplest one that is adequate is picked. For example, in a static domain where our agent is the only one that can make changes, and where the actions are deterministic, the agent control can be simply described by a sequence of actions. But even then automatic synthesis is difficult; its NP-complete even under simplifying assumptions. Thus the agent control language may include features whereby the agent designer may specify some domain knowledge that will help the agent synthesize its control.

[4] ???Not true anymore. Java objects are tightly linked. ??Has there been program correctness research on Java objects.

Based on the above differences we need to look at agent control very differently from standard programming. To start with we need to explore control architectures ranging from reactive architecture consisting of "sense; react" cycles, deliberative architecture consisting of "observe; (re)plan; execute a bit" cycles, and various levels of hybrid architectures that allow reaction for a large number of cases, and judicious deliberation for the other cases. We then need to consider whether plans are simple action sequences, or may have sensing actions, conditionals, and more general features. From the synthesis point of view we need to explore plan synthesis algorithms for various kinds of goals, and various kinds of of plan structures. Finally, we need to consider various ways to specify domain knowledge that can help in plan synthesis. This includes procedural knowledge as in Golog, partial ordering knowledge as in HTNs and temporal knowledge specified in a temporal logic.

1.2.7 The reasoning issues: prediction, planning, explanation, diagnosis, plan verification, and control design

In the previous subsections we discussed several different languages: languages to specify actions and their effects; languages to specify observations; languages to specify directives; and languages to specify control. Let us now give names to specifications in each of these languages and tie them together.

- Domain description D: The specification D consists of specification of actions, including sensing actions and their effects, the executability conditions of the actions, and the relationship (possibly causal) between fluents. The characterization of D is a transition function that shows the transition between states due to actions, and in presence of sensing action and incompleteness, the transition between knowledge states due to regular and sensing actions.

- Evolution description E: The evolution specification consists of specification of triggering conditions, their inhibition conditions, timing of naturally occurring actions, and ordering information between triggers. The characterization of D and E together is a set of possible trajectories.

- Observation specification O_a and O_b: Observation specifications are observations about action occurrences and value of fluents at particular time points, and ordering information between time points. Observations are grouped into two kinds: abducible observations O_a, the observations that can be explained or that are caused; and basic observations O_b, the observations that record happenings out of free will. The characterization of D, E and O_b together is a set of mapped trajectories, where each mapped trajectory consists of a trajectory and a mapping of time points, including the current time point, to states in the trajectory. We will refer to the mapped trajectories of D, E and O_b together as a *model* of (D, E, O_b). The observations in O_a are then used to select those mapped trajectories that explain O_a and filter out the remaining.

- Goal specification G: Goals are high level specification of what one desires in terms of how the world should evolve. We plan to specify it using an appropriate temporal logic. Its characterization will consist of structures made up of states, trajectories, and transition graphs, depending on the particular temporal logic used.

- Control specification C: The control specification is a control program in a particular language. In the simplest case, it will be a sequence of actions, and in more general cases it may include features such as sensing actions, conditional statements, and non-determinism operators together with control knowledge.

- A query Q is of the form G **by_means_of** C **at_timepoint** t, where C is a control specification, G is a goal specification and t is a time point. If C is a simple sequence of actions $a_1, \ldots, a_n$ and G is a fluent f, then the query f **by_means_of** $a_1, \ldots, a_n$ **at_timepoint** t refers to enquiring if f is true in the situation obtained by executing the sequence $a_1, \ldots, a_n$ at the situation corresponding to time point t. On the other hand if G is the temporal goal $\Box f$ then the query $\Box f$ **by_means_of** $a_1, \ldots, a_n$ **at_timepoint** t refers to enquiring if f is true in all situations reached during the execution of the sequence $a_1, \ldots, a_n$ at the situation corresponding to time point t. Thus " **by_means_of** C" may refer to after C is executed or during the execution of C depending on the goal G.
- The entailments $M \models Q_1, \ldots, Q_n$ and $(D, E, O_b) \models Q_1, \ldots, Q_n$: $M \models Q_1, \ldots, Q_n$ is true if $Q_1 \ldots Q_n$ evaluate to true with respect to the model M. The entailment $(D, E, O_b) \models Q_1, \ldots, Q_n$ is true if each of the Q_is evaluate to true with respect to all the models of (D, E, O_b). In the entailments above queries can be replaced by observations.

We now illustrate the various kinds of reasoning by using the models of (D, E, O_b) and the entailment $(D, E, O_b) \models Q_1, \ldots, Q_n$.

- Prediction and plan verification: Given a D, E and O_b, the current time point t_n, the goal G, if we would like to verify if G is indeed obtained by means of a control C then we need to check if
  $(D, E, O_b) \models G$ **by_means_of** C **at_timepoint** t_n
  is true or not.
- Planning: Given a D, E and O_b, the current time point t_n, the goal G, planning for the goal G starting from t_n means finding a control C such that $(D, E, O_b) \models G$ **by_means_of** C **at_timepoint** t_n.
- Explanations and hypothetical explanations: Given (D, E, O_b) and abducible observations O_a, an explanation model of O_a with respect to (D, E, O_b) is a model M of (D, E, O_b) such that $M \models O_a$.

  If no such explanation model exists then we define an hypothetical explanation model of O_a with respect to (D, E, O_b) as a model M of $(D, E, O_b \cup O'_b)$ for some O'_b such that $M \models O_a$.
- Abductive Entailment and hypothetical abductive entailment: Given (D, E, O_b), abducible observations O_a, and queries $Q_1, \ldots, Q_n$:

  The abductive entailment relation $\models_a$ is defined as follows:

  $\langle (D, E, O_b), O_a \rangle \models_a Q_1, \ldots, Q_n$ if O_a has at least one explanation model with respect to (D, E, O_b) and for all explanation models M of O_a with respect to (D, E, O_b), $M \models Q_1, \ldots, Q_n$.

  The hypothetical abductive entailment relation $\models^h_a$ is defined as follows:

  $\langle (D, E, O_b), O_a \rangle \models^h_a Q_1, \ldots, Q_n$ if O_a has at least one hypothetical explanation model with respect to (D, E, O_b) and for all hypothetical explanation models M of O_a with respect to (D, E, O_b), $M \models Q_1, \ldots, Q_n$.
- Diagnosis: Diagnosis involves finding (perhaps a minimal set of) abnormal properties of the world that explain the observations, when the assumption that everything is normal can not explain the abducible observations O_a. I.e., (D, E, O_b) which includes the normality assumption with respect to O_b does not have a model that entail O_a. In that case we need to find a modification of O_b by minimally changing the normality assumption such that the modified triplet (D, E, O'_b) has at least one model that entails O_a.

- Hypothesis generation: In hypothesis generation, one is looking for hitherto unknown aspects of the world (a domain description proposition or an evolution description proposition) that will explain observations O_a that is unexplainable with respect to the currently known (D, E, O_b). If (D, E, O_b) does not have any models that entail O_a, then D', E', O'_b are possible hypothesis if $(D \cup D', E \cup E', O_b \cup O'_b)$ has a model that entails O_a. One may limit and order possible hypothesis in various ways: using predefined sets from which D', E' and O'_b come from, using minimality, based on how many or what percentage of models of $(D \cup D', E \cup E', O_b \cup O'_b)$ entail O_a, etc. Also, different kinds of hypothesis can be generated by restricting one or two of D', E' and O'_b to be an empty set. For example, if D' and E' are required to be empty sets then we only generate hypothetical explanations.

Notes

Free will: `http://en.wikipedia.org/wiki/Free_will`

Autonomy: `http://en.wikipedia.org/wiki/Autonomy`

Chapter 2

Representing Actions and the Environment

If one were to start the design of an autonomous agent from scratch then a first step in the systematic design of an autonomous agent would be to analyze the kind of directives that will be given to the agent, the kind of world the agent would be in and from that design the actuators and sensors for the agent. However, often designers are given an unprogrammed artifact with built-in abilities with respect to acting and sensing and are required to develop the software that would make the artifact an autonomous agent with respect to certain kinds of directives and certain types of worlds. This means the designers would have to develop software that can synthesize controls for given directives or at the minimum can verify a given control and adapt it to new directives. Thus our first research issue becomes the representation of the world and the actions that the agent can do and the impact of these actions on the world.

Representing actions and their effects on the world has a long history in AI. The realization that the number of states of the world is often exponential with respect to the number of fluents – properties of the world, immediately ruled out a representation that stores the explicit mapping from states and actions to states. Some of the early historical landmarks on action representation include the use of STRIPS operators for planning [FN71], the invention of situation calculus [MH69], the discovery of the frame problem, the early attempt to solve the frame problem using non-monotonic logics [MH69], use of formalization of the frame problem as a benchmark for non-monotonic logics and the Yale shooting misunderstanding [**?**]. Some of the lessons from these were as follows:

1. The original STRIPS syntax is very restrictive. With slight generalization (say actions with non-deterministic effects) the semantics is no longer straightforward.

2. Representing what changes in the world due to an action is comparatively easy and succinct in a logical language. However representing what does not change, is either hard or verbose. Finding a succinct representation of what does not change is often referred to as the frame problem.

3. While non-monotonic logics can be used to for a succinct solution of the frame problem, the semantic nuances of non-monotonic logics are even difficult for specialists and need special care.

4. Representing English statements in non-monotonic logic leads to misunderstandings as the semantics of English is not precise.

Based on the above lessons the early nineties saw a large body of research on high level action description languages whose syntax were English-like and whose semantics were defined using set theory and simple mathematical notions rather than mathematical logics. The reasoning with respect to these languages could

then be done through a translation to a logical language and one now had a way to formally prove that the logical encodings were correct. The use of set theory and simple mathematical notions in defining the semantics of these languages freed the reader from knowing or learning a logical language to understand the semantics. In the rest of this section we describe several of these high level languages. Our presentation will track the evolution of these languages and will focus on the new aspects of each succeeding language. This will make the nuances of each aspect clearer as opposed to if we were to present a single language with all the features. We start with the simplest language called $\mathcal{A}$.

2.1 The Language $\mathcal{A}$: Motivation

The language $\mathcal{A}$ was proposed in [GL93] as a simple action description language with an English-like syntax and a semantics defined using simple set theory. Later in [Kar93] it was used to show the correctness of various logical solution of the frame problem. To illustrate the English-like syntax of $\mathcal{A}$ let us consider the specification of the actions in one version of the Yale shooting problem. The specification of the actions and their effects consists of the following sentences:

> *Loading the gun* causes the gun to be *loaded. Shooting* causes the turkey *to die*. *Shooting* can be executed only if the gun is *loaded*.

Using the syntax of $\mathcal{A}$ the above information is expressed as follows:

load	**causes**	*loaded*
shoot	**causes**	$\neg$*alive*
executable	*shoot*	**if** *loaded*

The general syntax of the action specification part of $\mathcal{A}$ is given through two kinds of constructs *fluent effect axioms* (also known as *dynamic laws*) and *executability axioms*. They are of the following form:

- fluent effect axiom:

$$a \textbf{ causes } l \textbf{ if } \varphi \tag{2.1}$$

- executability axiom:

$$\textbf{executable } a \textbf{ if } \varphi \tag{2.2}$$

where a is an action, l is a fluent literal, and φ is a set of fluent literals or a special symbol $\top$, which is not a fluent and represents *true*. Intuitively, (2.1) says that if a is executed in a situation, in which φ holds, then l will be true in the resulting situation. (2.2) specifies the condition under which a can be executed. When $\varphi = \top$, we will drop the **if**-part in (2.1) and simply write a **causes** l.

A domain specification D in $\mathcal{A}$ consists of a set of fluent effect axioms and executability axioms. We will assume that for each action a, the executability axiom **executable** a **if** $\top$ belongs to D if the spefication of D does not contain any executability axiom whose action is a.

Recall that an autonomous agent would often take a domain specification, observations and a given goal to either validate a given control or synthesize the control. In $\mathcal{A}$ the observation language can be used to

express observations of the kind: "*The turkey is initially alive.*". Using the syntax of $\mathcal{A}$ this observation is expressed as follows:

initially *alive*.

In general, observations (O) in $\mathcal{A}$ are a collection of value propositions of the form:

$$\textbf{initially } l \tag{2.3}$$

where l is a fluent literal.

We often refer to the pair (D, O) where D is a domain specification and O is a set of observations as an *action theory*.

Queries in $\mathcal{A}$ are used to express questions such as: "Would the turkey be alive after a shoot action was attempted in the initial situation?" This is expressed in $\mathcal{A}$ as follows:

alive **after** [*shoot*]

In general a query Q in $\mathcal{A}$ is of the form:

$$l \textbf{ after } [a_1 \circ \ldots \circ a_n] \tag{2.4}$$

where l is a fluent literal and a_i's are actions (the $\circ$ operator stands for the concatenation operator). Intuitively the above query means: *Would l be true if the sequence of actions $[a_1 \circ \ldots \circ a_n]$ were attempted in the initial situation?*

For convenience, we introduce the following shorthands:

- For a sequence of literals $l_1, \ldots, l_n$,

  $$\textbf{initially } l_1, \ldots, l_n$$

  denotes the collection of observations $\{\textbf{initially } l_i \mid 1 \leq i \leq n\}$.

- For a sequence of sets of literals $\varphi_1, \ldots, \varphi_n$,

  $$a \textbf{ causes } l \textbf{ if } \varphi_1 \vee \ldots \vee \varphi_n$$

  denotes the collection of fluent effect axioms $\{a \textbf{ causes } l \textbf{ if } \varphi_i \mid 1 \leq i \leq n\}$.

2.2 Reasoning With Respect to a Theory in $\mathcal{A}$: Semantics of $\mathcal{A}$

The semantics of $\mathcal{A}$ defines the entailment relation $(D, O) \models Q$. In other words, given a domain specification D and observation O, can we conclude the query Q from D and O. Even with the simple syntax of queries, the entailment $(D, O) \models Q$ can be used for various kinds of reasoning that an autonomous agent may do. The following example illustrates this.

Example 1 (Formulating Plan Verification) Consider the domain specification[1]

$$D_1 = \left\{ \begin{array}{l} \mathit{load} \textbf{ causes } \mathit{loaded} \\ \mathit{shoot} \textbf{ causes } \neg \mathit{alive} \textbf{ if } \mathit{loaded} \end{array} \right\}$$

and the observation

$$O_1 = \{\textbf{initially } \neg \mathit{loaded}, \mathit{alive}\}.$$

If the agent's goal is to make $\neg \mathit{alive}$ true and it is given a possible plan of $[\mathit{load}, \mathit{shoot}]$ for achieving this goal, then the agent can verify the plan by checking whether or not

$$(D_1, O_1) \models \neg \mathit{alive} \textbf{ after } [\mathit{load}, \mathit{shoot}]$$

holds. In this case, intuitively the above entailment does hold. □

Example 2 (Formulating Simple Planning) Consider D_1 and O_1 from the previous example. Suppose the agent's goal is still to make $\neg \mathit{alive}$ true. However, instead of verifying a given plan, it has to find a plan for achieving his goal. In other words, the agent needs to find a sequence of actions α such that

$$(D_1, O_1) \models \neg \mathit{alive} \textbf{ after } \alpha$$

holds. In this case, intuitively the above entailment holds when $\alpha = [\mathit{load}, \mathit{shoot}]$, i.e., α is *load* followed by *shoot*. □

Example 3 (Formulating Conformant Planning) Consider the domain specification D_1 from Example 1 and let $O_2 = \{\textbf{initially } \mathit{alive}\}$. Suppose the agent's goal is still to make $\neg \mathit{alive}$ true. As in Example 2, the agent needs to find a sequence of actions α such that

$$(D_1, O_2) \models \neg \mathit{alive} \textbf{ after } \alpha$$

holds. Since O_2 does not completely specify the initial situations, the plan (or α) must be able to achieve the goal no matter which of the possible initial situations is the initial situation. Planning in this situation is referred to as conformant planning. □

Example 4 (Formulating Abduction) Consider D_1 and O_2 from Example 1 and 3 respecitively; and suppose the agent wants to find out under what additional conditions (about the initial situation) he can guarantee that $\neg \mathit{alive}$ **after** $[\mathit{load}, \mathit{shoot}]$. This can be posed as finding a set of observations O' such that

$$(D_1, O_2 \cup O') \models \neg \mathit{alive} \textbf{ after } \alpha$$

holds. □

We now give the semantics of the language $\mathcal{A}$.

[1] We omit **executable** *load* **if** $\top$ and **executable** *shoot* **if** $\top$ from D_1.

2.3 Semantics of $\mathcal{A}$

To characterize the semantics of $\mathcal{A}$ let us ponder about the information contained in D and O: O contains information about the initial state and D tells us what will be true (or false) in a state after an action is executed. In other words, D tells us about the transition between states due to actions. Moreover, to answer queries of the form

$$f \textbf{ after } [a_1 \circ \ldots \circ a_n]$$

it is sufficient to know what the initial state is and the state transition due to actions. Thus the semantics of $\mathcal{A}$ involves finding the initial state σ_0 and the transition function Φ (between states due to actions) with respect to a given D and O, and using them to define the entailment between (D, O) and queries.

For a particular domain specification D, we have an associated signature consisting of a set of actions $\mathbf{A}$ and a set of fluents $\mathbf{F}$. *Fluent literals* are fluents or their negations. A *state* is a subset of $\mathbf{F}$. A fluent f holds in a state σ if $f \in \sigma$ and a fluent literal $\neg f$ holds in a state σ if $f \notin \sigma$. Given a fluent literal l, we use $\bar{l}$ to denote its negation. In other words, if $l = f$ for some fluent f then $\bar{l} = \neg f$ and if $l = \neg f$ for some fluent f then $\bar{l} = f$. An action a is *executable* in a state σ if D contains an executable condition **executable** a **if** φ and φ holds in σ.

Definition 1 A set of fluents σ is an *initial state* corresponding to O, if for all observations of the form **initially** l (where l is a fluent literal), l holds in σ.

The transition function of a domain specification is defined next.

Definition 2 Given a domain specification D in $\mathcal{A}$ its *transition function* is any function Φ from states and actions to states that satisfies the following conditions:

For all actions a, fluents f, and states σ:

- if a is executable in σ then $\Phi(a, \sigma)$ is a state satisfying the following properties:
  - If a **causes** f **if** φ belongs to D and φ holds in σ then f must be in $\Phi(a, \sigma)$.
  - If a **causes** $\neg f$ **if** φ belongs to D and φ holds in σ then f must not be in $\Phi(a, \sigma)$.
  - If D does not include such fluent effect axiom (about a and f) then $f \in \Phi(a, \sigma)$ iff $f \in \sigma$.
- if a is executable in σ then $\Phi(a, \sigma)$ is undefined. □

Theorem 1 For any domain specification D, there is at most one transition function.

Proof. Assume that D has two distinct transition function Φ and Ψ. This means that there exists some state s and action a such that $\Phi(a, \sigma) \neq \Psi(a, \sigma)$. Clearly, neither $\Phi(a, \sigma)$ nor $\Psi(a, \sigma)$ can be undefined. This means that there exists some $f \in \mathbf{F}$ such that $f \in \Phi(a, \sigma)$ and $f \notin \Psi(a, \sigma)$. $f \in \Phi(a, \sigma)$ implies that (i) there exists some a **causes** f **if** φ in D such that φ holds in σ or (ii) $f \in \sigma$. (i) would indicate that $f \in \Psi(a, \sigma)$ and hence cannot happen. (ii), together with $f \notin \Psi(a, \sigma)$, implies that there exists some a **causes** $\neg f$ **if** ψ in D such that ψ holds in σ. But this means that f must not be in $\Phi(a, \sigma)$. A contradiction with $f \in \Phi(a, \sigma)$, i.e., a contradiction with our assumption that D has two distinct transition functions. □

Definition 3 A domain specification D is said to be *consistent* if it has a transition function.

Remark 1 *Following Theorem 1, a consistent domain specification has a unique transition function. For convenience, the transition function of a consistent domain specification D will be denoted by* Φ_D.

The next example shows that not every domain specification is consistent.

Example 5 (Inconsistent Domain Specification) it is easy to see that the domain specification with two fluent effect axioms

$$a \textbf{ causes } f \textbf{ if } h$$
$$a \textbf{ causes } \neg f \textbf{ if } g$$

is inconsistent since it does not have any transition function Φ. In particular, for the state $\sigma = \{h, g\}$ and action a, there is no state satisfying Definition 2. Thus, $\Phi(a, \sigma)$ is not defined. □

Remark 2 *Inconsistent domain specifications indicate that there is something wrong and/or something missing in the modeling process. (Need more elaboration ???)*

Theorem 2 For any consistent domain specification D, its transition function Φ_D satisfies the condition:

- if a is executable in σ then
$$\Phi_D(a, \sigma) = \sigma \cup E^+_D(a, \sigma) \setminus E^-_D(a, \sigma)$$
where
$$E^+_D(a, \sigma) = \{f \mid a \textbf{ causes } f \textbf{ if } \varphi \in D \text{ and } \varphi \text{ holds in } \sigma\}$$
and
$$E^-_D(a, \sigma) = \{f \mid a \textbf{ causes } \neg f \textbf{ if } \varphi \in D \text{ and } \varphi \text{ holds in } \sigma\}.$$
- if a is not executable in σ then $\Phi_D(a, \sigma)$ is undefined.

Proof. Follows directly from Definition 2. □

Example 6 Consider the action theory (D_1, O_1) from Example 1. Let

$$\sigma_0 = \{alive\}.$$

We have that

$$\Phi_{D_1}(shoot, \sigma_0) = \{alive\}$$

because $E^+_{D_1}(shoot, \sigma_0) = E^-_{D_1}(shoot, \sigma_0) = \emptyset$ and

$$\Phi_{D_1}(load, \sigma_0) = \{alive, loaded\}$$

since $E^+_{D_1}(load, \sigma_0) = \{loaded\}$ and $E^-_{D_1}(load, \sigma_0) = \emptyset$ □

It is easy to see that if for every fluent f, $\{\textbf{initially } f, \textbf{initially } \neg f\} \not\subseteq O$, then there exists at least one initial state corresponding to O. We will say that O is *consistent* if there exists at least one initial state corresponding to O.

Definition 4 For a transition function Φ, a state σ, and a sequence of actions $[a_1 \circ \ldots \circ a_n]$, we define the extended transition function of Φ, denoted by $\widehat{\Phi}$, as follows.

$$\widehat{\Phi}([a_1 \circ \ldots \circ a_n], \sigma) = \begin{cases} \sigma & \text{if } n = 0 \\ \Phi(a_n, \widehat{\Phi}([a_1 \circ \ldots \circ a_{n-1}], \sigma) & \text{otherwise} \end{cases}$$

Definition 5 (σ_0, Φ) is a model of (D, O) iff Φ is the transition function of D and σ_0 is an initial state corresponding to O.

Definition 6 Let (D, O) be an action theory.

- (D, O) is said to be *consistent* if it has a model.
- (D, O) is said to be *complete* if it has a unique model.
- For an action sequence α, $(D, O) \models f$ **after** α if for all models (σ_0, Φ) of (D, O), f holds in $\widehat{\Phi}(\alpha, \sigma_0)$.
- An action sequence α is a *plan* for a fluent formula φ if $(D, O) \models \varphi$ **after** α.

Observe that consistency of an action theory requires the consistency of its domain specification and observations. Because of the uniqueness of the transition function, completeness is equivalent to the uniqueness of the initial state. This means that for each $f \in \mathbf{F}$, $\{\textbf{initially } f, \textbf{initially } \neg f\} \cap O \neq \emptyset$.

Example 7 Continuing with Example 6, we have that the theory (D_1, O_1) is complete and has a unique initial state $\sigma_0 = \{alive\}$. Furthermore, $\Phi_{D_1}(load, \sigma_0) = \{loaded, alive\}$.
Let $\sigma_1 = \{loaded \circ alive\}$. We have that

$$E^+_{D_1}(shoot, \sigma_1) = \emptyset \quad \text{and} \quad E^-_{D_1}(shoot, \sigma_1) = \{alive\}.$$

Hence, $\widehat{\Phi}([load \circ shoot], \sigma_0) = \Phi(shoot, \sigma_1) = \{loaded\}$. We have that $\neg alive$ holds in $\{loaded\}$, i.e.,

$$(D_1, O_1) \models \neg alive \textbf{ after } [load \circ shoot].$$

□

Exercise 1 *Let* $D'_1 = D_1 \cup \{shoot \textbf{ causes } \neg loaded\}$. *Prove that*

$$(D'_1, O_1) \models \neg alive \textbf{ after } [load, shoot].$$

2.4 Complexity of Reasoning and Planning in $\mathcal{A}$

$\mathcal{A}$ is among the simplest languages for the specification of actions and their effects. In this section we analyze the complexity of reasoning and planning in this simple langauge. To this end, we say that the size of a domain specification D in the language $\mathcal{A}$ is the sum of the number of actions, the number of fluents, and the number of laws of the form (2.1) and denote it by $|D|$. We consider the following decision problems:

- $\text{CS}^{\mathcal{A}}$ (*Consistent domain*) Given a domain specification D, determine whether D is consistent or not.
- $\text{CI}^{\mathcal{A}}$ (*Consistent initial state*) Given a consistent domain specification D, a set of observations O, and a set of fluents σ_0, determine whether σ_0 is an initial state with respect to (D, O).
- $\text{PC}^{\mathcal{A}}$ (*Planning with complete initial state*) Given a consistent domain specification D, a complete set of observations about the initial state O, and a fluent literal l, a polynomial function $f : N \to N$, determining an action sequence $[a_1 \circ \ldots \circ a_m]$, where m is bounded by $f(|D|)$, such that $(D, O) \models l$ **after** $[a_1 \circ \ldots \circ a_m]$.

Theorem 3 **1.** $\text{CS}^{\mathcal{A}}$ *is in P.*

2. $\text{CI}^{\mathcal{A}}$ *is in P.*

3. $\text{PC}^{\mathcal{A}}$ *is NP-complete.*

Proof.

1. Φ_D exists iff for every pair of fluent effect axioms of the form

$$a \textbf{ causes } f \textbf{ if } \varphi \quad \text{and} \quad a \textbf{ causes } \neg f \textbf{ if } \psi,$$

there exists no state σ such that both φ and ψ hold in σ. Due to the consistency of a state, this condition is satisfied iff there exists a fluent literal l such that $l \in \varphi$ and $\bar{l} \in \psi$, i.e., $\varphi \cap \bar{\psi} \neq \emptyset$, where $\bar{\delta} = \{\bar{l} \mid l \in \delta\}$. Clearly, this can be checked in polynomial time (in the size of the number of fluents). The conclusion follows from the fact that there are less than $|D|^2$ pairs of fluent effect axioms.

2. Since O consists of observations of the form **initially** l, it has at most $2 * |\mathbf{F}|$ value propositions where $|\mathbf{F}|$ is the number of fluents in $\mathbf{F}$. Deciding whether or not σ_0 satisfies **initially** l is equivalent to checking whether $l \in \sigma_0$ (if $l \in \mathbf{F}$) or $l \notin \sigma_0$ (if $l = \neg f$ for some $f \in \mathbf{F}$). This can be done in at most $|\mathbf{F}|$ operations. Thus, the problem is polynomial in the size of the number of fluents of D.

3. Given (D, O), l, and f, let $\sigma_0 = \{f \mid f$ is a fluent and **initially** f belongs to $O\}$. Since O is complete, σ_0 is unique. Furthermore, it is easy to see that given a state σ and an action a, computing $\Phi_D(a, \sigma)$ is polynomial. As such, computing $\widehat{\Phi}_D([a_1 \circ \ldots \circ a_m], \sigma_0)$ is also polynomial if m is bounded by $f(|D|)$.

(i) **Membership:** It is easy to see that the following guess-and-check NP-algorithm can be used to solve the $\text{PC}^{\mathcal{A}}$ problem.

- *Guess*: a sequence of actions $[a_1 \circ \ldots \circ a_m]$ where m is bounded by $f(|D|)$.
- *Check*: whether l holds in $\widehat{\Phi}_D([a_1 \circ \ldots \circ a_m], \sigma_0)$.

This indicates that $\text{PC}^{\mathcal{A}}$ belongs to the class of NP problems.

(ii) **Hardness:** We now reduce 3SAT to this problem. Let F be the formula

$$(f_1^1 \vee f_1^2 \vee f_1^3) \wedge \ldots \wedge (f_n^1 \vee f_n^2 \vee f_n^3).$$

Let $g_1, \ldots, g_n, g$ be new atoms not in the language of F. We construct (D, O) as follows.

- The language of D consists of actions a, a', and a_p for each atom p in the language of F. D consists of the following fluent effect axioms:
  - For each i,

$$\begin{array}{l} a \textbf{ causes } g_i \textbf{ if } f_i^1 \\ a \textbf{ causes } g_i \textbf{ if } f_i^2 \\ a \textbf{ causes } g_i \textbf{ if } f_i^3 \end{array}$$

belong to D.

– D contains the fluent effect axiom

$$a' \textbf{ causes } g \textbf{ if } g_1, \ldots, g_n$$

– For each atom p in the language of F, D contains the fluent effect axioms

$$a_p \textbf{ causes } p \textbf{ if } \neg p$$

and

$$a_p \textbf{ causes } \neg p \textbf{ if } p.$$

- O consists of the following observations:
  - **initially** $\neg p$ for every atom p in the language of F, and
  - **initially** $\neg g_1, \ldots, \neg g_n, \neg g$.
- The goal of the planning problem is g.

Let α be an action sequence such that $(D, O) \models g$ **after** α. We can observe the following:

- α must contain the actions a and a' and a must occur before a'. (Because a' is the only action for achieving g. Furthermore, for g to be true after the execution of a', $g_1, \ldots, g_n$ need to be true and this can only be achieved by a.)
- $(D, O) \models g$ **after** β where β is obtained from α by (i) removing all the action occurrences after the occurrence of a' and (ii) removing all the action occurrences between a and a'.

The above observations, together with the fact that a_p flips the truth value of p, allow us to conclude that there exists a sequence of actions $[a_1 \circ \ldots \circ a_m]$ such that

$$(D, O) \models g \textbf{ after } [a_1 \circ \ldots \circ a_m]$$

iff there exists a sequence of actions $[b_1 \circ \ldots \circ b_k \circ a \circ a']$ where $k \leq n_p$ (n_p is the number of atoms in the language of F) and $b_i \neq b_j$ for every pair of i and j, $1 \leq i \neq j \leq n$ such that

$$(D, O) \models g \textbf{ after } [b_1 \circ \ldots \circ b_k \circ a \circ a'].$$

Furthermore, let I be a truth value assignment for atoms in the language of F. Assume that $p_1, \ldots, p_k$ are true and other atoms are false w.r.t. I. It is easy to check that $(D, O) \models g$ **after** $[a_{p_1} \circ \ldots \circ a_{p_k} \circ a \circ a']$ iff I is a model of F. This one-to-one correspondence between interpretations of F and plans of length less than or equal to $n_p + 2$ for g shows that F is satisfiable iff there exists a plan for g with respect to (D, O). □

2.5 Implementing Reasoning and Planning in $\mathcal{A}$ By Translating to SAT

In this section we show how we can encode domain specifications of $\mathcal{A}$ in SAT and use model finding and reasoning in SAT to reason and plan with respect to $\mathcal{A}$. We will address the following problems:

- **Hypothetical reasoning**: Given a domain specification D, a complete set of observations O, a fluent literal l, and an action sequence $[a_1 \circ \ldots \circ a_t]$, checking whether or not $(D, O) \models l$ **after** $[a_1 \circ \ldots \circ a_t]$ holds.
- **Planning with complete information**: Given a domain specification D, a complete set of observations O, a fluent literal l, an interger t, checking whether or not there exists an action sequence $[a_1 \circ \ldots \circ a_t]$ such that $(D, O) \models l$ **after** $[a_1 \circ \ldots \circ a_t]$.

2.5.1 Encoding for Hypothetical Reasoning

Given an integer n, we construct a theory $tr\_do(D, O, n)$ for reasoning about effects of action sequences which contains at most n actions. The alphabet of $tr\_do(D, O, n)$ consists of propostion of the form $f[i]$ and $a[i]$ where f is a fluent, a is an action, and i is an integer between 1 and $n + 1$. Intuitively, $tr\_do(D, O, n)$ encodes the evolution of the world through the execution of the action sequence. Intuitively, $f[i]$ (resp. $\neg f[i]$) encodes that f is true (resp. false) in the state after the execution of the first $i - 1$ actions of the sequence. Similarly, $a[i]$ (resp. $\neg a[i]$) states that the action a is executed after the execution of the first $i - 1$ actions of the sequence. We introduce the following notations.

- For each fluent f, let
  - $\mathbf{A}_f^+$ be the collection of fluent effect axioms of the form a **causes** f **if** φ in D; and
  - $\mathbf{A}_f^-$ be the collection of fluent effect axioms of the form a **causes** $\neg f$ **if** φ in D.
- For a set of fluent literals φ and an integer i,
  - $\varphi[i]$ denotes the formula $\bigwedge_{l \in \varphi} l[i]$; and
  - $\neg\varphi[i]$ denotes the formula $\bigvee_{l \in \varphi} \bar{l}[i]$. (Recall that $\bar{l}$ denotes the negation of l.)

The theory $tr\_do(D, O, n)$ is defined as follows.

- For each fluent f and an integer i, $1 \leq i \leq n$, the formula

$$f[i+1] \Leftrightarrow \left(\bigwedge_{a \textbf{ causes } f \textbf{ if } \varphi \in \mathbf{A}_f^+} (a[i] \wedge \varphi[i]) \right) \vee \left(f[i] \wedge \bigwedge_{a \textbf{ causes } \neg f \textbf{ if } \varphi \in \mathbf{A}_f^-} (\neg a[i] \vee \neg\varphi[i]) \right) \tag{2.5}$$

  belongs to $tr\_do(D, O, n)$.
- For each **initially** f in O, the atom $f[1]$ is in $tr\_do(D, O, n)$.
- If $a_1, \ldots, a_r$ are all the actions in the domain, then
  - for all $1 \leq i \leq n$, if **executable** a **if** φ belongs to D and $\varphi \neq \top$, $a[i] \Rightarrow \varphi[i]$ belongs to $tr\_do(D, O, n)$.
  - for all $1 \leq i \leq n$, the formula $a_1[i] \vee \ldots \vee a_r[i]$ is in $tr\_do(D, O, n)$.
  - for all $1 \leq i \leq n$ and $1 \leq j, k \leq r$ where $j \neq k$, the formula $\neg(a_j[i] \wedge a_k[i])$ is in $tr\_do(D, O, n)$.
- A query Q given by l **after** $[a_1 \circ \ldots \circ a_t]$ is translated to the formula

$$a_1[1] \wedge \ldots \wedge a_t[t]$$

  which we will refer to as $tr\_q(Q)$.

The following proposition is about the correctness of the above translation.

Theorem 4 Given a consistent domain specification D, a complete set of initial state observations O and a query Q of the form l **after** $[a_1 \circ \ldots \circ a_t]$, $(D, O) \models Q$ iff $tr\_do(D, O, t) \cup tr\_q(Q) \models l[t+1]$.

Proof. Let M be a model of $tr\_do(D, O, t)$ and $s_i(M) = \{f \mid f \in \mathbf{F},\ f[i] \text{ is true in } M\}$. The encoding of $tr\_do(D, O, t)$ ensures that for each integer i, $1 \leq i \leq t$, there exists a unique action a such that $a[i]$ is true in M and if $a[i]$ is true in M then a is executable in $s_i(M)$. We first prove that $s_1(M) = \sigma_0$ where σ_0 is the initial state of (D, O). Since O is complete, for every fluent $f \in \mathbf{F}$, we have that

- $f \in \sigma_0$ iff **initially** f belongs to O iff $f[1]$ belongs to $tr\_do(D, O, t)$ iff $f \in s_1(M)$.
- $f \notin \sigma_0$ iff **initially** $\neg f$ belongs to O iff $\neg f[1]$ belongs to $tr\_do(D, O, t)$ iff $f \notin s_1(M)$.

The above two items conclude that $s_1(M) = \sigma_0$.

We will now prove that $s_{i+1}(M) = \Phi(a_i, s_i(M))$ for $1 \leq i \leq t+1$ by induction over i where $a_i[i]$ is true in M.

- **Base**: The construction of $tr\_do(D, O, t)$ implies that there exists some action a_1 such that $a_1[1]$ is true in M and a_1 is executable in $s_1(M)$. Thus $\Phi(a_1, s_1(M))$ is defined. Let f be a fluent. We have that $f \in \Phi(a_1, s_1(M))$ if one of the following two conditions is satisfied.

  (i) $f \in E^+_D(a_1, s_1(M))$. This implies that there exists a fluent effect axiom a_1 **causes** f **if** φ in $\mathbf{A}^+_f$ such that φ is true in $s_1(M)$. By Equation 2.5, we have that $f[2]$ is true in M, i.e., $f \in s_2(M)$.

  (ii) $f \in s_1(M)$ and $f \notin E^-_D(a_1, s_1(M))$. This implies that $f[1]$ is true in M and for every fluent effect axiom b **causes** $\neg f$ **if** φ in $\mathbf{A}^-_f$, either $b \neq a_1$, and hence, $\neg b[1]$ is true in M or $b = a_1$, and hence, φ does not hold in $s_1(M)$, i.e., $\neg\varphi[1]$ is true in M. Therefore, $f[1] \wedge \bigwedge_a$ **causes** $\neg f$ **if** $\bigvee_{\varphi \in \mathbf{A}^-_f}(\neg a[1] \vee \neg\varphi[1])$ is true in M, and hence, $f[2]$ is true in M, i.e., $f \in s_2(M)$.

- **Step**: The proof for the inductive step is similar to the base case and is omitted here for brevity.

The above property allows us to show that if M is a model of $tr\_do(D, O, t) \cup tr\_q(Q)$ then $s_{t+1}(M) = \widehat{\Phi}([a_1 \circ \ldots \circ a_t], \sigma_0)$, which proves the 'if'-direction of the proposition.

To prove the 'only if'-direction of the proposition, let M be an interpretation defined by (i) $f[1]$ is true in M iff $f \in \sigma_0$; (ii) for each fluent f and integer $2 \leq i \leq t+1$, $f[i]$ is true in M iff $f \in \widehat{\Phi}([a_1 \circ \ldots \circ a_{i-1}], \sigma_0)$; and (ii) for each action a, $a[i]$ is true in M iff $a = a_i$. It is easy to see that M is indeed a model of $tr\_do(D, O, t) \cup tr\_q(Q)$ and $s_{t+1}(M) = \widehat{\Phi}([a_1 \circ \ldots \circ a_t], \sigma_0)$, which concludes the 'only if'-direction of the proposition. □

2.5.2 Planning in Complete Action Theories

The following proposition follows immediately from Theorem 4.

Theorem 5 Let D be a consistent domain specification, O be a complete set of observations about the initial state, and l be a fluent literal. A sequence of actions $[a_1 \circ \ldots \circ a_t]$ is a plan for l iff there is a model of $tr\_do(D, O, t) \cup l[t+1]$ in which $\{a_1[1], \ldots, a_t[t]\}$ are true. □

The above suggests that to find a plan, if we know that a plan of length t exists then we should give a propositional solver the propositional theory $tr\_do_2(D, O, t) \cup g[t+1]$ and ask it to find a model. Each model (if exists) will encode a plan.

If we do not know the plan length but have an idea of an upper bound, then we can have a dummy action which does not affect the world or we can replace $l[t+1]$ by $l[1] \vee \ldots \vee l[t+1]$.

Example 8 Let us consider the example (D_1, O_1). $tr\_do(D_1, O_1, n)$ contains the following formulas:

- $\neg loaded[1] \wedge alive[1]$ (translating of O_1).
- $loaded[i+1] \Leftrightarrow (load[i] \vee loaded[i])$ where $1 \leq i \leq n+1$ (Equation 2.5, for fluent *loaded*).
- $alive[i+1] \Leftrightarrow (alive[i] \wedge (\neg shoot[i] \vee \neg loaded[i]))$ where $1 \leq i \leq n+1$ (Equation 2.5, for fluent *alive*).
- $load[i] \vee shoot[i]$ and $\neg(load[i] \vee shoot[i])$ where $1 \leq i \leq n$ (translating of actions).

To reason if $(D_1, O_1) \models alive$ **after** *load*, *shoot*, we translate the query $Q = \neg alive$ **after** *load*, *shoot* into the formula $load[1] \wedge shoot[2]$, i.e., $tr\_q(Q) = load[1] \wedge shoot[2]$.

Now we can answer $(D_1, O_1) \models Q$ by checking whether or not $tr\_do(D_1, O_1, 2) \cup tr\_q(Q) \models \neg alive[3]$ holds. □

2.6 Representing the Relationship Between the Fluents in the Environment

So far in this section we mostly discussed the representation of actions and their effects on the environment. In this we did not explicitly consider the relationship between various fluents of the world. Any connection between them were addressed by making sure that the actions affecting one also affect the other in an appropriate way. For example, suppose there are two fluents *dead* and *alive* meaning the turkey is dead and meaning the turkey is alive respectively. Now if we were to use the language $\mathcal{A}$, for any effect axiom of an action that specifies the impact on one of the two fluents there would be a corresponding effect axiom about the other fluent. For example, whenever we say something about *alive* we must say the opposite about *dead*. Hence to match *shoot* **causes** $\neg$*alive* **if** *loaded* we will need to add *shoot* **causes** *dead* **if** *loaded*.

From the point of view of representing the effect of an action, often an action has some direct effects and because many fluents are linked with each other, it has some indirect effects. To express all the effects of an action directly one would then need to compute the indirect effects. But computation or reasoning is supposed to come later; after the representation. Thus a superior alternative is to express direct effects of actions as well as the relationship between the fluents and let the semantics or the reasoning mechanism compute the indirect effects. With respect to the *alive* and *dead* example, this approach will lead to adding the knowledge 'dead *iff* ¬ alive'. But that is beyond the syntax of $\mathcal{A}$. So we consider the language $\mathcal{AR}$ which allows specification of relationship between fluents.

2.6.1 The Language $\mathcal{AR}$

The language $\mathcal{AR}$ was first introduced in [KL94] and was called $\mathcal{AR}_0$. It was later improved and extended to deal with nonpropositional fluents in [GKL97]. The language also considers *non-inertial fluents* and indeterminate fluent effect axioms (or indeterminate effect propositions). The following discussion is based on the materials in [GKL97] where we consider only propositional and inertial fluents. We also consider only deterministic effect propositions.

Syntax: The syntax of $\mathcal{AR}$ is a superset of $\mathcal{A}$. In addition to the propositions allowed by $\mathcal{A}$, $\mathcal{AR}$ allows classical domain constraints of the form:

$$\textbf{always} \ \phi \tag{2.6}$$

where ϕ is a propositional formula. Intuitively, this means that in every state of the world, the formula ϕ is true.

As an example, the formula

$$\textbf{always}\ \ dead \Leftrightarrow \neg alive$$

states that *dead* and $\neg$*alive* are equivalent. This implies that there exists no state of the world containing $\{dead, alive\}$, in which both *dead* and *alive* are true and the formula $dead \Leftrightarrow \neg alive$ is false. Example 7 shows that $(D_1, O_1) \models dead$ **after** $[load, shoot]$. As such, given the domain specification $D_2 = D_1 \cup \{$ **always** $dead \Leftrightarrow \neg alive\}$, we would like to conclude that $(D_2, O_1) \models dead$ **after** $[load, shoot]$.

Semantics: Similar to $\mathcal{A}$, the semantics of $\mathcal{AR}$ is based on defining an initial state and a transition function between states where each state is a set of fluents satisfying *all* domain constraints. In the case of $\mathcal{A}$ the transition from one state to another due to an action was straightforward: the positive effects of the action were made true in the new state and the negative effects were made false in the new state and all other fluent values were left untouched. In case of $\mathcal{AR}$, we have direct effects that can be computed using E_D^+ and E_D^-, as well as indirect effects. For example, in the action theory (D_2, O_1), the action *shoot*, if executed after the action *load*, has $\neg$*alive* as its direct effect and *dead* as its indirect effect. Taking into account the indirect effects, that are caused by the classical domain constraints, poses a challenge.

This challenge is addressed by borrowing ideas from the belief update literature. The current state can be thought of as the initial belief and the effect of an action can be thought of as an update to the initial belief. Thus the resulting state should be a state that satisfies the effect of the actions as dictated by the effect propositions as well as the domain constraints and is minimally different from the initial state.

To define the closeness between the states we use the following well known ordering based on set difference:

$$\sigma_1 \leq_\sigma \sigma_2 \ \ \text{iff} \ \ (\sigma_1 \setminus \sigma) \cup (\sigma \setminus \sigma_1) \subseteq (\sigma_2 \setminus \sigma) \cup (\sigma \setminus \sigma_2).$$

As usually, we write $\sigma_1 <_\sigma \sigma_2$ to denote that $\sigma_1 \leq_\sigma \sigma_2$ and $\sigma_2 \not\leq_\sigma \sigma_1$. For a set of states S and a state σ, $\texttt{min}_{<_\sigma}(S) = \{\sigma' \mid \sigma' \in S$, there exists no $\sigma'' \in S$ such that $\sigma'' <_\sigma \sigma'\}$.

Example 9 Consider the domain specification D_2. Let $\sigma = \{alive, loaded\}$, $\sigma_1 = \{loaded, dead\}$, and $\sigma_2 = \{dead\}$. We have that

$$(\sigma_1 \setminus \sigma) \cup (\sigma \setminus \sigma_1) = \{alive, alive\}$$

and

$$(\sigma_2 \setminus \sigma) \cup (\sigma \setminus \sigma_2) = \{dead, alive, loaded\}.$$

This implies that $\sigma_1 \leq_\sigma \sigma_2$. Clearly, we also have that $\sigma_1 <_\sigma \sigma_2$. □

Based on the above intuition we now list the conditions that σ' must satisfy to be a state that can be reached by executing an action a in a state σ.

- σ' must be a valid state, defined as states that satisfy all the domain constraints. (Note that $\sigma \cup E_D^+(a, \sigma) \setminus E_D^-(a, \sigma)$ may not be a valid state.)
- $E_D^+(a, \sigma) \subseteq \sigma'$ must be true.
- $E_D^-(a, \sigma) \cap \sigma' = \emptyset$ must be true.
- σ' must be as close to σ as possible.

As before, we will require that $\Phi(a, \sigma)$ is undefined whenever a is not executable in σ. Let

$$States(D) = \{\sigma \mid \sigma \subset \mathbf{F},\ \sigma \text{ satisfies all domain constraints in } D\}.$$

The set of possible states after the execution of a in σ is defined by

$$\Omega_D(a, \sigma) = \{\sigma' \mid \sigma' \in States(D), E^+_D(a, \sigma) \subseteq \sigma', E^-_D(a, \sigma) \cap \sigma' = \emptyset\}.$$

Finally, we define $\Phi(a, \sigma)$ as the elements in $\Omega_D(a, \sigma)$ which are as close to σ as possible:

$$\Phi(a, \sigma) = \begin{cases} \min_{<_\sigma}(\Omega(a, \sigma)) & \text{if } a \text{ is executable in } \sigma \\ \emptyset & \text{if } a \text{ is not executable in } \sigma \end{cases}$$

Example 10 Consider the domain specification D_2, the action *shoot*, and the state $\sigma = \{alive, loaded\}$. We have that

- $States(D_2)$ consists of $\{alive\}$, $\{alive, loaded\}$, $\{dead\}$, and $\{dead, loaded\}$.
- $E^+_{D_2}(shoot, \sigma) = \emptyset$ and $E^-_{D_2}(shoot, \sigma) = \{alive\}$.
- $(\sigma \cup E^+_{D_2}(shoot, \sigma)) \setminus E^-_{D_2}(shoot, sigma) = \{loaded\}$. Observe that $\{loaded\} \notin States(D_2)$ becauses it violates the domain constraint **always** $dead \Leftrightarrow \neg alive$.
- Since $E^+_{D_2}(shoot, \sigma) \subseteq \sigma'$ and $\sigma' \cap E^-_{D_2}(shoot, \sigma) = \emptyset$ for every element σ' in $\Omega(shoot, \sigma)$ and the only two states in $States(D_2)$ satisfying these conditions are $\{dead\}$ and $\{dead, loaded\}$, we have that $\Omega(shoot, \sigma)) = \{\{dead\}, \{dead, loaded\}\}$.
- Example 9 shows that $\{dead, loaded\} <_\sigma \{dead\}$. Thus, $\Phi(shoot, \sigma) = \mathtt{min}_{<_\sigma}(\Omega(shoot, \sigma)) = \{\{dead, loaded\}\}$.

□

Observe that transitions between states of $\mathcal{AR}$ action theories could be non-deterministic.

Example 11 Consider the domain specification D_3,

$$D_3 = \left\{ \begin{array}{l} toss \textbf{ causes } tossed \\ \textbf{always } tossed \Leftrightarrow head \bigoplus tail \end{array} \right\}$$

where $\bigoplus$ is the exclusive-or operation. Here, $States(D_3) = \{\emptyset, \{tossed, head\}, \{tossed, tail\}\}$ and $\{tossed\}$ is not a state.

If the action *toss* is executed in the state $\emptyset$, then, intuitively, the resulting states may be $\{tossed, head\}$ or $\{tossed, tail\}$. This is verified by $\Phi_{D_3}(toss, \emptyset) = \{\{tossed, head\}, \{tossed, tail\}\}$. □

The semantics of $\mathcal{AR}$ action theory will be defined by modifying Definitions 1, 4, 5, and 6 to take into account domain constraints. In particular, let (D, O) be an $\mathcal{AR}$ action theory. We say that

- σ_0 is an initial state if $\sigma_0 \in States(D)$ and σ_0 satisfies all observations in O.
- the extended transition function $\widehat{\Phi}$ of a transition function Φ is defined by
  - $\widehat{\Phi}([\,], \sigma) = \{\sigma\}$

– if $\widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], \sigma) = \emptyset$ or $\Phi(a_k, s') = \emptyset$ for some $s' \in \widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], \sigma)$ then

$$\widehat{\Phi}([a_1 \circ \ldots \circ a_k], \sigma) = \emptyset.$$

– if $\widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], \sigma) \neq \emptyset$ and $\Phi(a_k, s') \neq \emptyset$ for every $s' \in \widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], \sigma)$ then

$$\widehat{\Phi}([a_1 \circ \ldots \circ a_k], \sigma) = \bigcup_{\sigma' \in \widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], \sigma)} \Phi(a_k, \sigma').$$

- (σ_0, Φ) is a model of (D, O) if σ_0 is an initial state and Φ is a transition function of D.
- $(D, O) \models \varphi$ **after** α iff for every model (σ_0, Φ) of (D, O), $\widehat{\Phi}(\alpha, \sigma_0) \neq \emptyset$ and φ holds in *every state* belonging to $\widehat{\Phi}(\alpha, \sigma_0)$.

2.6.2 Why Classical Constraints Are Not Enough?

Domain constraints allows us to compactly represent many interesting constraints between fluents. The following domaind specification, taken from [Lin95], shows that the use of classical logic formula to represent constraints might sometimes lead to counterintuitive results. In this domain, we have a suitcase with two lock and a spring loaded mechanism which will open the suitcase when both of the locks are in the up position. We can flip the lock to put it in the up position. This is detailed in the next example.

Example 12 (Lin's Suitcase) Consider the action theory (D_4, O_4) where

$$D_4 = \left\{ \begin{array}{l} \textbf{always } up_1 \wedge up_2 \Rightarrow open \\ flip_1 \textbf{ causes } up_1 \\ flip_2 \textbf{ causes } up_2 \end{array} \right\}$$

and $O_4 = \{\textbf{initially } up_1, \neg up_2, \neg open\}$.

There are eight possible states of the world

$$\begin{array}{ll} \sigma_1 = \emptyset & \sigma_5 = \{open\} \\ \sigma_2 = \{up_1\} & \sigma_6 = \{open, up_1\} \\ \sigma_3 = \{up_2\} & \sigma_7 = \{open, up_2\} \\ \sigma_4 = \{up_1, up_2\} & \sigma_8 = \{open, up_1, up_2\} \end{array}$$

of which σ_4 does not satisfy the constraint $up_1 \wedge up_2 \Rightarrow open$. Thus we have that

$$States(D_4) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_5, \sigma_6, \sigma_7, \sigma_8\}$$

and σ_2 is the initial state of (D_4, O_4). We are interested in computing $\Phi_{D_4}(flip_2, \sigma_2)$. We have that

- $E^+_{D_4}(flip_2, \sigma_2) = \{up_2\}$.
- $E^-_{D_4}(flip_2, \sigma_2) = \emptyset$.
- $(\sigma_1 \cup E^+_{D_4}(flip_2, \sigma_2)) \setminus E^-_{D_4}(flip_2, \sigma_2) = \{up_1, up_2\}$ and is not a state.
- $\Omega(flip_2, \sigma_2) = \{\sigma_3, \sigma_7, \sigma_8\}$.

- $(\sigma_3 \setminus \sigma_2) \cup (\sigma_2 \setminus \sigma_3) = \{up_1, up_2\}$.
- $(\sigma_7 \setminus \sigma_2) \cup (\sigma_2 \setminus \sigma_7) = \{open, up_1, up_2\}$.
- $(\sigma_8 \setminus \sigma_2) \cup (\sigma_2 \setminus \sigma_8) = \{open, up_1, up_2\}$.
- $\Phi_{D_4}(flip_2, \sigma_2) = \{\sigma_7, \sigma_8\}$.

Thus we have that executing $flip_2$ in the state $\{up_1\}$ results in either $\{up_2, open\}$ or $\{up_2, open, up_1\}$. In both states, up_2 (the direct effect of $flip_2$) is true and $open$ (the intended indirect effect of $flip_2$) is true as well. However, in one state up_1 becomes false. In other words, $flip_2$ also has another *unintended* indirect effect: it makes up_1 false in one of its possible successor states. □

The main reason for the counterintuitive result in the above example lies in the fact that $up_1 \wedge up_2 \Rightarrow open$ is logically equivalent to $\neg open \wedge up_2 \Rightarrow \neg up_1$. The later constraint stipulates that if up_2 is true, either $open$ or up_1 should be false. This is certainly not our intention in the specification of the domain constraint **always** $up_1 \wedge up_2 \Rightarrow open$. This demonstrates that relation6s between the fluents in an environment does not always behave like a classical constraints. Several proposals have been developed to address this issue [Bar95, Lin95, LR94a, MT95, Sha99, San96, McI00, Thi97]. Many of these proposals adopt the 'one-directional' specification of the relation between fluents. We will discuss one of these proposals in the next subsection.

2.6.3 Adding Static Causality to $\mathcal{A}$ — The Language $\mathcal{B}$

The language $\mathcal{B}$ is first presented in [GL98]. Like $\mathcal{AR}$, the syntax of the language $\mathcal{B}$ is also a superset of $\mathcal{A}$. Instead of constraint of the form (2.6), $\mathcal{B}$ introduces a new type of propositions, called *static causal axioms* (also known as *static causal laws* or *causal laws*), of the form

$$\varphi \textbf{ s\_causes } l \tag{2.7}$$

where φ is a set of fluent literals (called the *body* of the axiom) and l is a fluent literal (called the *head*) or the special symbol $\perp$ which denotes *false*. Intuitively, (2.7) says that whenever φ holds so is l. For $\perp$ means to represent $false$, a proposition of the form φ **s_causes** $\perp$ implies that no state of the world satisifes φ. In the following, we will assume that the body of any static causal axiom is a nonempty set.

Remark 3 Static causal axioms can be used to express the mutual exclusive property between fluents. For example, the two axioms

$$dead \textbf{ s\_causes } \neg alive \qquad\qquad alive \textbf{ s\_causes } \neg dead$$

expresses the fact that *dead* and *alive* cannot be true at the same time.

For convenience, we will use the shorthand $l_1 \oplus \ldots \oplus l_n$ to denote the set of static causal axioms consisting of

- l_i **s_causes** $\neg l_j$ for each pair $i \neq j$ and
- $\bigwedge_{j=1, j\neq i}^{n} \neg l_j$ **s_causes** l_i for each i.

In essense, this set of static causal axioms state that $l_1, \ldots, l_n$ are mutual exclusive.

It should be emphasized that the intended use of a static causal axiom of the form (2.7) differs from a domain constraint in that it is used only in one direction. The semantics of the language $\mathcal{B}$ is formulated, as in the case of $\mathcal{A}$ or $\mathcal{AR}$, by defining the notion of a state and transitions between states. As in the case of $\mathcal{AR}$, the execution of an action in a state can possibly result in several states. The key idea in defining the semantics of $\mathcal{B}$ is that each fluent literal in the resulting state must have a reason for its existence. A fluent literal can be true if it is a *directly effect* of an action; an *indirectly effect* of an action; or *by inertia*. Before we formally define these notions, let us introduce some additional notions.

Let D be a domain specification in $\mathcal{B}$. A set of fluent literals s is said to be *consistent* if it does not contain f and $\neg f$ for some fluent f. It is *complete* with respect to a set of fluents X if $s \cap \{f, \neg f\} \neq \emptyset$ for every $f \in X$. An *interpretation* I of the fluents in D is a maximal consistent set of fluent literals of D. A fluent f is said to be true (resp. false) in I iff $f \in I$ (resp. $\neg f \in I$). The truth value of a fluent formula in I is defined recursively over the propositional connectives in the usual way. For example, $f \wedge g$ is true in I iff f is true in I and g is true in I. We say that a formula φ holds in I (or I satisfies φ), denoted by $I \models \varphi$, if φ is true in I.

Let u be a consistent set of fluent literals and K a set of static causal axioms. We say that u is closed under K if for every static causal axiom

$$\varphi \textbf{ s\_causes } l$$

in K, if $u \models \varphi$ then $u \models f$. By $Cl_K(u)$ we denote the least consistent set of literals from D that contains u and is also closed under K. We call $Cl_K(u)$ *the closure of u with respect to K*. It is worth noting that $Cl_K(u)$ might be undefined. For instance, if u contains both f and $\neg f$ for some fluent f, then $Cl_K(u)$ cannot contain u and be consistent; another example is that if $u = \{f, g\}$ and K contains

$$f \textbf{ s\_causes } h \quad \text{and} \quad f, g \textbf{ s\_causes } \neg h,$$

then $Cl_K(u)$ does not exist because it has to contain both h and $\neg h$, which means that it is inconsistent. Abusing the notation, for a domain specification D and a set of fluent literals u, we write $Cl_D(u)$ to denote the closure of u with respect to the set of static causal axioms in D. Before we go on, we prove a property of the closure operator Cl_D.

For a set of literals σ, let

$$\Gamma(\sigma) = \sigma \cup \{l \mid \exists \varphi \textbf{ s\_causes } l \in D, \text{ such that } \varphi \subseteq \sigma\}. \tag{2.8}$$

Let $\Gamma^0(\sigma) = \Gamma(\sigma)$ and $\Gamma^{i+1}(\sigma) = \Gamma(\Gamma^i(\sigma))$ for $i \geq 0$. Since, by the definition of Γ, for any set of literals σ' we have $\sigma' \subseteq \Gamma(\sigma')$, the sequence $\langle \Gamma^i(\sigma) \rangle_{i=0}^{\infty}$ is monotonic with respect to the set inclusion operation. In addition, $\langle \Gamma^i(\sigma) \rangle_{i=0}^{\infty}$ is bounded by the set of fluent literals. Thus, there exists $\sigma^{\texttt{limit}}$ such that $\sigma_D^{\texttt{limit}} = \bigcup_{i=0}^{\infty} \Gamma^i(\sigma)$. Furthermore, $\sigma_D^{\texttt{limit}}$ is unique and satisfies all static causal axioms in D.

Lemma 1 For any set of literals σ, we have $\sigma_D^{\texttt{limit}} = Cl_D(\sigma)$.

Proof. By induction we can easily show that $\Gamma^i(\sigma) \subseteq Cl_D(\sigma)$ for all $i \geq 0$. Hence, we have

$$\sigma_D^{\texttt{limit}} \subseteq Cl_D(\sigma)$$

Furthermore, from the construction of $\Gamma^i(\sigma)$, it follows that $\sigma^{\texttt{limit}}$ satisfies all static causal laws in D. Because of the minimality property of $Cl_D(\sigma)$, we have

$$Cl_D(\sigma) \subseteq \sigma_D^{\texttt{limit}}$$

Hence, we have

$$\sigma_D^{\texttt{limit}} = Cl_D(\sigma)$$

□

The following corollary follows immediately from the above lemma.

Corollary 1 *For two sets of literals $\sigma \subseteq \sigma'$, $Cl_D(\sigma) \subseteq Cl_D(\sigma')$.*

It is important to observe that $^{\texttt{limit}}$ is bounded by the number of fluent literals in D. So, we have

Corollary 2 *For any set of fluent literals σ, $Cl_D(\sigma)$ can be computed in polynomial time in the size of D.*

Formally, the notion of a state in $\mathcal{B}$ domain specifications is defined as follows.

Definition 7 Let D be a $\mathcal{B}$ domain specification. A *state* s of D is a set of fluent literals satisfying the following two conditions:

- s is complete, i.e., $s \cap \{f, \neg f\} \neq \emptyset$ for each $f \in \mathbf{F}$, and
- s is closed under the set of static causal axioms of D.

An action a is *possibly executable* in a state s if there exists an executability axiom

$$\textbf{executable } a \textbf{ if } \varphi$$

in D such that $s \models \varphi$.

Definition 8 The *direct effect of an action a* in a state s is the set

$$e_D(a, s) = \{l \mid a \textbf{ causes } l \textbf{ if } \varphi \in D, s \models \varphi\}.$$

At this point, it is instructive to see whether or not a straightforward extension of the semantics of $\mathcal{AR}$ is appropriate for $\mathcal{B}$. Intuitively, we could extend the definition of closeness between states as defined in Section 2.6.1 to states as defined here and require that a successor state s' of the execution of a in a state s should (*i*) contain the direct effects of a; (*ii*) satisfy *all* static causal axioms; and (*iii*) as close to s as possible. The following example shows that it is not the case.

Example 13 Consider the domain specification

$$D = \left\{ \begin{array}{l} a \textbf{ causes } g \\ f, g \textbf{ s\_causes } f \\ \neg f, g \textbf{ s\_causes } h \end{array} \right\}$$

Consider the state $s = \{\neg f, \neg g, \neg h\}$. Let s' denote a state resulting from the execution of a in s. Intuitively, g should be true in s' because g is the direct effect of a in s. g is true means that the body of one of the static causal axioms f, g **s_causes** f or $g, \neg def - trans - bf$ **s_causes** h will be satisfied by s'. Let $s_1 = \{g, f, \neg h\}$ and $s_2 = \{g, \neg f, h\}$. We can easily check that both s_1 and s_2 satisfy the static causal axioms in D and s_1 is as close to s as s_2 is. This means that s' can either be s_2 or s_1 if a straightforward generation of the definition of the transition function of $\mathcal{AR}$ to $\mathcal{B}$ were used.

Observe that the difference between s_1 and s_2 is that f is true in s_1 and false in s_2 respectively. As f is false in s, it is natural to look for a reason why f changes its value in s_1. From the axioms in D, f is true only if the body of the static causal axioms f, g **s_causes** f becomes true, i.e., something magically must happen to change the value of f. This means that f should stay false in any successor state of a in s. As such, any reasonable semantics for $\mathcal{B}$ should accept s_2 but not s_1 as a possible sussessor state of a in s. □

The key idea in defining the semantics of $\mathcal{B}$ lies in the causality principle, started by [] and further elaborated by []. Formally, $\Phi(a, s)$, the set of states that may be reached by executing a in s given a domain specification D, is defined as follows.

Definition 9 Let D be a domain specification, a be an action, and s be a state. A function Φ that maps pairs of actions and states into sets of states is a *transition function* of D if

$$\Phi(a, s) = \begin{cases} \{s' \mid s' \text{ is a state and } s' = Cl_D(e_D(a, s) \cup (s \cap s'))\} & \text{if } a \text{ is possibly executable in } s \\ \emptyset & \text{otherwise} \end{cases}$$

Similar to Theorem 1, we can show that each domain specification has a unique one transition function.

Theorem 6 For any domain specification D, there is at most one transition function.

We will write Φ_D to denote the transition function of D. The subscript will be omitted if no confusion is possible. We will say that a is executable in s if $\Phi_D(a, s) \neq \emptyset$. This is somewhat different than the notion of executability of an action in case of $\mathcal{A}$. We will return to this issue in a short while. In the definition of $\Phi_D(a, s)$, we can see that a fluent literal l can be true in s' in three ways: (i) $l \in e_D(a, s)$ (direct effects); (ii) $s \cap s'$ (inertial); and $Cn_D(e_D(a, s) \cup (s \cap s')) \setminus (e_D(a, s) \cup (s \cap s'))$ (indirect effects.).

The difference between the semantics of $\mathcal{AR}$ and $\mathcal{B}$ action theories is highlighted in the next example. Let us revisit the action theory representing the suitcase with the spring loaded mechanism in the notation of the language $\mathcal{B}$.

Example 14 (Lin's Suitcase Revisisted) Consider the action theory (D_5, O_5) where

$$D_5 = \left\{ \begin{array}{l} up_1, up_2 \textbf{ s\_causes } open \\ flip_1 \textbf{ causes } up_1 \\ flip_2 \textbf{ causes } up_2 \end{array} \right\}$$

and $O_5 = \{\textbf{initially } up_1, \neg up_2, \neg open\}$. D_5 differs from D_4 only in the use of static causal axiom to exress the spring loaded mechanism.

Let $s = \{up_1, \neg up_2, \neg open\}$. Clearly, $flip_2$ is executable in s and $e_{D_5}(flip_2, s) = \{up_2\}$. Consider $u = \{open, up_2, \neg up_1\}$. We have that u is also a state of D_5. Furthermore

$$Cl_{D_5}(e_{D_5}(fip_2, s) \cup (s \cap u)) = Cl_{D_5}(\{up_2\}) = \{up_2\} \neq u$$

which indicates that $u \notin \Phi_{D_5}(flip_2, s)$. On the other hand, for $v = \{open, up_2, up_1\}$,

$$Cl_{D_5}(e_{D_5}(fip_2, s) \cup (s \cap v)) = Cl_{D_5}(\{up_1, up_2\}) = \{up_1, up_2, open\} = v$$

which indicates that $v \in \Phi_{D_5}(flip_2, s)$. □

The next example shows that even though we do not consider indeterminate actions, the transions caused by actions in $\mathcal{B}$ action theories can still be non-deterministic.

Example 15 (Non-deterministic Transitions) Consider the domain specification

$$D_6 = \left\{ \begin{array}{l} f, g \textbf{ s\_causes } \neg h \\ f, h \textbf{ s\_causes } \neg g \\ make_f \textbf{ causes } f \end{array} \right\}$$

Let $s = \{\neg f, g, h\}$. We compute $\Phi_{D_6}(make_f, s)$. We have that

- $e_{D_6}(make_f, s) = \{f\}$.
- for $u = \{f, g, \neg h\}$, $s \cap u = \{g\}$, and $Cn_{D_6}(E_{D_6}(make_f, s) \cup (s \cap u)) = Cn_{D_6}(\{f, g\}) = \{f, g, \neg h\} = u$.
- for $v = \{f, \neg g, h\}$, $s \cap v = \{h\}$, and $Cn_{D_6}(E_{D_6}(make_f, s) \cup (s \cap v)) = Cn_{D_6}(\{f, h\}) = \{f, \neg g, h\} = v$.

Thus $\Phi_{D_6}(make_f, s) = \{u, v\}$. □

Here is another example.

Example 16 Consider the following domain specification

$$D_2 = \left\{ \begin{array}{l} a \textbf{ causes } f \\ f, \neg h \textbf{ s\_causes } g \\ f, \neg g \textbf{ s\_causes } h \\ \neg f \textbf{ s\_causes } k \end{array} \right\}$$

Let $s = \{\neg f, \neg g, \neg h, k\}$. Clearly s is a state since it satisfies all static causal axioms in D_2. Executing a in s results in two possible next states

$$\Phi_{D_2}(a, s) = \{\{f, \neg g, h, k\}, \{f, g, \neg h, k\}\}$$

In the first possible next state $\{f, \neg g, h, k\}$, f holds because it is a direct effect of a, i.e., $f \in E(a, s)$; $\neg g$ and k hold because of inertia; and h holds because it is an indirect effect of a (in particular, h holds because of the static causal law $f, \neg g$ **s_causes** h).

Likewise, we can explain why each literal in the second possible next state holds. □

As we have mentioned earlier the notion of action executability in $\mathcal{B}$ is different than it was defined in $\mathcal{A}$. In the case of $\mathcal{A}$, the satisfaction of an executability axiom of an action a in a state s is enough to guarantee that a is executable in s. As it turns out, static causal axioms can also indirectly cause actions to become inexecutable. This is illustrated in the following example.

Example 17 (Ramification and Qualification) Let us consider the domain specification in which the action *kill* causes the turkey to be not *alive*, the action *make_walk* causes the turkey to be *walking*, and the relation stating that if the turkey is dead, it cannot walk. This can be encoded as follows.

$$D_7 = \left\{ \begin{array}{l} kill \textbf{ causes } \neg alive \\ make\_walk \textbf{ causes } walking \\ \neg alive \textbf{ s\_causes } \neg walking \end{array} \right\}$$

This domain specification has three states: $s_1 = \{walking, alive\}$, $s_2 = \{\neg alive, \neg walking\}$, and $s_3 = \{alive, \neq walking\}$. We have that

$$\Phi_{D_7}(kill, s_1) = \{s_2\}$$

since $s_2 \cap s_1 = \emptyset$, $e_{D_7}(kill, s_1) = \{\neg alive\}$. $Cn_{D_7}(\{\neg alive\}) = \{\neg alive, \neg walking\} = s_2$.

Let us now compute $\Phi_{D_7}(make\_walk, s_2)$. We have that $e_{D_7}(make\_walk, s_2) = \{walking\}$. Thus any state s belonging to $\Phi_{D_7}(make\_walk, s_2)$ satisfies that $e_{D_7}(make\_walk, s_2) \subseteq s$. Since the only state satisfying this condition is s_1, $s_2 \cap s_1 = \emptyset$, and $Cn_{D_7}(\{walking\}) = \{walking\} \neq s_1$, we can conclude that $\Phi_{D_7}(make\_walk, s_2) = \emptyset$. This means that *make_walk* can not be executed in s_2. □

As we can see in the above example, the presence of the static causal axiom $\neg alive$ **s_causes** $\neg walking$ prevents the action *make_walk* to be executable in the state s_2 even though the action is possibly executable in it. (Recall that by default, when no executability axiom for *make_walk* is specified,

executable *make_walk* **if** $\top$

belongs to the specification.) Obviously, it is reasonable to consider that *make_walk* should not be executable whenever $\neg alive$ is true. This leads us to the question of whether or not it would make sense for us to add to the domain specification the executability axiom **executable** *make_walk* **if** *alive*? Clearly, the presence of this executability axiom will prevent *make_walk* to be executable in s_2. What will be the benefit of not adding this executability axiom?

Adding **executable** *make_walk* **if** *alive* to the domain specification might have been good in this situation. Yet, the price we have to pay is that the specification is not elaboration tolerant, i.e., the addition of new axioms to the specification might require some modifications of the domain. For example, in addition to the information described in Example 17, we also know that the turkey is not able to walk if it is sick. Intuitively, this will have the effect of preventing *make_walk* to be executable in any state in which *sick* is true. Therefore, the executability axiom **executable** *make_walk* **if** *alive* is no longer adequate to characterize the situations, when *make_walk* is executable, and needs to be replaced with **executable** *make_walk* **if** *alive*, $\neg sick$. On the other hand, adding *sick* **s_causes** $\neg wakking$ to D_7 will achieve the same effect and does not require any changes in its previously encoded propositions.

Finally, we would like to emphasize that static causal axioms do not behave exactly as a logical implication (e.g. Example 14). In general, a static causal axiom

$$\varphi \textbf{ if } l$$

is not the same as the static causal axiom

$$\varphi, \bar{l} \textbf{ if } \bot$$

even though both disallow any interpretation satisfying $\varphi \cup \{\bar{l}\}$ to be considered as states. The former represents a ramification constraint while the latter represents a qualification constraint. The former encodes a reason for the literal l to become true, the later does not.

Example 18 (Ramification vs. Qualification) Consider the domain specifications

$$D_8 = \left\{ \begin{array}{l} f \textbf{ s_causes } \neg g \\ make\_f \textbf{ causes } f \end{array} \right\} \qquad D_9 = \left\{ \begin{array}{l} f, g \textbf{ s_causes } \bot \\ make\_f \textbf{ causes } f \end{array} \right\}$$

We have that

- $\Phi_{D_8}(make\_f, \{\neg f, g\}) = \{\{f, \neg g\}\}$. In this domain, the static causal axiom indirectly changes the value of g. In other words, $\neg g$ has a reason to become true since f becomes true.

- $\Phi_{D_9}(make\_f, \{\neg f, g\}) = \emptyset$. Here, the static causal axiom does not represent a reason for g to become false when f becomes true. As such, the value of g should not be changed, which causes the execution of $make\_f$ in the state $\{\neg f, g\}$ to result in no state.

□

Like $\mathcal{AR}$, the semantics of $\mathcal{B}$ action theories will be defined by adapting the Definitions 1, 4, 5, and 6 to take into account static causal axioms. In particular, let (D, O) be an $\mathcal{AR}$ action theory. We say that

- s_0 is an initial state if s_0 is a state of D satisfying all observations in O.
- the extended transition function $\widehat{\Phi}$ of a transition function Φ is defined by
  - $\widehat{\Phi}([\,], s) = \{s\}$
  - if $\widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], s) = \emptyset$ or $\Phi(a_k, s') = \emptyset$ for some $s' \in \widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], s)$ then
$$\widehat{\Phi}([a_1 \circ \ldots \circ a_k], s) = \emptyset.$$
  - if $\widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], s) \neq \emptyset$ and $\Phi(a_k, s') \neq \emptyset$ for every $s' \in \widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], s)$ then
$$\widehat{\Phi}([a_1 \circ \ldots \circ a_k], s) = \bigcup_{s' \in \widehat{\Phi}([a_1 \circ \ldots \circ a_{k-1}], s)} \Phi(a_k, s').$$
- (s_0, Φ) is a model of (D, O) if s_0 is an initial state and Φ is a transition function of D.
- $(D, O) \models \varphi$ **after** α iff for every model (s_0, Φ) of (D, O), $\widehat{\Phi}(\alpha, s_0) \neq \emptyset$ and φ holds in *every state* belonging to $\widehat{\Phi}_D(\alpha, s_0)$.

Due to the nondeterminism of the transition function Φ, the definition of the function $\widehat{\Phi}$ requires that a_k is executable in every state reached after the execution of $[a_1 \circ \ldots \circ a_{k-1}]$. This is necessary to guarantees that the execution of the sequence $[a_1 \circ \ldots \circ a_k]$ is successful on every possible trajectory. Let us look a simple example.

Example 19 Consider the action theory

$$D = \left\{ \begin{array}{l} a \textbf{ causes } f \\ b \textbf{ causes } t \\ \textbf{executable } b \textbf{ if } f, \neg h \\ f, \neg h \textbf{ s\_causes } g \\ f, \neg g \textbf{ s\_causes } h \end{array} \right\}$$

and

$$O = \{\textbf{initially } \neg f, \neg g, \neg h, \neg t\}$$

We have that $s_0 = \{\neg f, \neg g, \neg h\, \neg t\}$ is the unique initial state of (D, O). Furthermore $\Phi_D(a, s_0) = \{s_1, s_2\}$ where $s_1 = \{f, \neg g, h, \neg t\}$ and $s_2 = \{f, g, \neg h, \neg t\}$. b is possibly executable in s_2 but not in s_1. Hence, $\Phi_D(b, s_1) = \emptyset$ and therefore $\widehat{\Phi}_D([a \circ b], s_0) = \emptyset$. Thus, we have that $(D, O) \not\models t$ **after** $[a \circ b]$.

Observe that if we do not require that b is executable in every state belonging to $\Phi(a, s_0)$, we would have $\widehat{\Phi}([a \circ b], s_0) = \{\{f, g, \neg h, t\}\}$ which implies that $(D, O) \models t$ **after** $[a \circ b]$. This result is incorrect. □

We will next define the notion of a trajectory.

Definition 10 *Let* (D, O) *be an* $\mathcal{B}$ *action theory. A* trajectory *in* (D, O) *is a sequence of the form*

$$s_0 a_1 s_1 \ldots s_{k-1} a_k s_k$$

where s_i*'s are states,* a_i*'s are actions, and for each* i, $1 \leq i \leq k$, $s_i \in \Phi_D(a_i, s_{i-1})$.

The nondeterminism of action theories also makes the notion of a possible plan useful.

Definition 11 Let (D, O) be an $\mathcal{B}$ action theory. A sequence of actions $\alpha = [a_1 \circ \ldots \circ a_k]$ is a *possible plan* for ψ if there exists a trajectory $s_0 a_1 s_1 \ldots s_{k-1} a_k s_k$ such that s_0 is an initial state and ψ is true in s_k.

2.7 Complexity of Reasoning and Planning in $\mathcal{B}$

In this section, we provide a few complexity results that are the driving force for our development of a theory of approximation for reasoning about effects of actions in the presence of incomplete information. We will study the following problems:

- $\text{CI}^{\mathcal{B}}$ (*Consistent initial state*) Given a consistent domain specification D, a set of observations O, and a set of fluent literals s_0, determine whether s_0 is an initial state with respect to (D, O).
- $\text{CS}^{\mathcal{B}}$ (*Consistent domain*) Given a domain specification D, determine whether D is consistent or not.
- $\text{DT}^{\mathcal{B}}$ (*Determinism*) Given a domain specification D, determine whether D is deterministic or not.
- $\text{AC}^{\mathcal{B}}$ (*Executable*) Given a domain specification D, and action a, and a state s, determine whether or not a is executable in s, i.e., $\Phi(a, s) \neq \emptyset$.
- $\text{PC}^{\mathcal{B}}$ (*Planning with complete initial state*) Given a consistent domain specification D, a set of observations about the initial state O, and a fluent literal l, a polynomial function $f : N \rightarrow N$, determining a plan c, whose size is bounded by $f(|D|)$, such that $(D, O) \models l$ **after** c.

Since the number of fluents in a domain specification is finite and the closure $Cl_D(u)$ can be computed in polynomial time in the size of the domain, we have the following theorem.

Theorem 7 $\text{CI}^{\mathcal{B}}$ is polynomial.

Before we discuss the complexity of the other problems, let us prove some auxiliary results. Let F be a 3SAT formula over the set of atoms $\{a_1, \ldots, a_n\}$,

$$F = \bigwedge_{i=1}^{k} (g_i \vee f_i \vee h_i)$$

where g_i, f_i, h_i is either an atom a or its negation $\neg a$. in F. Let Π_F be the logic program containing the following rules:

- For each i, Π_F contains three rules:

$$g_i(1) \leftarrow z_i(1), \overline{f_i(1)}, \overline{h_i(1)} \tag{2.9}$$
$$f_i(1) \leftarrow z_i(1), \overline{g_i(1)}, \overline{h_i(1)} \tag{2.10}$$
$$h_i(1) \leftarrow z_i(1), \overline{f_i(1)}, \overline{g_i(1)} \tag{2.11}$$

- For each atom a_i, Π_F contains the following rules:

$$a_i(1) \leftarrow a_i(0),\ not\ \overline{a_i(1)} \tag{2.12}$$
$$\overline{a_i(1)} \leftarrow \overline{a_i(0)},\ not\ a_i(1) \tag{2.13}$$
$$\leftarrow a_i(1), \overline{a_i(1)} \tag{2.14}$$
$$\leftarrow a_i(0), \overline{a_i(0)} \tag{2.15}$$

- For each i, Π_F contains also the following rule:

$$z_i(1) \leftarrow occ(t, 0) \tag{2.16}$$

  where $t, z_1, \ldots, z_k$ are constants that do not appear in F.

Let s be an interpretation of variables in the language of F. By $s(1)$ we denote the set $\{l(0) \mid l \in s\}$. We prove the following:

Lemma 2 If F is satisfiable, then $P = \Pi_F \cup \{occ(t, 0)\} \cup s(0)$ is consistent for every interpretation s of the variables in F.

Proof. Let $Z = \{z_i(1) \mid i = 1, \ldots, k\}$ Two cases:

- s satisfies F. Then, $s(1) \cup s(0) \cup \{occ(t, 0)\} \cup Z$ is an answer set of P.

- s does not satisfy F. Let $K = \{s' \mid s'$ is an interpreation of $\{a_1, \ldots, a_n\}$ which satisfies $F\}$. For each interpretation $s' \in K$, let $v(s', s) = \{l \mid l \in s', \bar{l} \in s\}$. F is satisfiable implies that $K \neq \emptyset$. Let $s' \in K$ such that $v(s', s)$ is minimal (with respect to $\subseteq$) among all $s'' \in K$. We will show that $S = s'(1) \cup s(0) \cup \{occ(t, 0)\} \cup Z$ is an answer set of P.

  It is easy to see that S satisfies all the rules of P. Let $Q = P^S$. By definition, Q contains the rules (2.9)-(2.11), (2.14)-(2.16) of P, the rule

$$l(1) \leftarrow l(0) \tag{2.17}$$

  for each $l \in s \cup s'$, and the rule

$$\overline{l(1)} \leftarrow \overline{l(0)} \tag{2.18}$$

  for each $l \in s' \setminus s$. Let X be the minimal model of Q. We want to show that $X = S$. Assume the contrary, $X \neq S$. Since S satisfies the rules of Q, $X \subsetneq S$, i.e., there exists some $l \in S \setminus X$. It is easy to see that l is of the form $a(1)$ or $\overline{a(1)}$) for some $a \in v(s', s)$ or $\neg a \in v(s', s)$.

  Let R_l be the set of rules in Q, which is of the form (2.9)-(2.11), whose head is l. $l \notin X$ implies that there exists no rule in R_l whose body is satisfied by X. This means that for each disjunct $\varphi_i = f_i \vee h_i \vee g_i$ and $l \in \{f_i, g_i, h_i\}$ there exists some $l' \in \{f_i, g_i, h_i\}$ such that $l' \in X$ and $l \neq l'$. This means that $s' \setminus \{l\} \cup \{\neg l\}$ satisfies F. This is a contradiction with the minimality of $v(s', s)$.

  The above shows that S is an answer set of P.

□

Lemma 3 If F is unsatisfiable, then $P = \Pi_F \cup \{occ(t, 0)\} \cup s(0)$ is inconsistent for every interpretation s of the variables in F.

Proof. Assume the contrary, P has an answer set X. It is easy to see that $\{l \mid l(1) \in X\}$ is an interpreation of variables in X which satisfies F. Contradiction. □

Let F be a 3NF formula over the variables $a_1, \ldots, a_n$

$$F = \bigwedge_{i=1}^{k} (g_i \vee f_i \vee h_i)$$

where g_i, f_i, h_i is either an atom a or its negation $\neg a$. Let D_F be the domain specification with the action t, the fluents $a_1, \ldots, a_n, z_1, \ldots, z_k$, and the fluent effect axiom

$$t \textbf{ causes } z_i$$

and for each i, the set of static causal axioms

$$z_i, \overline{f_i}, \overline{g_i} \textbf{ s\_causes } h_i$$
$$z_i, \overline{f_i}, \overline{h_i} \textbf{ s\_causes } g_i$$
$$z_i, \overline{h_i}, \overline{g_i} \textbf{ s\_causes } f_i$$

From Lemma 2 and 3, we can show the following:

Lemma 4 F is satisfiable iff D_F is consistent.

This allow us to prove the following result.

Theorem 8 $\mathrm{AC}^{\mathcal{B}}$ is NP-complete.

Proof. The problem can be represented as a formula $\exists s'.s' \in \Phi(a, s)$. So, membership of $\mathrm{AC}^{\mathcal{B}}$ in NP-complete follows. Hardness is proved by reducing a 3SAT formula F to D_F as described in Lemma 4 and takes and arbitrary s, t as the action, and $\Phi(a, s) \neq \emptyset$ iff F is satisfiable. □

Let $\bigvee_{i=1}^{n} l_i$ be a disjunction and z be an atom not occurring in it. By $L_{\bigvee_{i=1}^{n} l_i; z}$, we denote the set of static causal axioms

$$\{z, \bigwedge_{1 \leq i \leq n, i \neq j} \overline{l_i} \textbf{ s\_causes } l_j \mid 1 \leq j \leq n\}$$

Theorem 9 $\mathrm{CS}^{\mathcal{B}}$ *is* $\Pi_2 P$*-complete.*

Proof. By definition, a domain specification D is consistent if $\Phi_D(a, s) \neq \emptyset$ for every pair of action a and state s such that a possibly executable in s. It is easy to see that checking whether a state $s' = Cn_D(u)$ for a state s' and a set of literals u can be done in polynomial time inthe size of $|D|$.

(i) **Membership:** The problem can be represented by the formula $\forall s, a \exists s'.[R(a, s) \supset Q(a, s, s')]$, where $R(a, s)$ is true iff a is possible executable in s and $Q(a, s, s')$ is true if $s' \in \Phi(a, s)$. It is easy to see that checking whether or not $R(a, s) \supset Q(a, s, s')$ holds takes polynomial time. Thus, $\mathrm{CS}^{\mathcal{B}}$ belongs to the class of $\Pi_2 P$ problems.

(ii) **Hardness:** We will reduce the problem of determining the satisfiability of the formula

$$F = \forall \vec{x} \exists \vec{y} P(\vec{x}, \vec{y})$$

where $P(\vec{x}, \vec{y})$ is a propostional formula over the variables in $\vec{x} = \{x_1, \ldots, x_k\}$ and $\vec{y} = \{y_1, \ldots, y_m\}$. We will assume that $P(\vec{x}, \vec{y})$ is given in the form

$$f_1^1 \vee f_1^2 \vee f_1^3 \wedge \ldots \wedge f_n^1 \vee f_n^2 \vee f_n^3$$

where each each of the conjunct of P contains variables from both $\vec{x}$ and $\vec{y}$. Let X_i be the disjunct obtained from $f_i^1 \vee f_i^2 \vee f_i^3$ by removing elements of the from y_j or $\neg y_j$ where $y_j \in \{y_1, \ldots, y_m\}$. Similarly, Y_i be the disjunct obtained from $f_i^1 \vee f_i^2 \vee f_i^3$ by removing elements of the from x_j or $\neg x_j$ where $x_j \in \{x_1, \ldots, x_k\}$. Without the lost of generality, we can assume that $Y_i \neq \top$ for every i.

We will construct the domain specification D_F as follows.

- The fluents of the domain are the variables in $\vec{x}$ and $\vec{y}$ and for each i, $1 \leq i \leq n$, a symbol z_i.
- D_F has one action, a.
- For every i, D_F contains the following fluent effect axioms:

$$a \textbf{ causes } z_i \textbf{ if } \neg X_i$$
$$a \textbf{ causes } \neg z_i \textbf{ if } X_i$$

- For every i,
  - if $Y_i = l$ then D_F contains the static causal axiom z_i **s_causes** l.
  - if $Y_i = l_1 \vee l_2$ then D_F contains the static causal axiom $L_{l_1 \vee l_2; z_i}$.
  - if $Y_i = l_1 \vee l_2 \vee l_3$ then D_F contains the static causal axiom $L_{l_1 \vee l_2 \vee l_3; z_i}$.

Clearly, the reduction is polynomial in the size of F. Using the techniques in Lemmas 2 and 3, we can prove that F is satisfiable iff D_F is consistent. This proves the hardness of the problem.

□

Theorem 10 $\mathrm{DT}^{\mathcal{B}}$ is coNP-complete.

Proof. See **??** (Turner) The problem can be represented as a formula

$$\forall s, a, s', s''(s' \in \Phi(a, s') \wedge s'' \in \Phi(a, s'') \supset s' = s'').$$

□

Theorem 11 $\mathrm{PC}^{\mathcal{B}}$ is $\Sigma_3 P$-complete.

Proof. See **??** (Turner or perhaps Eiter) Given an action theory (D, O) and a literal l, The problem of checking whether there exists a plan of length n for l can be represented by the formula

$$\exists a_1, \ldots, a_n \forall s_0, \ldots, s_n \exists s_1', \ldots, s_n' \left[(s_0 \in \Sigma_0) \wedge \bigwedge_{i=1}^{n} (s_i' \in \Phi(a_1, s_{i-1})) \wedge \bigwedge_{i=1}^{n} (s_i \in \Phi(a_1, s_{i-1})) \wedge (l \in s_n) \right]$$

where the a_i's are actions, s_i and s_i' are staes. This formula basically states that for every initial state s_0, the sequence $[a_1 \circ \ldots \circ a_n]$ is executable in s_0. Furthermore, every state reachable from s_0 must satisfy l. □

The quantification over s_i' in the formula representing the planning problem can be removed if the action theory is deterministic. This leads to the following result.

Corollary 3 For deterministic action theory, $\text{PC}^{\mathcal{B}}$ is $\Sigma_2 P$, if $\Phi(a, s) \neq \emptyset$ can be determined in polynomial time.

2.8 Reasoning and Planning in $\mathcal{B}$ Using Answer Set Programming

In this section, we describe a translation of $\mathcal{B}$ action theories to logic programs which can be used for reasoning and planning in $\mathcal{B}$ theories. Similar to the translation in Section 2.5, we will translate a theory (D, O) into a logic program $\pi(D, O, n)$. The program is developed so that its answer sets can be computed using available answer set solvers such as SMODELS [SNS02], DLV [CEF+97]. It can be used in reasoning about the effects of action sequences which contain at most n actions. It can also be used in generating plans consisting of at most n actions.

The main predicates of program $\pi(D, O, n)$ are:

- $step(T)$ is true if $0 \leq T \leq n$.
- $fluent(F)$ is true if F is a fluent.
- $action(A)$ is true if A is an action
- $holds(L, T)$ is true if literal L holds at T.
- $possible(A, T)$ is true if action A is executable at T.
- $occ(A, T)$ is true if action A occurs at T.

To simplify the representation of the program, we use a fix set of variables. They are:

- F is used to detenote a *fluent* variable. This means that for each rule containing the variable F, the body contains the atom $fluent(F)$.
- T denotes a *step* variable.

We will also use $holds(\varphi, T)$ as a shorthand for $holds(l_1, T), \ldots, holds(l_n, T)$ where $\varphi = \{l_1, \ldots, l_n\}$.
The program has the following facts to define variables of sort *step*:

$$step(0) \leftarrow$$
$$\ldots$$
$$step(n) \leftarrow$$

For each action a and fluent f in the domain, the program $\pi_(D, O, n)$ contains

$$action(a) \leftarrow$$

and

$$fluent(f) \leftarrow$$

respectively.

The main rules of $\pi(D, O, n)$ are given next.

- **Rules encoding the initial situation.** For each value proposition

  initially l

  in O, $\pi(D, O, n)$ contains the fact

  $$holds(l, 0) \leftarrow \tag{2.19}$$

- **Rules encoding actions' executability axiom.** For each executability axiom

  executable a **if** φ

  in D, $\pi(D, O, n)$ contains the rules

  $$possible(a, T) \leftarrow holds(\varphi, T) \tag{2.20}$$
  $$\leftarrow occ(a, T),\ not\ possible(a, T) \tag{2.21}$$

  The first rule states that if φ holds then a is executable. The second rule, called a *constraint*, prevents a to execute if its executability axiom is not satisfied.

- **Rules for reasoning about the effect of actions.** For each fluent effect axiom

  a **causes** l **if** φ

  in D, the following rule is added to $\pi(D, O, n)$:

  $$holds(l, T) \leftarrow occ(a, T), holds(\varphi, T) \tag{2.22}$$

  This rule states that if φ holds at T and a is executed at T then l is true at $T + 1$.

- **Rules for reasoning about static causal axiom.** For each static causal axiom

  φ **s_causes** l

  in D, $\pi(D, O, n)$ contains the rules

  $$holds(l, T) \leftarrow holds(\varphi, T) \tag{2.23}$$

- **Inertial rules.** For each fluent F, $\pi(D, O, n)$ contains the following rules

  $$holds(F, T + 1) \leftarrow holds(F, T),\ not\ holds(\neg F, T) \tag{2.24}$$
  $$holds(\neg F, T + 1) \leftarrow holds(\neg F, T),\ not\ holds(F, T) \tag{2.25}$$
  $$\leftarrow holds(F, T), holds(\neg F, T) \tag{2.26}$$

  When used in conjunction with (2.22), these rules define the transition function Φ.

As with the translation in Section 2.5 (Theorems 4 and 5), we will now establish the correctness of the translation from (D, O) to $\pi(D, O, n)$. Since $\mathcal{B}$ action theories might be nondeterministic, the formulation of the corresponding theorems relies on the notion of a trajectory and a possible plan, instead of a

2.8.1 $\pi(D, O, n)$ in Hypothetical Reasoning

Let $Q = l$ **after** $[a_1 \circ \ldots \circ a_t]$ be a query and $occs(Q) = \{occ(a_i, i-1) \mid 1 \leq i \leq t$. The correctness of $\pi(D, O, n)$ is prove in the following theorem.

Theorem 12 Let (D, O) be a $\mathcal{B}$ consistent theory and $Q = l$ **after** $[a_1 \circ \ldots \circ a_t]$ be a query. Then,

$$(D, O) \models l \textbf{ after } [a_1 \circ \ldots \circ a_t]$$

iff every answer set of $\pi(D, O, t) \cup occs(Q)$ contain $holds(l, t)$.

Proof. See Section [SBTM06]. □

2.8.2 $\pi(D, O, n)$ in Planning

Consider the case where $\pi(D, O, n)$ can also be used in generating plans achieving a conjunction of literals ψ (or goal) with at most n actions. To do it, we add the following rules to $\pi(D, O, n)$:

$$1\ \{occ(X, T) : action(X)\}\ 1 \leftarrow not\ goal(T) \tag{2.27}$$
$$goal(T) \leftarrow holds(\psi, T) \tag{2.28}$$
$$goal(T + 1) \leftarrow not\ goal(T) \tag{2.29}$$
$$\leftarrow not\ goal(n) \tag{2.30}$$

The first rule, often reffered as a *choice rule*, generates action occurrences. Intuitively, this rule says that if the goal has not been satisfied at the step T then exactly one atom of the form $occ(A, T)$ must be true. The second and third rules state that the goal holds whenever ψ holds. The last rule is a constraint that makes sure that ψ holds at the step n. We denote with $\Pi(D, O, n, \psi)$ the program consisting of $\pi(D, O, n)$ and the rules (2.27)–(2.30). For a set of atoms in the language $\Pi(D, O, n, \psi)$, let $\alpha(M)$ be the sequence $[a_1 \circ \ldots \circ a_t]$ such that $occ(a_i, i-1) \in M$. We have that

Theorem 13 Let (D, O) be a consistent $\mathcal{B}$ action theory, ψ be a set of fluent literals, and M be an answer set of $\Pi(D, O, n, \psi)$. Then, $\alpha(M)$ is a possible plan for ψ.

Proof. See [SBTM06]. □

2.9 From Fluent Literals to Fluent Formulas in $\mathcal{B}$ Action Theories

Unlike the language $\mathcal{AR}$, $\mathcal{B}$ does not allow fluent formulas in the axioms. This omission is intended to simplify the presentation of the semantics of $\mathcal{B}$. In order to allow fluent formulas in $\mathcal{B}$ action theories, the following changes need to be made:

Syntactically, l and φ in axioms of the form 2.1, 2.2, 2.7, and 2.3 can be formulas. Furthermore, the following modifications need to be made.

- A set of fluent literals σ satisfies a static causal axiom φ **s_causes** ψ iff $\sigma \models \varphi$ implies $\sigma \models \psi$;
- A set of fluent literals is closed under a set of static causal axioms R if it satisfies all the axioms in R;

- A set of fluent literals σ is complete with respect to a formula φ if for every fluent literal l occurring in φ, either $l \in \sigma$ or $\bar{l} \in \sigma$;
- The closure of a formula σ, denoted by $Cl_D(\sigma)$, is a minimal set of fluent literals that is complete with respect to σ and is closed under the set of static causal axioms in D;
- An actiom a is possibly executable in a state s if there exists an executability axiom **executable** a **if** φ and $s \models \varphi$;
- The direct effects of an action a in a state s is the conjunction $\bigwedge_{\psi \in \Omega} \psi$ where $\Omega = \{\psi \mid$ there exists a **causes** $\psi \in \varphi$ and $s \models \varphi\}$;
- The transition function Φ of a domain specification D is defined by $\Phi(a, s) = \{s' \mid s' = Cn_D(e(a, s) \wedge \bigwedge_{l \in s' \setminus s} l\}$;
- A state s is an initial state of an action theory (D, O) if $s \models \varphi$ for every **initially** φ in O.

Since the use of fluent formulas in the representation yields a more compact representation of domain specifications, we will often use them in our examples. It should be noted that, however, fluent formulas may not be needed in describing nondeterministic actions.

2.10 Circumpscription Encoding of $\mathcal{B}$ Action Theories

2.11 Bibliographic Notes

- Talk about $\mathcal{A}$
- Talk about $\mathcal{B}$
- Encoding of action theories in SAT and correctness
- Why is it easier to encode $\mathcal{B}$ theories in LP than in SAT

  present an encoding of $\mathcal{B}$ action theories in logic programming

  Section [**?**] describes a translation of $\mathcal{A}$ action theories into a SAT instance, which can be used in hypothetical reasoning and planning.

  The discussion in the previous section indicates that a straightforward extension of

  Will

  Can the same be done for $\mathcal{B}$ theories? At the first sight, it seems so simple: all we need to do is to encode the static causal axioms in D to the theory $tr\_do(D, O, n)$ and the rest would follow. In particular, each static causal axiom

  $$l \textbf{ if } \varphi$$

  could be translated into a set of implications

  $$\varphi[i] \Rightarrow l[i]$$

  for $1 \leq i \leq n$ in $tr\_do(D, O, n)$. Unfortunately, this will not work due to the fact that an implication

  Examples in Sub-Section 2.6.2 show that, unfortunately, this does not work.

  In this section, we will describe a new implementation that can be

- Extending $\pi(D, O, n)$ to deal with arbitrary formula can be found in [SBTM06].

Chapter 3

Representing Observations

The previous chapter discusses various languages for specifying the effects of actions and relationships among fluents of the environment. An agent equiped with an action theory describing his surrounding environment can reason about its actions and make plan to achieve a goal. In a perfect world, it is sufficient for the agent to actually achieve any goal within its capability as he—given any goal—can start by observing the environment, making a plan, and executing the plan to achieve the goal. The following story reveals that life for autonomous agents might be more complicated than that.

> John is at at home. He has a car which he parked on the drive way after coming back from work yesterday night. His suitcase is unpacked. He needs to go to the airport with a packed suitcase. He makes the plan to pack and then drive to the airport. After packing he gets out of the house only to *see* that his car is no longer there.

Obviously, for John to achieve his goal, he needs to be able to incorporate his observation(s) over various situtions into his reasoning process. This chapter presents a language for representing and reasoning with narratives. Section 3.1 presents the syntax of a language for representing observations. Section 3.2 describes the language $\mathcal{L}$ for representing and reasoning with narratives. Section 3.3 discusses $\mathcal{L}_Q$, a query language for $\mathcal{L}$.

3.1 $\mathcal{L}_O$ — An Observation Language

Observations are made in different situations. For example, in the beginning of the introductory example, John observes that his suitcase is unpacked. He observes that his car disappeared after he packed his suitcase and got out of the house. To be able to represent John's knowledge about the state of the world, an observation language needs to allow for the representation of the state of the world in different situations. We will use a set **S** of constants, called *situation constants* (or *situations*), to denote the situations for which an agent has some observations. We assume two distinguised special situation constants s_0 and s_c which denote the *initial* and *current* situations respectively. We will also assume that agents do not have observations about s_c. Situation constants will be written as s (possibly with subscripts) and are different from states which are introduced in the previous chapters. A situation encodes a history of the world from the initial situation while a state is an interpretation of fluents at a particular situation.

There are different types of observations that an agent can make. For example,

- John notes that his car disappeared. This is an example of an observation about the truth value of a fluent (or a fluent formula) at a particular situation.
- Mary drives from her house to the school to in the morning. This is an example of an observation about the occurrences of an action (or action sequence) between two situations. The action in this example is *driving* which occurs between the situation in which Mary is at home and the situation in which she is at the school.
- Mary sees John runs out of the house. This is an example of an observation about the occurrence of an action (or action sequence) at a situation. Unlike the observation in the previous example, Mary does not observe the results of the action.
- Jimmy knows that John runs out of the house after he turns on the light. This indicates that the situation, in which John runs out of the house, is *after*, the situation in which Jimmy turns on the light. This is an example of an observation about the temporal relationship between situations.

To accommodate the different types of observations, the language $\mathcal{L}_O$ allows propositions of the following forms:

$$\varphi \textbf{ at } \mathsf{s} \tag{3.1}$$

$$\alpha \textbf{ between } \mathsf{s}_1, \mathsf{s}_2 \tag{3.2}$$

$$\alpha \textbf{ occurs\_at } \mathsf{s} \tag{3.3}$$

$$\mathsf{s}_1 \textbf{ preceeds } \mathsf{s}_2 \tag{3.4}$$

where φ is a fluent formula, α is a (possibly empty) sequence of actions, and $\mathsf{s}, \mathsf{s}_1, \mathsf{s}_2$ are situation constants which differ from s_c.

Observations of the forms (3.1) and (3.4) are called *fluent facts* and *precedence facts*, respectively. Observations of the forms (3.2) and (3.3) are referred to as *occurrence facts*. These two types of observations are different in that (3.2) states exactly what happened between two situations s_1 and s_2, whereas (3.3) only says what occurred in the situation s.

Example 20 The observations in the stohlen car story can be represented by the following set:

$$\Gamma_{car} = \left\{ \begin{array}{c} \mathit{has\_car} \textbf{ at } \mathsf{s}_0 \\ \neg \mathit{packed} \textbf{ at } \mathsf{s}_0 \\ \mathit{packed} \textbf{ at } \mathsf{s}_1 \\ \neg \mathit{has\_car} \textbf{ at } \mathsf{s}_2 \\ \mathsf{s}_0 \textbf{ preceeds } \mathsf{s}_1 \\ \mathsf{s}_1 \textbf{ preceeds } \mathsf{s}_2 \\ \mathsf{s}_2 \textbf{ preceeds } \mathsf{s}_C \end{array} \right\}$$

□

Observe that observations described in the previous chapters are different from the observations in this chapter. In $\mathcal{A}$ (or $\mathcal{B}$), the observations are about the current states. In this chapter, observations are about what happened in the past. For this reason, we use Γ (possibly with subscript) to denote a set of observations in this chapter.

3.2 $\mathcal{L}$ — A Language for Representing and Reasoning with Narratives

A narrative consists of the observations and a domain specification that an agent can use to explain her/his observations. The language $\mathcal{L}$ uses $\mathcal{B}$ and $\mathcal{L}_O$ for the specification of the domain specification and observations, respectively. The ontology of $\mathcal{L}$ consists of those of $\mathcal{B}$ and $\mathcal{L}_O$. We assume a single agent's world which evolves in a branching tree from an initial situation s_0. The branches of the tree are determined by the actions that can be performed in a situation – by either the agent or the environment – to transition the world to a new situation. Thus, each situation is simply an action history from s_0. It is also assumed that actions cannot happen simultaneously and no actions occur except those needed to explain the facts of the domain.

Formally, a *narrative* is a pair (D, Γ) where D is a domain specification and Γ is a set of observations of the form (3.1)-(3.4) such that $\{\mathsf{s}_0$ **preceeds** $\mathsf{s},\ \mathsf{s}$ **preceeds** $\mathsf{s}_c \mid \mathsf{s} \in \mathbf{S}\} \subseteq \Gamma$. In our examples, we will often omit the set of precedence facts related to s_0 and s_c when we specify Γ.

Example 21 The narrative representing the introductory example, $N_{car} = (D_{car}, \Gamma_{car})$, is given by the set of observations Γ_{car} (Example 20) and the following domain specification:

$$D_{car} = \left\{ \begin{array}{l} pack \textbf{ causes } packed \\ \quad \textbf{executable } pack \textbf{ if } at\_home \\ steal\_car \textbf{ causes } \neg has\_car \\ drive \textbf{ causes } at\_airport \\ \quad \textbf{executable } drive \textbf{ if } has\_car \\ at\_home \textbf{ s\_causes } \neg at\_airport \\ at\_airport \textbf{ s\_causes } \neg at\_home \end{array} \right\}$$

□

Observe that D_{car} is different from other domain specifications that we have discussed previously in the following sense. It contains one action, namely *steal_car*, whose occurrences is not controllable by the agent (John). This is an example of a *exogeneous actions*.

Intuitively, a narrative specifies possible histories of the world. As such any semantics of a narrative should present us with a specification of how the world evolves, from the initial situation to the current situation. The history needs to conform to the agent's observations and changes are caused only by actions. It should also allow us to draw conclusions that explain the observations. As an example, in the narrative (D_{car}, Γ_{car}), we should be able to conclude that

$$steal\_car \textbf{ at } \mathsf{s}_0 \quad \text{or} \quad steal\_car \textbf{ at } \mathsf{s}_1$$

since this are the only two possibilities that allow us to explain that the car disappear at s_2 (we assume that there is no other situation constants.) Furthermore, we should be able to conclude that

$$at\_airport \textbf{ at } \mathsf{s}_C$$

given the narrative

$$(D_{car} \cup \{rent\_car \textbf{ causes } has\_car\}, \Gamma_{car} \cup \{[rent\_car \circ drive] \textbf{ occurs\_at } \mathsf{s}_2\}).$$

(the symbol $\circ$ denotes the concatenation.)

To specify the semantics of a narrative, we use two functions, Σ and Ψ. Σ maps situation constants to action sequences and Ψ picks one among the various transitions given by $\Phi(a, s)$ and maps action sequences to a unique state with the condition that $\Psi(\alpha \circ a) \in \Phi(a, \Psi(\alpha))$. In essense, Σ assigns to each situation constant a history of the world (from the initial situation) and Ψ makes sure that the evolution of the world along a history is in accordance with the specification in the domain specification. We call Σ a *situation assignment* and Ψ a *causal model* of the narrative. Their precise definitions follow.

Definition 12 (Causal Interpretation) A *causal interpretation* Ψ of a domain specification D is a partial function from action sequences to interpretations of **F** such that

- its domain is nonempty, and
- if $\alpha \in Dom(\Psi)$ then $\beta \in Dom(\Psi)$ for every prefix β of α.

Notice that $[] \in Dom(\Psi)$ for every causal interpretation Ψ, where [] is the empty sequence of actions. Due to the second property, the function Ψ is said to be prefix-closed.

Definition 13 (Causal Model) A *causal model* of D is a causal interpretation Ψ such that:

(i) $\Psi([])$ is a state of D; and

(ii) for every $\alpha \circ a \in Dom(\Psi)$, $\Psi(\alpha \circ a) \in \Phi(a, \Psi(\alpha))$.

Situation assigment is defined next.

Definition 14 (Situation Assignment) A *situation assignment* of a set of situation constants **S** with respect to D is a mapping Σ from **S** into the set of action sequences of D that satisfy the following properties:

(i) $\Sigma(\mathsf{s}_0) = []$;

(ii) for every $\mathsf{s} \in \mathbf{S}$, $\Sigma(\mathsf{s})$ is a prefix of $\Sigma(\mathsf{s}_c)$.

We will now define the notion of an interpreation of a narrative.

Definition 15 (Interpretation) An *interpretation* M of a narrative (D, Γ) is a pair (Ψ, Σ), where Ψ is a causal model of D, Σ is a situation assignment of **S**, and $\Sigma(\mathsf{s}_c)$ belongs to the domain of Ψ.

Given an interpreation M, we need to specify the truth value of observations in M. For an interpretation $M = (\Psi, \Sigma)$ of (D, Γ):

(i) φ **at** s is true in M if φ holds in $\Psi(\Sigma(\mathsf{s}))$;

(ii) α **between** $\mathsf{s}_1, \mathsf{s}_2$ is true in M if $\Sigma(\mathsf{s}_1) \circ \alpha = \Sigma(\mathsf{s}_2)$;

(iii) α **occurs_at** s is true in M if the sequence $\Sigma(\mathsf{s}) \circ \alpha$ is a prefix of $\Sigma(\mathsf{s}_c)$;

(iv) s_1 **preceeds** s_2 is true in M if $\Sigma(\mathsf{s}_1)$ is a prefix of $\Sigma(\mathsf{s}_2)$.

When a fact o is true in M, we say that M satisfies o and write $M \models o$. We will next define the notion of a model of a narrative. Intuitively, a model of a narrative should allow us to explain the observations. Furthermore, as stated in the beginning of this section, the action occurrences in a model need to be minimal, i.e., only those needed to explain the narratives should be present. The notion of minimality is defined using the following definition.

Definition 16 (Compatible Causal Models) Let Ψ_0 and Ψ_1 be two causal models. We say that Ψ_1 is *compatilbe with* Ψ_0 if

- $\Psi_1([\,]) = \Psi_0([\,])$, and
- $\Psi_1(\alpha \circ a) = \Psi_0(\beta \circ a)$, for every pair of $\alpha \in Dom(\Psi_1)$ and $\beta \in Dom(\Psi_0)$ such that $\alpha \circ a \in Dom(\Psi_1)$, $\beta \circ a \in Dom(\Psi_0)$, and $\Psi_1(\alpha) = \Psi_0(\beta)$.

This leads to the following definition of models of narratives.

Definition 17 (Model of Narrative) An interpretation $M = (\Psi, \Sigma)$ is a *model* of a narrative (D, Γ) if:

(i) facts in Γ are true in M;

(ii) there is no other interpretation $M' = (\Psi', \Sigma')$ such that M' satisfies all the facts in Γ, Ψ' is compatible with Ψ, and $\Sigma'(\mathsf{s_c})$ is a subsequence of $\Sigma(\mathsf{s_c})$.

A narrative is *consistent* if it has a model. Otherwise, it is *inconsistent*.

The condition (ii) in the above definition requires that every action occurrence in the history of the world ($\Sigma(\mathsf{s_c})$) is neccessary for us to explain the observations.

The next examples illustrate the above defitions.

Example 22 Consider the narrative (D_{car}, Γ_{car}) (Examples 20-21). Let us derive a model (Ψ_0, Σ_0) for this narrative.

Because of the precedence facts, we know that

$$[\,] = \Sigma_0(\mathsf{s}_0) \sqsubseteq \Sigma_0(\mathsf{s}_1) \sqsubseteq \Sigma_0(\mathsf{s}_2) \sqsubseteq \Sigma_0(\mathsf{s_c})$$

where $\sqsubseteq$ denotes the prefix relationship between action sequences. The two observations

$$\neg \mathit{packed}\ \mathbf{at}\ \ \mathsf{s}_0 \qquad \mathit{packed}\ \mathbf{at}\ \ \mathsf{s}_1$$

indicates that an action with effect *packed* must have happened at s_0. With this in mind, we define

$$\Sigma_0(\mathsf{s}_1) = [\mathit{pack}]$$

Similarly, we can have

$$\Sigma_0(\mathsf{s}_2) = [\mathit{pack} \circ \mathit{steal\_car}]$$

Since there exists no observation between s_2 and $\mathsf{s_c}$, we can have

$$\Sigma_0(\mathsf{s}_c) = [\mathit{pack} \circ \mathit{steal\_car}]$$

The situation assignment is summarized in the following figure.

To complete the defintion of (Σ_0, Ψ_0), we now need to define Ψ_0. Observe that $Dom(\Psi_0)$ must contain $\Sigma_0(\mathsf{s_c})$ (Definition 15). Definition 12 implies that $Dom(\Psi_0)$ must also contain $[\mathit{pack}]$ and $[\,]$. In other words, we need to define

$$\Psi_0([\,]), \quad \Psi_0([\mathit{pack}]), \quad \Psi_0([\mathit{pack} \circ \mathit{steal\_car}])$$

Figure 3.1: A situation assignment for (D_{car}, Γ_{car})

It is easy to see that the domain specification D_{car} is deterministic. Therefore, the definition of Ψ_0 only depends on the situation assignment and the definition of $\Psi_0([\,])$. Definition 17 requires that $\Psi_0([\,])$ is a state satisfying the two observations has_car **at** s_0 and $\neg packed$ **at** s_0. This implies that

$$\Psi_0([\,]) = \{has\_car, \neg packed\} = s_0.$$

The definition of a causal model (Definition 13) leads to the following:

$$\Psi_0([pack]) \in \Phi(pack, s_0) = \{\{has\_car, packed\}\} \quad \text{which implies that} \quad \Psi_0([pack]) = \{has\_car, packed\}.$$

For similar reasons,

$$\Psi_0([pack \circ steal\_car]) = \{\neg has\_car, packed\}.$$

We will now show that $M = (\Psi_0, \Sigma_0)$ is indeed a model of (D_{car}, Γ_{car}). Clearly, Ψ_0 and Σ_0 satisfy the conditions in Definitions 13 and 14. Thus, M is an interpretation of the narrative.

It is easy to verify that the facts in Γ_{car} are satisfied in M. Furthermore, removing $pack$ from $\Sigma_0(\mathsf{s}_c)$ will not allow us to explain $packed$ **at** s_1 and removing $steal\_car$ from $\Sigma_0(\mathsf{s}_c)$ will not allow us to explain $\neg has\_car$ **at** s_2. Thus, M also satisfies the second condition of Definition 17, i.e., M is a model of (D_{car}, Γ_{car}).

Observe that the narratives have another model $M_1 = (\Psi_0, \Sigma_1)$ where

$$\Sigma_1(\mathsf{s}_0) = [], \quad \Sigma_1(\mathsf{s}_1) = \Sigma_1(\mathsf{s}_2) = \Sigma_1(\mathsf{s}_c) = [pack \circ steal\_car]$$

We conclude the example by presenting an interpretation $M_2 = (\Psi_2, \Sigma_2)$, which is not a model of (D_{car}, Γ_{car}). Let

$$\Sigma_2(\mathsf{s}_0) = [], \quad \Sigma_2(\mathsf{s}_1) = [pack], \quad \Sigma(\mathsf{s}_2) = [pack \circ steal\_car], \quad \Sigma(\mathsf{s}_c) = [pack \circ steal\_car \circ steal\_car]$$

where $\Psi_2([]) = \Psi_0([])$, $\Psi_2([pack]) = \Psi_0([pack])$, $\Psi_2([pack \circ steal\_car]) = \Psi_0([pack \circ steal\_car])$, and $\Psi_2(\Sigma_2(\mathsf{s}_c)) = \{\neg has\_car, packed\}$. The reason lies in that we can remove one of the occurences of $steal\_car$ and still obtain an interpretation satisfying all the observations. □

We will next consider another narrative that contains some occurence facts.

Example 23 Consider the story

> John goes to room SH124 to teahch his morning class. The light is off and he turns on the light as he was the first to enter the room. The light goes on and John finishes his lecture on time. He turns the light off and goes back to his office. Late afternoon, he goes to SH124 for his second class of the day. The light is off and he turns on the light again. This time, the ligth does not go on.

This story can be represented by a narrative with a domain consisting of three actions *turn_on, turn_off* and *burn_out(bulb)*. Intuitively, *burn_out*(*bulb*) is an exogenous action that makes the bulb defective. We denote the defectiveness of the bulb by the fluent *ab*(*bulb*). The information related to John's lamp in the story can then be described by the following narrative $N_{bulb} = (D_{bulb}, \Gamma_{bulb})$:

$$D_{bulb} = \left\{ \begin{array}{l} \mathit{turn\_on} \textbf{ causes } \mathit{light\_on} \textbf{ if } \neg \mathit{ab}(\mathit{bulb}), \neg \mathit{light\_on} \\ \mathit{turn\_off} \textbf{ causes } \neg \mathit{light\_on} \\ \mathit{ab}(\mathit{bulb}) \textbf{ s\_causes } \neg \mathit{light\_on} \\ \mathit{burn\_out}(\mathit{bulb}) \textbf{ causes } \mathit{ab}(\mathit{bulb}) \\ \textbf{executable } \mathit{burn\_out}(\mathit{bulb}) \textbf{ if } \neg \mathit{ab}(\mathit{bulb}) \end{array} \right\}$$

$$\Gamma_{bulb} = \left\{ \begin{array}{l} \mathit{turn\_on} \textbf{ between } \mathtt{s}_0, \mathtt{s}_1 \\ \mathit{turn\_off} \textbf{ occurs\_at } \mathtt{s}_1 \\ \mathit{turn\_on} \textbf{ between } \mathtt{s}_2, \mathtt{s}_3 \\ \mathtt{s}_1 \textbf{ preceeds } \mathtt{s}_2 \\ \mathtt{s}_2 \textbf{ preceeds } \mathtt{s}_3 \\ \neg \mathit{light\_on} \textbf{ at } \mathtt{s}_0 \\ \mathit{light\_on} \textbf{ at } \mathtt{s}_1 \\ \neg \mathit{light\_on} \textbf{ at } \mathtt{s}_2 \\ \neg \mathit{light\_on} \textbf{ at } \mathtt{s}_3 \end{array} \right\}$$

Observe that we do not list the precedence facts related to $\mathtt{s}_0$ and $\mathtt{s}_c$ in Γ_{bulb}. The set of observations can be viewed in the following (semi)-graphical representation:

Situation	●[$\mathtt{s}_0 \rightarrow \ldots$] [$\mathtt{s}_1 \rightarrow \ldots$] [$\mathtt{s}_2 \rightarrow \ldots$] [$\mathtt{s}_3 \rightarrow \ldots$] [$\mathtt{s}_c$]●
Fluent facts	$\neg$*light_on* *light_on* $\neg$*light_on* $\neg$*light_on*
Occurrence facts	|—*tunr_on*—|
	tunr_off
	|—*tunr_on*—|

Figure 3.2: John's Bulb Story

We will now determine the models of N_{bulb}. To simplify the represntation, we begin with the list the states of D_{bulb}. It is easy to see that D_{bulb} has only three states

$$\begin{aligned} s_0 &= \{\neg \mathit{light\_on}, \neg \mathit{ab}(\mathit{bulb})\} \\ s_1 &= \{\mathit{light\_on}, \neg \mathit{ab}(\mathit{bulb})\} \\ s_2 &= \{\neg \mathit{light\_on}, \mathit{ab}(\mathit{bulb})\} \end{aligned}$$

The fourth interpretation of the domain, $\{\mathit{light\_on}, \mathit{ab}(\mathit{bulb})\}$ is not a state since it does not satisfy the static causal axiom of the domain. The transition function of D_{bulb} is given by

$$\begin{array}{lll} \Phi(\mathit{turn\_on}, s_0) = \{s_1\} & \Phi(\mathit{turn\_on}, s_1) = \{s_1\} & \Phi(\mathit{turn\_on}, s_2) = \{s_2\} \\ \Phi(\mathit{turn\_off}, s_0) = \{s_0\} & \Phi(\mathit{turn\_off}, s_1) = \{s_0\} & \Phi(\mathit{turn\_off}, s_2) = \{s_2\} \\ \Phi(\mathit{burn\_out}(\mathit{bulb}), s_0) = \{s_2\} & \Phi(\mathit{burn\_out}(\mathit{bulb}), s_1) = \{s_2\} & \Phi(\mathit{burn\_out}(\mathit{bulb}), s_2) = \emptyset \end{array}$$

The transition function Φ is depicted in Figure 3.3.
We will now compute a model $M = (\Psi, \Sigma)$ for N_{bulb}. By definition, $\Sigma(\mathtt{s}_0) = [\ ]$.

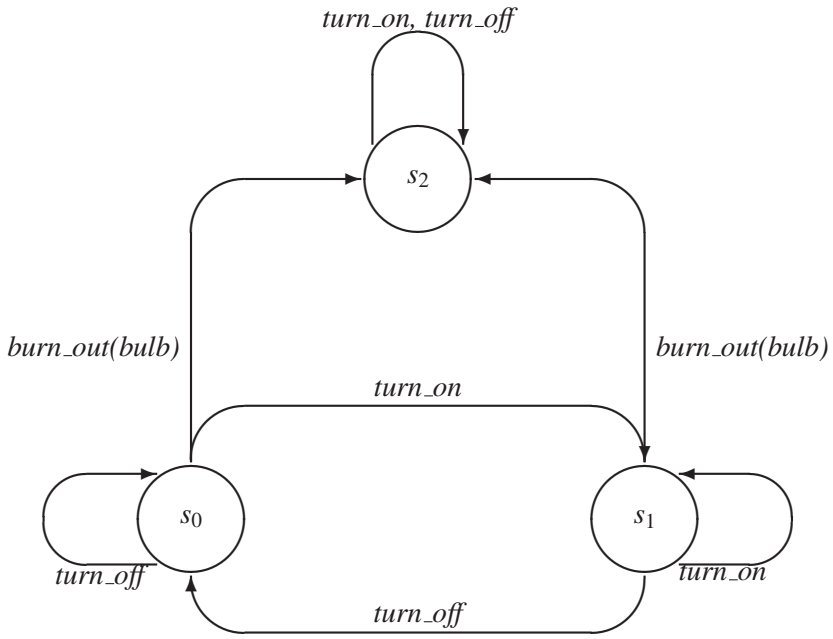

Figure 3.3: Transition Graph of D_{bulb}

Since *turn_on* **between** $\mathtt{s}_0, \mathtt{s}_1 \in \Gamma_{bulb}$, we have that $\Sigma(\mathtt{s}_1) = [turn\_on]$. The observation *light_on* **at** $\mathtt{s}_1$ allows us to conclude that $\Psi([\,]) = s_0$ and $\Psi(\Sigma(\mathtt{s}_1)) = s_1$.

Because *turn_off* **occurs_at** $\mathtt{s}_1 \in \Gamma_{bulb}$, we have $[turn\_on \circ turn\_off] \sqsubseteq \Sigma(\mathtt{s}_2)$. Furthermore, $\Sigma(\mathtt{s}_2) \circ turn\_on = \Sigma(\mathtt{s}_3) \sqsubseteq \Sigma(\mathtt{s}_c)$ due to *turn_on* **between** $\mathtt{s}_2, \mathtt{s}_3 \in \Gamma_{bulb}$. The definition of a causal model and the fact $\neg$*light_on* **at** $\mathtt{s}_3$ imply that $\Psi(\Sigma(\mathtt{s}_2))$ must be s_2. The only possibility for this to happen is to have

$$\Sigma(\mathtt{s}_2) = [turn\_on \circ turn\_off \circ burn\_bulb]$$

The minimality requirement implies that we can have $\Sigma(\mathtt{s}_3) = \Sigma(\mathtt{s}_c)$. In summary, the model of the narrative is

$\Sigma(\mathtt{s}_0) = []$,
$\Sigma(\mathtt{s}_1) = [turn\_on]$,
$\Sigma(\mathtt{s}_2) = [turn\_on \circ turn\_off \circ burn\_out(bulb)]$, and
$\Sigma(\mathtt{s}_3) = \Sigma(\mathtt{s}_c) = turn\_on \circ turn\_off \circ burn\_out(bulb) \circ turn\_on$ with
$\Psi([]) = s_0$
$\Psi(\sigma(\mathtt{s}_1)) = s_1$
$\Psi([turn\_on \circ turn\_off]) = s_0$,
$\Psi([turn\_on \circ turn\_off \circ burn\_out(buib)]) = s_2$, and
$\Psi(\Sigma(\mathtt{s}_3)) = s_2$. □

A narrative might have more than one model.

Example 24 Let us consider a modification of the N_{bulb} narrative. Assume that John runs out of the room after switching on the light bulb to receive a phone call. After finishing the call, he goes back to the room only to note that the light is not on.

The set of observations for this narrative differs from the previous one in that John did not observe the occurence of the action *turn_on* between s_2 and s_3. He only observed that *turn_on* occurs at s_2. Let us call the new set of observations Γ'_{bulb}, i.e.,

$$\Gamma'_{bulb} = \Gamma_{bulb} \setminus \{\mathit{turn\_on}\ \textbf{between}\ \ \mathsf{s}_2, \mathsf{s}_3\} \cup \{\mathit{turn\_on}\ \textbf{occurs\_at}\ \ \mathsf{s}_2\}$$

It is easy to check that the model of $(D_{bulb}, \Gamma_{bulb})$ is also a model of $(D_{bulb}, \Gamma'_{bulb})$. Besides, the new narrative will also admit the following models:

$M_1 = (\Psi_1, \Sigma_1)$ and $M_2 = (\Psi_2, \Sigma_2)$ where $\Psi_1([]) = \Psi_2([]) = s_0$, and

$\Sigma_1(\mathsf{s}_0) = []$,
$\Sigma_1(\mathsf{s}_1) = \mathit{turn\_on}$,
$\Sigma_1(\mathsf{s}_2) = \mathit{turn\_on} \circ \mathit{turn\_off}$, and
$\Sigma_1(\mathsf{s}_3) = \Sigma_1(\mathsf{s}_c) = \mathit{turn\_on} \circ \mathit{turn\_off} \circ \mathit{turn\_on} \circ \mathit{turn\_off}$,

$\Sigma_2(\mathsf{s}_0) = []$,
$\Sigma_2(\mathsf{s}_1) = \mathit{turn\_on}$,
$\Sigma_2(\mathsf{s}_2) = \mathit{turn\_on} \circ \mathit{turn\_off}$, and
$\Sigma_2(\mathsf{s}_3) = \Sigma_2(\mathsf{s}_c) = \mathit{turn\_on} \circ \mathit{turn\_off} \circ \mathit{turn\_on} \circ \mathit{burn\_out}(\mathit{bulb})$. □

3.3 $\mathcal{L}_Q$ — The Query Language of $\mathcal{L}$

We now define a query language $\mathcal{L}_Q$ for narratives. Queries in $\mathcal{L}_Q$ are of the following form:

$$\varphi\ \textbf{after}\ \alpha\ \textbf{at}\ \ \mathsf{s} \tag{3.5}$$

where $\mathsf{s} \in \mathbf{S}$, i.e., s could be the current situation s_c. We will use the following abbrevations for a query of the form (3.5):

- α **at** s if $\varphi = \top$.
- φ **at** s if $\alpha = [\,]$.
- **currently** φ if $\alpha = [\,]$ and $\mathsf{s} = \mathsf{s}_c$.

Definition 18 (Narrative Entailment) A query φ **after** α **at** s is *true* in a model $M = (\Psi, \Sigma)$ of a narrative (D, Γ), denoted by $(D, \Gamma) \models_M \varphi$ **after** α **at** s, if φ is true in $\widehat{\Phi}(\alpha, \Sigma(\mathsf{s}))$.

A query q is *entailed* by a narrative (D, Γ), denoted by $(D, \Gamma) \models q$, if q is true in every model of (D, Γ).

We can show that

$$(D_{car}, \Gamma_{car}) \models\ \textbf{currently}\ \neg \mathit{has\_car}$$

and

$$(D_{bulb}, \Gamma_{bulb}) \models\ \textbf{currently}\ \mathit{ab}(\mathit{bulb})$$

The first entailment says that John does not have a car. The second entailment says that the bulb of the room SH124 is defective.

Observe that (3.5) encodes two types of queries. The first type of queries asks about what *actually* happened and the second type of queries discusses *hypothetical* situations. For example, the query

$$\textbf{currently } ab(bulb)$$

asks whether the bulb is defective now? The query

$$[turn\_off \circ burn\_bulb] \textbf{ at } \mathtt{s}_1$$

asks whether the sequence $[turn\_off \circ burn\_bulb]$ happended at $\mathtt{s}_1$. These queries differ from the query

$$(D_{bulb}, \Gamma_{bulb}) \models ab(bulb) \textbf{ after } turn\_on \textbf{ at } \mathtt{s}_c$$

which asks whether the bulb will be defective after the action *turn_on* is executed *now*.

Queries of the form (3.5) differ from queries of the form (2.4) (Chapter 2) in that the latter are strictly about hypothetical statements.

3.4 Bibliographical Notes

The stohlen car example is motivated from the stohlen car example [**?**] and the going to the airport example from [**?**]. The propositional language $\mathcal{L}$ was developed in [**?**] to specify narratives and to reason with them. The language $\mathcal{L}$ defined in this chapter differs from the original language $\mathcal{L}$ in that it allows nondeterministic effects of actions. In this section we extend $\mathcal{L}$ with static causal axioms.

Chapter 4

Representing and Reasoning about Sensing Actions

All action languages from Chapter 2 and the language $\mathcal{L}$ developed in Chapter 3 discuss actions that change the world. In this chapter, we will discuss the representation of and reasoning with another type of actions, *sensing* actions (also referred to as *knowledge producing* actions in the literature). We will also formulate several notions such as *query*, *plan*, *entailment*, etc. for domain specifications with sensing actions. Furthermore, we will develop a theory of diagnosis for dynamic systems whose behavior is given by a domain specification with sensing actions.

Unlike actions that change the world, sensing or knowledge producing actions change what the agent knows about the world. For example, the action of looking at the traffic light does not change the status of the light being *green*, *yellow*, or *red*; but it changes the knowledge of an agent approaching the intersection. Sensory information acquired from sensing actions is critical for autonomous agents in the same way that information obtained through looking, smelling, touching, and hearing is for our daily activities. Well, just imagine how miserable our life would be without all these faculty!

Sensing actions also play an important role when an agent needs to plan in presence of incomplete information. Consider the case when an agent initially (i.e., in the initial situation) does not know whether the traffic light of a busy intersection is *red*, *yellow*, or *green* and his/her goal is to cross the intersection. We will assume that the agent can only perform the actions: *stop*, *wait*, and *drive*. Furthermore, the action *drive* cannot be executed if the light is *red*. For simplicity, we will assume that the traffic light changes its color in the order *red*, *green*, and *yellow* at every step. We now argue that the agent can not just construct a classical plan – consisting of a sequence of actions – that will always (i.e., regardless of what the real state of the world is) succeed in reaching the agent's goal.

Intuitively, to cross the intersection, our agent needs to execute the action *drive*. Let us first consider the plan consisting of the single action *drive*. This plan will not work if the light is initially *red* as the action *drive* cannot be executed if the light is *red*. Let us now consider the plan [$stop \circ drive$]. This plan will not work if the traffic light is initially *yellow*. Can the insertion of the action *wait* before *stop* help? Unfortunately, [$wait \circ stop \circ drive$] does not solve the problem if the light is initially *green*. In fact, we can easily show that no matter how long the agent waits and how many time he stops before he executes the action *drive*, there exists an iniitial situation in which he is attempting to cross the intersection when the light is *red*.

It is easy to see that a simple conditional plan

$$look \circ \textbf{cases}\begin{pmatrix} green & \rightarrow drive \\ yellow & \rightarrow stop \circ drive \\ red & \rightarrow stop \circ wait \circ drive \end{pmatrix}$$

will help our agent crosses the intersection.

If we were to follow the approach to representing and reasoning about actions and their effects, set forth by the previous chapters, the above story illustrates several points that need to be address in the presence of sensing actions:

- How to distinguish between the *real-state of the world* and the *mental state of an agent* in terms of what it knows and what it does not know?
- What is a 'state' that we should use in the transition function? Equivalently, how to represent the knowledge of the agent?
- How should the transition function be defined? Or, how does the agent's knowledge change after an action is perfomred?
- What is a plan?

In this chapter, we develop a language for representing and reasoning about sensing actions. We will start the section with a short discussion on reasoning about knowledge (Section 4.1). The syntax of the language $\mathcal{B}_K$, an extension of the language $\mathcal{B}$ to consider sensing actions, is discussed in Section 4.2 The semantics of $\mathcal{B}_K$ is presented in Section 4.3. Section 4.4 provises a glimpse on the complexity results related to reasoning and planning with $\mathcal{B}_K$ action theories. Section 4.5 presents a theory of approximation for action theories with incomplete information and/or static causal axioms. Section 4.6 considers narratives with $\mathcal{B}_K$ theory and presents a theory of diagnosis for systems with $\mathcal{B}_K$ domain specifications.

4.1 Reasoning with Knowledge — The Knowledge Operator

Philosophers use a modal operator $\mathbf{K}$ to express knowledge. For a fluent formula φ, $\mathbf{K}\,\varphi$ means that the agent knows that φ is true in the current state of the world. For example, John knows that the light is on is expressed as $\mathbf{K}\ light\_on$; John knows that his car has been stohlen is expressed as $\mathbf{K}\ \neg has\_car$; and John know that his car has been stohlen and the suitcase is packed is expressed as $\mathbf{K}\ packed \wedge \neg has\_car$. In presence of multiple agents multiple modal operators $\mathbf{K}_1$, $\mathbf{K}_2$, $\mathbf{K}_3$, ... are used. Let us consider the use of modal logic in a well-known puzzle, a variation of the "muddy children" or "three wise men" stoy.

Example 25 A certain king wishes to test his three wise men. He arranges them in a circle so that they can see and hear each other. None can see his own forehead though. The king then tells the three men that he will put a white or black spot on each of their foreheads but that at least one spot will be white. In fact all three spots are white. He then repeatedly asks them, "Do you know the color of your spot?" What do they answer? They answer the question with "*No*" the first two times and "*Yes*" thereafter.

Let us number the three wise men by 1, 2, and 3. For simplicity, we denote the color of the spot on each of them by $white_i$ (i.e., $\neg white_i$ denotes that the spot is black) for $i = 1, 2, 3$. The knowledge of each of them at the initial state is represented by the formulas

$$\neg\mathbf{K}_1\ white_1 \wedge \neg\mathbf{K}_1\ \neg white_1 \qquad \neg\mathbf{K}_2\ white_2 \wedge \neg\mathbf{K}_2\ \neg white_2 \qquad \neg\mathbf{K}_3\ white_3 \wedge \neg\mathbf{K}_3\ \neg white_3$$

The story follows with that all three men say "*No*" the first two times the answer was asked. They all say "*Yes*" after the third time the question was asked.

Indeed, given the knowledge that there is at least one white spot and everyone sees two white spots, none of the wise men can conclude that his spot is white or black.

After the first question was asked, everyone will realize that none of the others see two black spots. For otherwise, the person will know that his spot must be white. This information can be represented by

$$\mathbf{K}_1(\mathbf{K}_2\neg(\neg white_1 \wedge \neg white_3) \wedge \mathbf{K}_3\neg(\neg white_1 \wedge \neg white_2))$$

Equivalently, this is represented by

$$\mathbf{K}_1(\mathbf{K}_2(white_1 \vee white_3) \wedge \mathbf{K}_3(white_1 \vee white_3)$$

In other words, he knows that the second and third men did not see two black spots. The knowledge of other men can be represented accodingly.

After the second question was asked and the answer is "*No*" from all three of them, the first wise men will come to know that his spot is white. For otherwise, each of other wise men should have been able to deduce that their spot is white after the first question were asked. This information can be represented by an implication

$$\mathbf{K}_1\neg white_1 \rightarrow \mathbf{K}_1(\mathbf{K}_2 white_2 \wedge K_3 white_3)$$

The reasoning of the other two wise men is similar. As a result, they will all answer answer "*Yes*" when the question were asked the third time, and thereafter. □

The modal operator **K** can be introduced to any logic for modeling knowledge. We will now briefly discuss the syntax and semantics of the simplest modal logic, the propositional modal logic with a single modal operator **K**. Let P be a set of propositions. A formula is either a proposition in P or of the form $\mathbf{K}\,\varphi$, $\neg\varphi$, and $\varphi \wedge \psi$. The semantics of a propositional modal logic theory is defined using *Kripke structures*. A Krippe structure M is a triple (π, S, R), where S is the set of nonempty states or possible worlds, π is an interpretation which associates each state $s \in S$ a truth assignment $\pi(s)$ over the propositions in P and R is a binary relation over S, often referred to as an *accessibility relation*. The truth value of a formula φ with respect to a state s and a Krippe model M is defined inductively over the structure of φ as follows.

- For each proposition $p \in P$: $(M, s)) \models p$ if $\pi(s)(p)$ is true.
- $(M, s) \models \varphi \wedge \psi$ if $(M, s) \models \varphi$ and $(M, s) \models \psi$.
- $(M, s) \models \neg\varphi$ if $(M, s) \not\models \varphi$.
- $(M, s) \models \mathbf{K}\,\varphi$ if $(M, s') \models \varphi$ for all s', $(s, s') \in R$.

If R is reflexive, symmetric, and transitive then we have the modal logic **S5** which is axiomatized as follows:

- **K**: $\mathbf{K}\,\alpha \wedge \mathbf{K}\,\alpha \rightarrow \beta) \rightarrow \mathbf{K} : \beta$
- T: $\mathbf{K} : \alpha \rightarrow \alpha$
- 4: $\mathbf{K} : \alpha \rightarrow \mathbf{K}\,\mathbf{K}\,\alpha$
- 5: $\neg\mathbf{K}\,\alpha \rightarrow \mathbf{K}\,\neg\mathbf{K}\,\alpha$

4.2 Adding Sensing Actions to $\mathcal{B}$ — The Language $\mathcal{B}_K$

In this section we introduce $\mathcal{B}_K$—an extension of the language $\mathcal{B}$— which allows reasoning about sensing actions. To do so, we extend $\mathcal{B}$ with knowledge effect axioms of the following form:

$$a \textbf{ determines } \varphi \tag{4.1}$$

where a is a sensing action and φ is a set of mutual exclusive fluent literals. Intuitively, the above statement conveys the meaning that if a is executed, then in the resulting situation the truth value of $f \in \varphi$ becomes known. We will refer to actions that appear in fluent effect axioms as non-sensing actions. We will also assume that the set of sensing actions and the set of non-sensing actions are disjoint. For convenience, in our examples, we will often write φ as a list of fluent literals. The requirement that fluent literals in φ are mutual exclusive implies that

- for each pair of $l, l' \in \varphi$,

  $$l \textbf{ s\_causes } \neg l'$$

  is a static causal axiom, and

- for each literal $l \in \varphi$,

  $$\bigwedge_{l' \in \varphi \setminus \{l\}} \neg l' \textbf{ s\_causes } l$$

  is a static causal axiom.

Remark 4 The language $\mathcal{B}_K$ without static causal axioms and the restriction that each φ in (4.1) is of the form $\{f, \neg f\}$ for some fluent f is the language $\mathcal{A}_K$ in [SB01a].

Example 26 (From [SB01a]) Consider the example of a highly sensitive door with a lock. Pushing the door (*push_door*), when it is locked, will jam the door. The same action will open the door if it is not jammed and the lock is unlocked. Flipping the lock (*flip_lock*) will unlock a locked door and lock an unlocked door. On the other hand the action of checking the lock of the door (*check_if_locked*) will result in the agent knowing if the door is locked or not.

This domain can be represented by the following domain specification.

$$D_{door} = \left\{ \begin{array}{l} push\_door \textbf{ causes } opened \textbf{ if } \neg locked, \neg jammed \\ push\_door \textbf{ causes } jammed \textbf{ if } locked \\ flip\_lock \textbf{ causes } locked \textbf{ if } \neg locked \\ flip\_lock \textbf{ causes } \neg locked \textbf{ if } locked \\ check\_if\_locked \textbf{ determines } locked, \neg locked \\ \textbf{executable } push\_door \textbf{ if } \neg jammed \end{array} \right\}$$

The knowledge effect axiom in D_{door} represents the effects of the sensing action *check_if_locked*. □

An action theory in $\mathcal{B}_K$ is a pair (D, O) where D is a domain specification in $\mathcal{B}_K$ and O is a set of observations of the form (2.3).

4.3 Semantics of $\mathcal{B}_K$ — Knowledge States, Combined States, and Transitions

We will define the semantics of $\mathcal{B}_K$ in the same fashion that we used in defining the semantics of $\mathcal{B}$ or any of the other action languages. More precisely, we will formulate a transition function that charaterizes the change of the agent's knowledge as well as the world state after actions are performed. To do so, we need to represent the knowledge of the agent.

The discussion in the previous section shows that we can no longer identify the agent's knowledge with a state identical to the world state. In the three wise men story, the initial knowledge of each men could be any possible interpretation of the three propositions $white_1$, $white_2$, and $white_3$ but the one in which none of the spot is white ($\{\neg white_1, \neg white_2, \neg white_3\}$), i.e., each of the seven interpretations is a possible state of the world to each men.

If all an agent knows about the door in Example 26 is that the door is not jammed and not opened then the two states

$$\{\neg\, jammed, \neg opened, locked\} \qquad\qquad \{\neg\, jammed, \neg opened, \neg locked\}$$

are indistinguishable to him. In other words, each of the above states is a *possible state* that the agent thinks he might be in. This suggests us to represent the knowledge of an agent by a *set of states* that an agent thinks he might be in and refer to it as a *knowledge state* (or a k-state).

In the view of an impartial observer, the knowledge state of an agent and the real state of the world can be paired together to exactly describe the situation, in which the agent is. We will call such a pair as a *combined state* (or c-state). Thus, in $\mathcal{B}_K$, we have three kinds of states: real states of the world, knowledge states of the agent, and combined states.

In the following we will use small letters beginning from s (possibly with indexes) to denote world states, uppercase Greek letters like Σ (possibly with indexes) to denote k-states, and lowercase Greek letters like σ, δ (possibly with indexes) to denote c-states.

Formally, a *state* s is defined as in Definition 7. A *knowledge state* is a set of states. A *combined state* is a pair $\langle s, \Sigma\rangle$ where s is a state and Σ is a k-state. Intuitively, the state s in a c-state $\langle s, \Sigma\rangle$ is the real state of the world whereas Σ is the set of possible states which an agent believes it might be in. We say a c-state $\langle s, \Sigma\rangle$ is *grounded* if $s \in \Sigma$. Intuitively, grounded c-states correspond to the assumption that the world state belongs to the set of states that the agent believes it may be in.

The truth of a propositional fluent formula w.r.t. a state s is defined as in Chapter 2. We say two states s and s' *agree on a set of fluent literal* L if $s \cap L = s' \cap L$. Given a c-state $\langle s, \Sigma\rangle$, we say that a fluent formula φ is *known to be true* (resp. *known to be false*) in $\langle s, \Sigma\rangle$ if φ is true (resp. false) in every state $s' \in \Sigma$; and φ is *known* in $\langle s, \Sigma\rangle$, if φ is known to be true or known to be false in $\langle s, \Sigma\rangle$.

We will now extend the notion of a transision function for domain specifications in $\mathcal{B}$ to domain specifications in $\mathcal{B}_K$. Intuitively, a transition function for $\mathcal{B}_K$ theories should behave in the same way as for $\mathcal{B}$ theories if sensing actions are not present and the initial state is complete. We will need to consider different types of states in $\mathcal{B}_K$ domain specifications.

We already know how the real state of the world changes when a non-sensing action is performed (Definition 9). As sensing actions do not change the world, for a sensing action a, we must have that $\Phi(a, s) = \{s\}$ if a is executable in s and $\Phi(a, s) = \emptyset$ otherwise. This gives us the following requirement for the definition of a transition function Φ:

$$\Phi(a,s) = \begin{cases} \Phi(a,s) & \text{by Definition 9 if } a \text{ is a non-sensing action} \\ \{s\} & \text{if } a \text{ is a sensing action and executable in } s \\ \emptyset & \text{otherwise} \end{cases} \tag{4.2}$$

The extension of Φ over k-states is done in the same way that has been done before. For an action a and a knowledge state Σ

$$\Phi(a,\Sigma) = \bigcup_{s' \in \Sigma} \Phi(a,s')$$

Φ is extended over c-states as follows.

Definition 19 ($\mathcal{B}_K$-Transition Function) Let D be a $\mathcal{B}_K$ domain specification. A function Φ, which maps pairs of actions and c-states into c-states, is called a *transition function* of D if for a c-state $\langle s, \Sigma\rangle$ and action a,

- $\Phi(a, \langle s, \Sigma\rangle) = \emptyset$ if a is not executable in s;
- $\Phi(a, \langle s, \Sigma\rangle) = \{\langle s', \Phi(a, \Sigma)\rangle \mid s' \in \Phi(a, s)\}$ if a is executable in s and a is a non-sensing action;
- $\Phi(a, \langle s, \Sigma\rangle) = \langle s, \{s' \mid s' \in \Sigma$ such that s and s' agree on the literals from $\varphi\}\rangle$. if a is executable in s and a is a sensing action whose knowledge effect axiom is a **determines** φ.

Similar to Theorem 1, we have the following theorem.

Theorem 14 Every domain specification D possesses a unique transition function Φ.

We will use Φ_D to denote the transition function of D. We will often omit the subscription D whenever is is clear from the context what D is. Let us see now the transition function of some of the domain specifications that we have discussed so far.

Example 27 Consider the domain sspecification D_{door} from Example 26 and the c-state $\langle s_1, \{s_1, s_2\}\rangle$ where

$$s_1 = \{\neg jammed, \neg opened, locked\} \quad \text{and} \quad s_2 = \{\neg jammed, \neg opened, \neg locked\}.$$

We have the following:

- $\Phi_{D_{door}}(push\_door, \langle s_1, \{s_1, s_2\}\rangle) = \{\langle s_3, \{s_3, s_4\}\rangle\}$ where

  $$s_3 = \{jammed, \neg opened, locked\} \quad \text{and} \quad s_4 = \{\neg jammed, opened, \neg locked\}.$$

- $\Phi_{D_{door}}(flip\_lock, \langle s_1, \{s_1, s_2\}\rangle) = \{\langle s_2, \{s_1, s_2\}\rangle\}$.
- $\Phi_{D_{door}}(check\_if\_locked, \langle s_1, \{s_1, s_2\}\rangle) = \{\langle s_1, \{s_1\}\rangle\}$.

□

An important property of the transition function Φ is that it maintains the groundness of c-states.

Proposition 1 Let D be a consistent domain specification and a be an action that is executable in $\langle s, \Sigma\rangle$. If $\langle s, \Sigma\rangle$ is a grounded c-state then $\Phi(a, \langle s, \Sigma\rangle) \neq \emptyset$ and every $\langle s', \Sigma'\rangle \in \Phi(a, \langle s, \Sigma\rangle)$ is grounded also.

Proof. D is consistent implies that $\Phi(a, s) \neq \emptyset$ if a is executable in s. If a is a non-sensing action, then for every $\langle s', \Sigma' \rangle \in \Phi(a, \langle s, \Sigma \rangle)$ we have that $s' \in \Phi(a, s) \subseteq \Sigma'$ (from the groundness of $\langle s, \Sigma \rangle$). If a is sensing action, then $s \in \Sigma'$ by definition, since s agrees with itself on every set of fluent literals. □

We will now define the semantics of an $\mathcal{B}_K$ action theory (D, O). We begin with the definition of initial c-states.

Definition 20 ($\mathcal{B}_K$-Initial (c)-State) Let (D, O) be an $\mathcal{B}_K$ action theory.

- A state s is called an *initial state* of (D, O) if for every observation **initially** l in O, l is true in s.
- A c-state $\langle s_0, \Sigma_0 \rangle$ is an *initial c-state* of (D, O) if s_0 is an initial state and Σ_0 is a set of initial states of (D, O).

We say an initial c-state $\langle s_0, \Sigma_0 \rangle$ is *complete* if Σ_0 is the set of *all* initial states. Intuitively, the completeness of initial c-states express the assumption that our agent has complete knowledge about what it knows and does not know about the initial state. We will refer to this as the *complete awareness assumption*[1]. Even though, we believe that this assumption should not be used indiscriminately, *since it reduces the number of initial c-states, we will use it in most of our examples.*

Definition 21 ($\mathcal{B}_K$-Model) A *model* of an action theory (D, O) is a pair $(\langle s_0, \Sigma_0 \rangle, \Phi)$ such that $\langle s_0, \Sigma_0 \rangle$ is a grounded initial c-state of (D, O) and Φ is a transition function of D.

A model $(\langle s_0, \Sigma_0 \rangle, \Phi)$ is called *rational* if $\langle s_0, \Sigma_0 \rangle$ is complete.

Since the transition function Φ as defined so far can only tell us which c-state is reached after executing *an action* in a given c-state, we need to extend the function to be able to reason – beyond action sequences – about conditional plans.

Definition 22 (Conditional Plan) Let D be an $\mathcal{B}_K$ action theory. A *conditional plan* in D is defined as follows.

- The empty sequence of action [ ] is a conditiona plan.
- If a is a non-sensing action and p is a conditional plan then $[a \circ p]$ is a conditional plan.
- If a is a sensing action with a **determines** φ in D, where $\varphi = \{l_1, \ldots, l_m\}$, and p_j's are conditional plans then
$$[a \circ \textbf{cases}(\{l_i \rightarrow p_i\}_{j=1}^{n})]$$
is a conditional plan.
- Nothing else is a conditional plan.

By this definition, clearly a sequence of non-sensing actions is also a conditional plan. The execution of a conditional plan of the form $[a \circ p]$, where a is a non-sensing action and p is another conditional plan, is done sequentially, i.e., a is executed first, followed by the execution of p. To execute a conditional plan of the form $[a \circ \textbf{cases}(\{l_j \rightarrow p_j\}_{j=1}^{n})]$, we first execute a and then evaluate each l_j with respect to our current knowledge. If one of the l_j's, say l_k, is known to be true, we execute the corresponding sub-plan p_k. Observe that because fluent literals in φ are mutual exclusive, such l_k uniquely exists.

[1] Turner [Tur94] used a similar assumption called "complete initial situation assumption" according to which each model of his logic programming formulation of actions would have complete information about the initial state.

Remark 5 For simplicity of the presentation, Definition 22 does not consider a few types of conditional plans that may be of interested in certain cases. For example, a single sensing action is not a conditional plan. A case statement without a sensing action preceeding it is also not a conditional plan. This is not as severe as one might think since they can be replaced by conditional plan conforming to Definition 22 as follows.

- The sequence of a single sensing action a can be replaced by the plan

$$a \circ \textbf{ causes } (\{l_i \rightarrow [\,]\}_{i=1}^{n})$$

- A case statement

$$\textbf{cases}(\{l_i \rightarrow p_i\}_{i=1}^{n})$$

  where l_i's are mutual exclusive fluent literals can be replaced by the plan

$$\mathsf{a} \circ \textbf{cases}(\{l_i' \rightarrow p_i\}_{i=1}^{n})$$

  where a is a new sensing action that determines $\{l_1', \ldots, l_n'\}$ and for each i, l_i' **s_causes** l_i is a static causal axiom.

For this reason, we sometimes use the above plans as examples of conditional plans whenever it is convenient to do so.

We will next extend the transition function Φ for reasoning about the effects of conditional plans.

Definition 23 ($\mathcal{B}_K$-Extended Transition Function) Let D be a domain specification and Φ be a transition function of D. The *extended transition function* of D, denoted by $\widehat{\Phi}$, which maps a pair of a conditional plan c and a c-state $\langle s, \Sigma\rangle$ into a c-state, is defined as follows.

- For $c = [\,]$,

$$\widehat{\Phi}(c, \langle s, \Sigma\rangle) = \{\langle s, \Sigma\rangle\}$$

- For $c = [a \circ p]$, where a is a non-sensing action and p is a conditional plan,
  - if $\Phi(a, \langle s, \Sigma\rangle) \neq \emptyset$ and $\widehat{\Phi}(p, \langle s', \Sigma'\rangle) \neq \emptyset$ for every $\langle s', \Sigma'\rangle \in \Phi(a, \langle s, \Sigma\rangle)$ then

$$\widehat{\Phi}(c, \langle s, \Sigma\rangle) = \bigcup_{\langle s', \Sigma'\rangle \in \Phi(a, \langle s, \Sigma\rangle)} \widehat{\Phi}(p, \langle s', \Sigma'\rangle),$$

  - otherwise, $\widehat{\Phi}(c, \langle s, \Sigma\rangle) = \emptyset$.

- For $c = [a \circ \textbf{cases}(\{l_j \rightarrow c_j\}_{j=1}^{n})]$, where a is a sensing action and c_j's are conditional plans,
  - if $\Phi(a, \langle s, \Sigma\rangle) = \emptyset$ then $\widehat{\Phi}(c, \langle s, \Sigma\rangle) = \emptyset$,
  - if $\Phi(a, \langle s, \Sigma\rangle) \neq \emptyset$ then

$$\widehat{\Phi}(c, \langle s, \Sigma\rangle) = \bigcup_{1 \leq j \leq n, \langle s', \Sigma'\rangle \in \Phi(a, \langle s, \Sigma\rangle), l_j \text{is known to be true in } \langle s', \Sigma'\rangle} \widehat{\Phi}(p_j, \langle s', \Sigma\rangle)$$

We say that a conditional plan c is executable in a c-state $\langle s, \Sigma\rangle$ if $\hat{\Phi}(c, \langle s, \Sigma\rangle) \neq \emptyset$. The entailment relation for $\mathcal{B}_K$ action theories is defined next.

Definition 24 ($\mathcal{B}_K$-Entailment) Let D be a domain specification, c be a conditional plan, and φ be a fluent formula. We say,

- $D \models_{\mathcal{B}_K}$ **Knows** φ **after** c if c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ for every model (σ_0, Φ) of D;
- $D \models_{\mathcal{B}_K}$ **K**φ **after** α if c is executable in σ_0 and φ is known to be true or known to be false in $\hat{\Phi}(\alpha, \sigma_0)$ for every model (σ_0, Φ) of D.

Rational entailment of queries w.r.t. D – denoted by $\models^r_{\mathcal{B}_K}$ – is defined similarly by only considering rational models of D.

The following examples elucidates the above definitions.

Example 28 Let us consider the domain specification D_{door} from Example 26 and the set of observations

$$O_{door} = \{\textbf{initially } \neg jammed, \textbf{initially } \neg opened\}.$$

The states of D_{door} are:

$$\begin{array}{ll} s_1 = \{\neg jammed, \neg locked, \neg opened\} & s_5 = \{jammed, \neg locked, \neg opened\} \\ s_2 = \{\neg jammed, locked, \neg opened\} & s_6 = \{jammed, locked, \neg opened\} \\ s_3 = \{\neg jammed, \neg locked, opened\} & s_7 = \{jammed, \neg locked, opened\} \\ s_4 = \{\neg jammed, locked, opened\} & s_8 = \{jammed, locked, opened\} \end{array}$$

The set of initial states of (D_{door}, O_{door}) is $\Sigma_0 = \{s_1, s_2\}$ and its two complete initial c-states are $\sigma_1 = \langle s_1, \Sigma_0\rangle$ and $\sigma_2 = \langle s_2, \Sigma_0\rangle$. Let Φ be the transition function of D_{door}. By Definition 21, (D_{door}, O_{door}) has two rational models: (σ_1, Φ) and (σ_2, Φ). We have:

$$\begin{array}{l} \widehat{\Phi}([check\_if\_locked], \sigma_1) = \{\langle s_1, \{s_1\}\rangle\} \\ \widehat{\Phi}([check\_if\_locked], \sigma_2) = \{\langle s_2, \{s_2\}\rangle\} \\ \widehat{\Phi}([check\_if\_locked \circ push\_door], \sigma_1) = \{\langle s_3, \{s_3\}\rangle\} \\ \widehat{\Phi}([check\_if\_locked \circ push\_door], \sigma_2) = \{\langle s_6, \{s_6\}\rangle\} \\ \widehat{\Phi}([check\_if\_locked \circ flip\_lock], \sigma_1) = \{\langle s_2, \{s_2\}\rangle\} \\ \widehat{\Phi}([check\_if\_locked \circ flip\_lock], \sigma_2) = \{\langle s_1, \{s_1\}\rangle\} \\ \widehat{\Phi}([check\_if\_locked \circ flip\_lock \circ push\_door], \sigma_1) = \{\langle s_6, \{s_6\}\rangle\} \\ \widehat{\Phi}([check\_if\_locked \circ flip\_lock \circ push\_door], \sigma_2) = \{\langle s_3, \{s_3\}\rangle\} \end{array} \tag{4.3}$$

Based on the above computation we have the following:

$$(D_{door}, O_{door}) \not\models^r_{\mathcal{B}_K} \textbf{Knows } opened \wedge \neg jammed \textbf{ after } [check\_if\_lock \circ push\_door]$$

and

$$(D_{door}, O_{door}) \not\models^r_{\mathcal{B}_K} \textbf{Knows } opened \wedge \neg jammed \textbf{ after } [check\_if\_lock \circ flip\_lock \circ push\_door].$$

It can be shown that there exists no sequence of actions α of D_{door} such that

$$(D_{door}, O_{door}) \models^{r}_{\mathcal{B}_K} \textbf{Knows}\ opened \wedge \neg jammed\ \textbf{after}\ \alpha.$$

Let us now consider the conditional plan:

$$p = check\_if\_locked \circ \textbf{cases}\begin{pmatrix} locked & \rightarrow & flip\_lock \circ push\_door \\ \neg locked & \rightarrow & push\_door \end{pmatrix}$$

We will show that $(D_{door}, O_{door}) \models^{r}_{\mathcal{B}_K} \textbf{Knows}\ opened\ \textbf{after}\ p$.

From the definition of $\widehat{\Phi}$ and the computation of $\widehat{\Phi}$ in (4.3), we have the following computation.

- Because $\neg locked$ is known to be true in $\Phi(check\_if\_locked, \sigma_1)$, we have

$$\widehat{\Phi}(p, \sigma_1) = \widehat{\Phi}(push\_door, \langle s_1, \{s_1\}\rangle)) = \{\langle s_3, \{s_3\}\rangle\}$$

- Because $locked$ is known to be true in $\Phi(check\_if\_locked, \sigma_2)$, we have that

$$\widehat{\Phi}(p, \sigma_2) = \widehat{\Phi}(flip\_lock \circ push\_door, \langle s_2, \{s_2\}\rangle) = \widehat{\Phi}(push\_door, \langle s_1, \{s_1\}\rangle) = \{\langle s_3, \{s_3\}\rangle\}$$

Since $opened \wedge \neg jammed$ is known to be true in the c-state $\langle s_3, \{s_3\}\rangle$, by Definition 24,

$$(D_{door}, O_{door}) \models^{r}_{\mathcal{B}_K} \textbf{Knows}\ opened \wedge \neg jammed\ \textbf{after}\ p.$$

□

The notion of a trajectory (Definition 10) to define the notion of a trajectory of conditional plans.

Definition 25 (Trajectory of Conditional Plan) Let c be a plan and $\langle s, \Sigma\rangle$ be a c-state. A *trajectory* of c wrt. $\langle s, \Sigma\rangle$ is a sequence of c-states $\langle s_i, \Sigma_i\rangle_{i=1}^{k}$ if

- $c = [\,]$, $k = 1$, and $\langle s, \Sigma\rangle = \langle s_1.\Sigma_1\rangle$; or
- $c = [a \circ p]$ where a is a non-sensing action, $\langle s_1, \Sigma_1\rangle \in \Phi(a, \langle s, \Sigma\rangle)$, and $\langle s_i, \Sigma_i\rangle_{i=2}^{k}$ is a trajectory of p wrt. $\langle s_1, \Sigma_1\rangle$; or
- $c = [a \circ \textbf{cases}(\{l_i \rightarrow p_i\}_{i=1}^{n})$ where a is a sensing action, $\langle s_1, \Sigma_1\rangle \in \Phi(a, \langle s, \Sigma\rangle)$, l_j is known to be true in $\langle s_1, \Sigma_1\rangle$, and $\langle s_i, \Sigma_i\rangle_{i=2}^{k}$ is a trajectory of p_j wrt. $\langle s_1, \Sigma_1\rangle$.

4.4 Complexity of Reasoning and Planning in $\mathcal{B}_K$

Since the execution of a sensing action does not change the state of the world and its executability condition can be checked in polynomial time, many complexity results in Chapter 2 (Section 2.7) remain valid for $\mathcal{B}_K$. In particular, we can show that

- $\text{CI}^{\mathcal{B}_K}$ is in P.
- $\text{CS}^{\mathcal{B}_K}$ is Π_3^P-complete.

- $\mathrm{DT}^{\mathcal{B}_K}$ is coNP-complete.
- $\mathrm{AC}^{\mathcal{B}_K}$ is NP-complete.

The planning problem in $\mathcal{B}_K$ is different than the planning problem in $\mathcal{B}$ as a conditional plan might contain branching statements. We will define the size of a conditional plan (or plan) c, denoted by $size(c)$, as follows:

1. $size([]) = 0$;
2. $size([a \circ c]) = 1 + size(c)$ if a is a non-sensing action and c is a plan; and
3. $size([a \circ \mathbf{cases}(\{g_j \rightarrow c_j\}_{j=1}^n)]) = 1 + \Sigma_{j=1}^n(1 + size(c_j))$ if a is a sensing action and c_j's are plans.

The planning problem is then defined as follows:

- $\mathrm{PI}^{\mathcal{B}_K}$ (*Planning with incomplete initial state*) Given a consistent domain specification D, a set of observations about the initial state O, and a fluent literal l, a polynomial function $f : N \rightarrow N$, determining a plan c, whose size is bounded by $f(|D|)$, such that $(D, O) \models l$ **after** c.

The complexity of the planning problem in this setting is given in the next theorem.

Theorem 15 $\mathrm{PI}^{\mathcal{B}_K}$ is PSPACE complete.

Specifically, if the size of the plan is fixed, we have the following theorem.

Theorem 16 $\mathrm{PI}^{\mathcal{B}_K}$ is Π_{2k+1}^P-complete if the size of the plan is bounded by k.

4.5 Approximation Reasoning

The complexity results in Chapter 2 (Section 2.7) and in the previous section show that reasoning and planning with incomplete information and actions with static causal axioms is computationally harder than with complete information and deterministic actions. In this section, we will develop a new method to reasoning in the presence of incomplete information and static causal axioms, called *approximation reasoning*, which provides some remedies in this regards in certain situation.

The main source of the computational complexity in the presence of incomplete informaiton lies in the number states belonging to the belief state of the agent when reasoning about the effects of his/her actions. An obvious way to deal with this high number is to approximate the agent's knowledge about the world by the intersection of all the states belonging to the agent's belief state.

Example 29 Let D be the domain specification with the following fluent effect axioms

$$a \textbf{ causes } f$$
$$b \textbf{ causes } g \textbf{ if } f$$

Suppose that our agent does not know whether f or g is true or false initially. Suppose also that the goal of the agent is to make g to be true.

It is easy to see that the plan $[a \circ b]$ will help the agent to achieve his goal. The following computation proves this fact.

The domain has four possible states: $s_0 = \{\neg f, \neg g\}$, $s_1 = \{\neg f, g\}$, $s_2 = \{f, \neg g\}$, and $s_3 = \{f, g\}$.

The initial belief state of our agent is $\Sigma_0 = \{s_0, s_1, s_2, s_3\}$. Hence, there are four possible initial c-states $\langle s_i, \Sigma_0\rangle$ for $i = 0, 1, 2, 3$. Furthermore, $\widehat{\Phi}([a \circ b], \langle s_i, \Sigma_0\rangle) = \{\langle s_3, \{s_3\}\rangle\}$ for every i. This shows that $(D, \emptyset) \models^r_{\mathcal{B}_K} g$ **after** $[a \circ b]$. Following this reasoning, we need to consider 4 initial c-states.

On the other hand, we can observe that if we were to represent the initial knowledge of the agent by $\emptyset$, his knowledge after the execution of the action a will be $\{f\}$ as f is a direct effect of a. This will be sufficient for the agent to conclude that g is true after the execution of b, followed by a. This reasoning process does not do reasoning by cases. □

The second source of complexity in reasoning about effects of actions and planning is the presence of static causal axioms. The main reason for this complexity lies in that they also introduce nondeterminism into the reasoning process, forcing the agent to consider multiple outcomes after the execution of an axtion (e.g., Examples 15 and 16). Although this type of nondeterminism is different from the nondeterminism caused by the lack of knowledge about the initial situation, the idea explored in Example 29 could also be applied to deal with it.

Example 30 Consider a variation of the domain specification from Example 15

$$D = \left\{ \begin{array}{l} f, \neg g \textbf{ s\_causes } h \\ f, \neg h \textbf{ s\_causes } g \\ a \textbf{ causes } f \\ b \textbf{ causes } g \textbf{ if } f \end{array} \right\}$$

Suppose that our agent knows that $s_0 = \{\neg f, \neg g, \neg h\}$ and his/her goal is to make g true. It is easy to see that the plan $[a \circ b]$ will help our agent to achieve her goal. The validation of this plan is shown below:

$$\Phi(a, s_0) = \{s_1, s_2\} \text{ where } s_1 = \{f, h, \neg g\},\ s_2 = \{f, \neg h, g\}$$

and

$$\Phi(b, s_1) = \{f, h, g\} \text{ and } \Phi(b, s_2) = \{f, \neg h, g\}$$

The reasoning process leads our agent through two states, s_1 and s_2. Subsequently, the result of the execution of b in these two states needs to be computed.

On the other hand, it is known that f must be true after the execution of a as it is a direct effect of a. This information is sufficent for our agent to conclude that g will be true after the execution of the sequence $[a \circ b]$. □

4.5.1 Φ^0 — An Approximation of Φ

In this subsection, we define an approximation semantics of $\mathcal{B}_K$. It is defined by a transition function Φ^0 that maps actions and approximation states, an approximation of the knowledge of the reasoning agent, to approximation states. Before providing the formal definition of the transition function, we introduce some notations and terminology. First, we relax the notion of a state in Definition 7 to be an approximate state defined as follows.

Definition 26 (Approximate State) A consistent set of literals δ is called an approximate state (or *a-state*, for short) if δ satisfies all static causal axioms in D.

Intuitively, δ represents the (possibly incomplete) current knowledge of the agent, i.e., it contains all fluent literals that are known to be true to the agent. When δ is a subset of some state s, we say that it is *valid*. For example, in a domain with two fluents f and g, the fact that the agent knows that f is true and does not know the truth value of g can be represented by the approximation state $\{f\}$.

Given an a-state δ and a literal l, we say that l is true (or l holds) in δ if $l \in \delta$. l is false in δ if $\bar{l} \in \delta$. Otherwise, l is unknown in δ. A conjunction of literals φ is true in δ if each of its conjuncts is true in δ, i.e., if $\varphi \subseteq \delta$. A fluent literal l possibly holds in δ if it is not false in δ. A conjunction of literals φ is true in δ if each of its conjuncts is not false in δ. An action a is *executable* in δ if there exists an executability axiom (2.2) in D such that φ holds in δ.

Next, we define what are the possible next a-states after the execution of an action a in a given a-state δ, provided that a is executable in δ. Consider the case that a is a non-sensing action. Let

$$e(a, \delta) = Cl_D(\{l \mid \exists\, a \textbf{ causes } l \textbf{ if } \varphi \text{ in } D \text{ such that } \varphi \text{ holds in } \delta\}) \tag{4.4}$$

and

$$pc(a, \delta) = \bigcup_{i=0}^{\infty} pc^i(a, \delta) \tag{4.5}$$

where

$$pc^0(a, \delta) = \{l \mid \exists \textbf{ causes } l \textbf{ if } \varphi \text{ in } D \text{ such that } l \notin \delta \text{ and } \varphi \text{ possibly holds in } \delta\} \tag{4.6}$$

and for $i \geq 0$,

$$pc^{i+1}(a, \delta) = pc^i(a, \delta) \cup \left\{ l \;\middle|\; \begin{array}{l} \exists\, \varphi \textbf{ s\_causes } l \text{ in } D \text{ such that } l \notin \delta, \varphi \cap pc^i(a, \delta) \neq \emptyset, \\ \text{and } \varphi \text{ possibly holds in } e(a, \delta) \end{array} \right\}$$

Intuitively, $e(a, \delta)$ and $pc(a, \delta)$ denote what *definitely holds* and what *may change* in the next situation respectively [2]. Specifically, $l \in e(a, \delta)$ means that l holds in the next situation and $l \in pc(a, \delta)$ means that l is not in δ but possibly holds in the next situation. This implies that $\delta \setminus \neg pc(a, \delta)$ is an approximation of the set of literals that hold by inertia after the execution of a in δ. Taking into account the effects of the static causal axioms, we have that the set of literals $\delta' = Cl_D(e(a, \delta) \cup (\delta \setminus \neg pc(a, \delta)))$ must hold in the next situation. This is proved in the nex lemma.

Lemma 5 *Let D be a domain sprecification and Φ be its transition function. For every state $s' \in \Phi(a, s)$, we have*

$$s' \setminus (e(a, s) \cup (s \cap s')) \subseteq pc(a, \delta)$$

Proof. Let σ denote $e(a, s) \cup (s \cap s')$. By Corollary 1, since $e(a, \delta) \subseteq e(a, s) \subseteq \sigma$, we have

$$Cl_D(e(a, \delta)) \subseteq Cl_D(\sigma) = s' \tag{4.7}$$

We now show that, for every $i \geq 1$,

$$\Gamma^i(\sigma) \setminus \Gamma^{i-1}(\sigma) \subseteq pc^i(a, \delta) \tag{4.8}$$

by induction on i.

[2] Note that the operator Cl_D is used in the definition of $e(a, \delta)$ to *maximize* what definitely holds in the next situation.

1. **Base case:** $i = 1$. Let l be a literal in $\Gamma^1(\sigma) \setminus \Gamma^0(\sigma)$. We need to prove that $l \in pc^1(a, \delta)$.

   By the definition of Γ, it follows that

   $$l \notin \Gamma^0(\sigma) = \sigma \tag{4.9}$$
   $$l \in \Gamma^1(\sigma) \subseteq s' \tag{4.10}$$

   and, in addition, there exists a static causal axiom

   $$\varphi \textbf{ s\_causes } l$$

   in D such that

   $$\varphi \subseteq \Gamma^0(\sigma) = \sigma \tag{4.11}$$

   By (4.9), we have $l \notin (s \cap s')$. By (4.10), we have $l \in s'$. Accordingly, we have $l \notin s$. On the other hand, because $\delta \subseteq s$, we have

   $$l \notin \delta \tag{4.12}$$

   It follows from (4.11) that $\varphi \subseteq s'$ since $\sigma \subseteq s'$. Because of the completeness of s', we have $\overline{\varphi} \cap s' = \emptyset$. On the other hand, by (4.7), we have $Cl_D(e(a, \delta)) \subseteq s'$. As a result, we have

   $$\overline{\varphi} \cap Cl_D(e(a, \delta)) = \emptyset \tag{4.13}$$

   We now show that $\varphi \not\subseteq s$. Suppose otherwise, that is, $\varphi \subseteq s$. This implies that $l \in s$. By (4.10), it follows that $l \in (s \cap s') \subseteq \sigma$ and this is a contradiction to (4.9). Thus, $\varphi \not\subseteq s$.

   On the other hand, we know that $\varphi \subseteq \sigma = e(a, s) \cup (s \cap s')$ and thus we have $\varphi \cap (e(a, s) \setminus s) \neq \emptyset$. In addition, it is easy to see that $e(a, s) \setminus s \subseteq e(a, s) \setminus \delta \subseteq pc^0(a, \delta)$. Therefore, we have

   $$\varphi \cap pc^0(a, \delta) \neq \emptyset \tag{4.14}$$

   From (4.12) – (4.14), and by the definition of $pc^1(a, \delta)$, we can conclude that $l \in pc^1(a, \delta)$. The base case is thus true.

2. **Inductive Step:** Assume that (4.8) is true for all $i \leq k$. We need to prove that it is true for $i = k + 1$. Let l be a literal in $\Gamma^{k+1}(\sigma) \setminus \Gamma^k(\sigma)$. We will show that $l \in pc^{k+1}(a, \delta)$.

   By the definition of Γ, there exists a static causal axiom

   $$\varphi \textbf{ s\_causes } l$$

   in D such that

   $$\varphi \subseteq \Gamma^k(\sigma) \subseteq s' \tag{4.15}$$

   Because $\varphi \subset s'$, we have $\overline{\varphi} \cap s' = \emptyset$. In addition, by (4.7), $Cl_D(e(a, \delta))$ is a subset of s'. A a result, we have

   $$\overline{\varphi} \cap Cl_D(e(a, \delta)) = \emptyset \tag{4.16}$$

It is easy to see that $\varphi \not\subseteq \Gamma^{k-1}(\sigma)$ for if otherwise then, by the definition of Γ, l must be in $\Gamma^k(\sigma)$, which is impossible. In other words, there exists $l' \in \varphi$ such that $l' \notin \Gamma^{k-1}(\sigma)$ but $l' \in \Gamma^k(\sigma)$. By the inductive hypothesis, we have $l' \in pc^k(a, \delta)$, which implies that

$$\varphi \cap pc^k(a, \delta) \neq \emptyset \tag{4.17}$$

Because $l \notin \Gamma^k(\sigma)$, we have $l \notin \sigma$. As a result, $l \notin (s \cap s')$. On the other hand, since $l \in \Gamma^{k+1}(\sigma) \subseteq s'$, it follows that $l \notin s$. Thus, we have

$$l \notin \delta \tag{4.18}$$

From (4.16) – (4.18), and by the definition of $pc^{k+1}(a, \delta)$, it follows that $l \in pc^{k+1}(a, \delta)$. So the inductive step is proven.

As a result, it is always the case that (4.8) holds. Hence, we have

$$\Gamma^i(\sigma) \setminus \sigma \subseteq \bigcup_{j=0}^{i}(pc^j(a, \delta)) = pc^i(a, \delta)$$

and thus,

$$\bigcup_{i=0}^{\infty}(\Gamma^i(\sigma) \setminus \sigma) \subseteq \bigcup_{i=0}^{\infty} pc^i(a, \delta)$$

Accordingly, by Lemma 1 and by the definition of $pc(a, \delta)$, we have

$$(s' \setminus \sigma) \subseteq pc(a, \delta).$$

The lemma is thus true. □

The above lemma leads us to the following definition of the possible next a-states after a non-sensing action gets executed. The transition function Φ^0 that maps actions and a-states into sets of a-states is defined as follows.

Definition 27 (Approximation Transition Function) Given a domain specification D, for any action a and a-state δ,

- if a is not executable in δ then
$$\Phi^0(a, \delta) = \perp$$
where $\perp$ is a symbol denoting that the execution of a in δ fails.

- if a is a non-sensing action that is executable in δ
$$\Phi^0(a, \delta) = \begin{cases} \{Cl_D(e(a, \delta) \cup (\delta \setminus \neg pc(a, \delta)))\} & \text{if } Cl_D(e(a, \delta) \cup (\delta \setminus \neg pc(a, \delta))) \text{ is consistent} \\ \emptyset & \text{otherwise} \end{cases}$$

- if a is a sensing action with the knowledge effect axiom a **determines** $\{l_1, \ldots, l_m\}$ and a is executable in δ then
$$\Phi^0(a, \delta) = \{Cl_D(\delta \cup \{l_i\}) \mid Cl_D(\delta \cup \{l_i\}) \text{ is consistent}\}$$

A studious reader will notice a slight change in the formation of the transition function Φ^0 where the symbol $\perp$ is introduced to denote that the action execution fails. In this situation, the agent *knows* that the execution fails. On the other hand, $\Phi_0(a, \delta) = \emptyset$ indicates two possibilities, either δ is an invalid a-state or a is not executable in δ. As we will see later, the second case cannot happen if our agent starts the reasoning process from a valid a-state. The third item of the definition makes sure that if a is a sensing action that determines $l_1, \ldots, l_m$ then its execution allows the agent to know the truth value of each of them. This will also allow the agent to eliminate the states of the world that are no longer consistent with his belief prior to the execution of actions. The following example illustrates this.

Example 31 Consider the domain specification D_{light} for the traffic light example:

$$D_{light} = \left\{ \begin{array}{l} look \textbf{ determines } \{green, red, yellow\} \\ green \oplus red \oplus yellow \end{array} \right\}$$

Let $\delta = \{\neg red\}$. We have

$$Cl_{D_{light}}(\delta \cup \{yellow\}) = \{\neg red, yellow, \neg green\} = \delta_1$$

$$Cl_{D_{light}}(\delta \cup \{red\}) = \{\neg red, red, \neg green, \neg yellow, green, yellow\} = \delta_2$$

$$Cl_{D_{light}}(\delta \cup \{green\}) = \{\neg red, green, \neg yellow\} = \delta_3$$

Among those, δ_2 is inconsistent. Therefore, we have

$$\Phi^0_{D_{light}}(look, \delta_1) = \{\delta_1, \delta_3\}.$$

Given that the agent knows that the light is not red, it is not sufficient for him/her to conclude that the light is green or the light is yellow. □

The above situation could arise in reasoning about the effects of a sequence of actions.

Example 32 Consider the following domain specification

$$D = \left\{ \begin{array}{l} b \textbf{ causes } h \\ g, h \textbf{ s\_causes } f \\ g, \neg h \textbf{ s\_causes } f \\ a \textbf{ determines } f, \neg f \end{array} \right\}$$

Let $\delta = \{g\}$. It is easy to see that the execution of a in δ will result into two a-states, $\delta_1 = \{g, \neg f\}$ and $\delta_2 = \{g, f\}$. Intuitively, only one of these two a-states will be the "correct" one as the world state does not change after the execution of a sensing action. Since our agent does not have complete knowledge about the world, δ_1 and δ_2 will be his/her knowledge about the world after the execution of a. Suppose now that the agent executes b. Intuitively, h must be true. However, adding h to δ_1 yields $\{g, \neg f, h\}$. The static causal axiom gives $\{g, \neg f, h, f\}$, i.e., an inconsistent a-state. This means that our agent now has the reason for removing δ_1 from the set of possible a-states that he/she needs to consider after the execution of a. □

Examples 31 and 32 raise a serious question on the soundness of Φ^0. Could it be possible that the use of Φ^0 leads to a wrong answer? In other words, an agent using the approximation will answer 'yes' to a query that would be answered with 'no' if the full semantics? We will next show that if the knowledge of the agent is consistent with the world state then this cannot happen. We need the following lemma.

Lemma 6 Let D be a domain specification, a be an action, and s, s' be states. Then, we have

$$Cl_D(e(a,s) \cup (s \cap s')) = Cl_D(Cl_D(e(a,s)) \cup (s \cap s')).$$

Proof. Let $E(a,s) = Cl_D(e(a,s))$. Let $\gamma = e(a,s) \cup (s \cap s')$ and $\gamma' = E(a,s) \cup (s \cap s')$. As $\gamma \subseteq \gamma'$, it follows from Corollary 1 that to prove this lemma, it suffices to prove that

$$Cl_D(\gamma') \subseteq Cl_D(\gamma)$$

It is easy to see that

$$\gamma' = Cl_D(E(a,s)) \cup (s \cap s') \subseteq Cl_D(E(a,s) \cup (s \cap s')) = Cl_D(\gamma)$$

Therefore, by Corollary 1, we have

$$Cl_D(\gamma') \subseteq Cl_D(Cl_D(\gamma)) = Cl_D(\gamma)$$

□

The next proposition shows that the function Φ_D^0 indeed approximates Φ_D.

Proposition 2 Let D be a consistent domain specification, a be an action executable in an a-state δ, and s be a state such that $\delta \subseteq s$.

- If a is a non-sensing action then there exists an a-state δ' such that $\Phi_D^0(a,\delta) = \{\delta'\}$ and δ' is a subset of every state $s' \in \Phi_D(a,s)$.
- If a is a sensing action then $\Phi_D^0(a,\delta) \neq \emptyset$.

Proof.

- Let

$$\gamma = e(a,\delta) \cup (\delta \setminus \overline{pc(a,\delta)}) \qquad \delta' = Cl_D(\gamma)$$

Let s' be some state in $\Phi_D(a,s)$. Such an s' exists because D is consistent. By Lemma 6 and by Definitions 7 and 9, we have

$$s' = Cl_D(\sigma) \tag{4.19}$$

where

$$\sigma = e(a,s) \cup (s \cap s').$$

It suffices to prove that $\delta' \subseteq s'$. But first of all, let us prove, by induction, the following

$$\Gamma^i(\gamma) \subseteq s' \tag{4.20}$$

for every integer $i \geq 0$, where Γ is the function defined in Equation 2.8.

1. **Base Case:** $i = 0$. Assume that $l \in \Gamma^0(\gamma) = \gamma$. We need to show that $l \in s'$. There are two possibilities for $l \in \gamma$.
   a) $l \in e(a,\delta)$. It is easy to see that $l \in s'$ because

$$e(a,\delta) \subseteq e(a,s) \subseteq \sigma \subseteq Cl_D(\sigma) = s'.$$

b) $l \notin e(a,\delta)$, $l \in \delta$, *and* $\bar{l} \notin pc(a,\delta)$. Since $\delta \subseteq s$, we have $l \in s$. Because of the completeness of s, it follows that $\bar{l} \notin s$. Accordingly, we have

$$\bar{l} \notin (s \cap s') \tag{4.21}$$

On the other hand, because $\bar{l} \notin pc(a,\delta)$, $\bar{l} \notin s$, and $(e(a,s) \setminus s) \subseteq pc^0(a,\delta) \subseteq pc(a,\delta)$, we have

$$\bar{l} \notin e(a,s) \tag{4.22}$$

From (4.21) and (4.22), it follows that $\bar{l} \notin \sigma$. In addition, since $\bar{l} \notin pc(a,\delta)$, by Lemma 5, we have $\bar{l} \notin s' \setminus \sigma$. Accordingly, we have $\bar{l} \notin s'$. Because s' is complete, we can conclude that $l \in s'$.

2. **Inductive Step:** Assume that (4.20) is true for all $i \leq k$. We need to show that $\Gamma^{k+1}(\gamma) \subseteq s'$. Let l be a literal in $\Gamma^{k+1}(\gamma)$. By the definition of $\Gamma^{k+1}(\gamma)$, there are two possibilities for l:
   a) $l \in \Gamma^k(\gamma)$. Clearly, in this case, we have $l \in s'$.
   b) *there exists a static causal axiom*

   $$\varphi \textbf{ s\_causes } l$$

   in D such that $\varphi \subseteq \Gamma^k(\gamma)$.
   By the inductive hypothesis, we have $\varphi \subseteq s'$. Hence, l must hold in s'.

   Therefore, in both cases, we have $l \in s'$. This implies that $\Gamma^{k+1}(\gamma) \subseteq s'$.

As a result, (4.20) always holds. By Lemma 1, we have

$$\delta' = \bigcup_{i=0}^{\infty} \Gamma^i(\gamma) \subseteq s'$$

Since s' is a state, δ' is consistent. Thus, by the definition of the Φ^0-function, we have

$$\Phi^0(a,\delta) = \{\delta'\}$$

Furthermore, $\delta' \subseteq s'$ for every $s' \in \Phi(a,s)$.

- Since δ is valid, there exists a state s such that $\delta \subseteq s$. Assume that the knowledge effect axiom

  $$a \textbf{ determines } \theta$$

  belongs to D.

  Since in every state of the world, exactly one literal in θ holds, there exists a literal $g \in \theta$ such that g holds in s and for all $g' \in \theta \setminus \{g\}$, g' does not hold in s. Hence, we have $\delta \cup \{g\} \subseteq s$. By Corollary 1, we have $\delta' = Cl_D(\delta \cup \{g\}) \subseteq Cl_D(s) = s$. Hence, δ' is consistent. By the definition of Φ, we have $\delta' \in \Phi_D(a,\delta)$. Since $\delta' \subseteq s$, δ' is a valid a-state.

□

We will next define an extension of Φ^0 for reasoning over conditional plans.

Definition 28 (Extended Approximation Transition Function) Let D be a domain specification D and Φ^0 be its approximation transition function. For any plan c and a-state δ,

- if $c = [\ ]$ then
$$\widehat{\Phi}^0(c, \delta) = \{\delta\}$$
- if $c = [a \circ p]$, where a is a non-sensing action and p is a sub-plan, then
  - $\widehat{\Phi}^0(c, \delta) = \bot$ if $\Phi^0(a, \delta) = \bot$ or $\widehat{\Phi}^0(p, \delta') = \bot$ for some $\delta' \in \Phi^0(a, \delta)$; and
  - $\widehat{\Phi}^0(c, \delta) = \bigcup_{\delta' \in \Phi^0(a,\delta)} \widehat{\Phi}^0(p, \delta')$, otherwise.
- if $c = [a \circ \textbf{cases}(\{l_j \rightarrow c_j\}_{j=1}^n)]$, where a is a sensing action and c_j's are sub-plans, then
  - $\widehat{\Phi}^0(c, \delta) = \bot$ if $\Phi^0(a, \delta) = \bot$ or $\widehat{\Phi}^0(p, \delta') = \bot$ for some $\delta' \in \Phi^0(a, \delta)$; and
  - $\widehat{\Phi}^0(c, \delta) = \bigcup_{1 \leq j \leq n, \delta' \in \Phi^0(a,\delta), g_j \text{ holds in } \delta'} \widehat{\Phi}^0(c_j, \delta')$, otherwise.

During the execution of a plan c, when a non-sensing action a is encountered (Second Item), there are three possibilities: $\Phi(a, \delta) = \bot$, $\Phi(a, \delta) = \emptyset$, or $\Phi(a, \delta) = \{\delta'\}$ for some a-state δ'. If the first case occurs then the result of execution of c in δ is also $\bot$. In this case, we say that c is not executable in δ; otherwise, c is *executable* in δ. If the second case occurs then by the definition, $\widehat{\Phi}(p, \delta) = \emptyset$. One may notice that, by Proposition 2, this case takes place only if δ is invalid, or the domain is inconsistent. When $\Phi(a, \delta) = \{\delta'\}$, then the result of the execution of c in δ is exactly as the result of the execution of the rest of c in δ'.

The above remark and Examples 32 and 32 imply that in some cases, for a plan c and an a-state δ, $\widehat{\Phi}(p, \delta)$ may be empty. Intuitively, this is because either δ is invalid or the domain is inconsistent. We will next extend Proposition 2 to consider conditional plans for consistent action theories, defined as follows.

Definition 29 (Consistent Action Theories) An action theory (D, O) is *consistent* if D is consistent and its initial a-state, defined by $Cl_D(\{l \mid \textbf{initially}\ l \in O\})$, is valid.

The next shows that the execution of an executable plan from a valid a-state of a consistent action theory will result in at least one valid a-state.

Theorem 17 Let (D, O) be a consistent action theory and let δ be its initial a-state. For every conditional plan c, if $\widehat{\Phi}(c, \delta) \neq \bot$ then $\widehat{\Phi}^0(c, \delta)$ contains at least one valid a-state.

Proof. Let us prove this proposition by using structural induction on c.

1. $c = [\ ]$. Trivial.
2. $c = [a \circ p]$, *where p is a conditional plan and a is a non-sensing action.*

   Assume that Theorem is true for p. We need to prove that it is also true for c. Suppose $\widehat{\Phi}(c, \delta) \neq \bot$. Clearly we have $\Phi(a, \delta) \neq \bot$. It follows from 2 that $\Phi^0(a, \delta) = \{\delta'\}$ for some valid a-state δ'. By inductive hypothesis, $\widehat{\Phi}^0(p, \delta')$ contains at least one valid a-state. Hence, $\widehat{\Phi}^0(c, \delta) \neq \bot$ contains at least one valid a-state.
3. $c = [a \circ \textbf{cases}(\{l_j \rightarrow c_j\}_{j=1}^n)]$, *where a is a sensing action that senses $l_1, \ldots, l_n$.*

   Assume that the theorem is true for c_j's. We need to prove that it is also true for c. Because $\widehat{\Phi}(c, \delta) \neq \bot$, we have $\Phi(a, \delta) \neq \bot$. Again, δ is valid, $\Phi^0(a, \delta)$ contains at least one valid a-state δ' where $\delta' = Cl_D(\delta \cup \{l_k\})$ for some k. This implies that l_k holds in δ'. By the inductive hypothesis, we have $\widehat{\Phi}^0(c_k, \delta')$ contains at least one valid a-state. Since $\widehat{\Phi}^0(c_k, \delta') \subseteq \widehat{\Phi}^0(c, \delta)$, $\Phi^0(c, \delta)$ contains at least one valid a-state.

□

The above theorem implies that if the action theory (D, O) is consistent and δ is its initial a-state then the execution of a conditional plan c in δ will yield at least a valid trajectory $\delta_0 a_1 \delta_1 a_2 \ldots a_n \delta_n$ where $\delta_i \in \Phi^0(a_i, \delta_{i-1})$ for $i = 1, \ldots, n$, each δ_i is a valid a-state, provided that c is executable in δ. This is consistent with the fact that if the initial a-state is complete (i.e., if we have complete information) then the execution of an executable plan in the initial a-state would return a valid trajectory. The approximation semantics of $\mathcal{B}_K$ action theories is defined next.

Definition 30 (Approximation Model) Let (D, O) be an $\mathcal{B}_K$ action theory.

- $\delta_0 = Cl_D(\{l \mid \textbf{initially } l \in O\})$ is called the *initial a-stae* of (D, O).
- (δ_0, Φ^0) is called the *approximation model* of (D, O) where Φ^0 is the approximation transition function of D.

We next define the entailment relationship between $\mathcal{B}_K$ action theories and queries using Φ^0.

Definition 31 (Approximation Entailment) Let (D, O) be an action theory. For a plan c and a fluent formula φ, we say that

- (D, O) entails the query **Knows** φ **after** c and write

  $$(D, O) \models^0 \textbf{Knows } \varphi \textbf{ after } c$$

  if $\widehat{\Phi}^0(c, \delta_0) \neq \perp$ and φ is true in every a-state in $\widehat{\Phi}^0(c, \delta_0)$ for every approximation model (δ_0, Φ^0) of (D, O); and

- (D, O) entails the query **Kwhether** φ **after** c and write

  $$(D, O) \models^0 \textbf{Kwhether } \varphi \textbf{ after } c$$

  if $\widehat{\Phi}^0(c, \delta_0) \neq \perp$ and φ is known in every a-state in $\widehat{\Phi}^0(c, \delta_0)$ for every approximation model (δ_0, Φ^0) of (D, O).

Example 33 Consider a security window with a lock that behaves as follows. The window can be in one of the three states *opened, closed*[3] or *locked*[4]. When the window is closed or opened, pushing it *up* or *down* will *open* or *close* it respectively. When the window is closed or locked, flipping the lock will lock or close it respectively.

Now, consider a security robot that needs to make sure that the window is locked after 9 pm. Suppose that the robot has been told that the window is not open (but whether it is locked or closed is unknown).

[3]The window is closed and unlocked.
[4]The window is closed and locked.

The domain can be represented by the following action theory:

$$D_1 = \left\{ \begin{array}{l} \textbf{executable } push\_up \textbf{ if } closed \\ \textbf{executable } push\_down \textbf{ if } open \\ \textbf{executable } flip\_lock \textbf{ if } \neg open \\ \\ push\_down \textbf{ causes } closed \\ push\_up \textbf{ causes } open \\ flip\_lock \textbf{ causes } locked \textbf{ if } closed \\ flip\_lock \textbf{ causes } closed \textbf{ if } locked \\ \\ open \oplus locked \oplus closed \\ \\ check \textbf{ determines } open, closed, locked \end{array} \right\}$$

$$O_1 = \left\{ \textbf{initially } \neg open \right\}$$

We will show that

$$(D_1, O_1) \models^0 \textbf{Knows } locked \textbf{ after } p_2 \tag{4.23}$$

where

$$p_2 = check;\ \textbf{cases} \begin{pmatrix} open & \rightarrow & [] \\ closed & \rightarrow & [flip\_lock] \\ locked & \rightarrow & [] \end{pmatrix}$$

Let $p_{2,1} = []$, $p_{2,2} = [flip\_lock]$ and $p_{2,3} = []$. It is easy to see that the initial a-state of $(D_1, \mathcal{I}_1)$ is $\delta_1 = \{\neg open\}$.

We have

$$Cl_{D_1}(\delta_1 \cup \{open\}) = \{open, \neg open, closed, \neg closed, locked, \neg locked\} = \delta_{1,1}$$

$$Cl_{D_1}(\delta_1 \cup \{closed\}) = \{\neg open, closed, \neg locked\} = \delta_{1,2}$$

$$Cl_{D_1}(\delta_1 \cup \{locked\}) = \{\neg open, \neg closed, locked\} = \delta_{1,3}$$

Among those, $\delta_{1,1}$ is inconsistent. Therefore, we have

$$\Phi^0(check, \delta_1) = \{\delta_{1,2}, \delta_{1,3}\}$$

On the other hand, we have

$$\widehat{\Phi}^0(p_{2,2}, \delta_{1,2}) = \{\{locked, \neg open, \neg closed\}\}$$

and

$$\widehat{\Phi}^0(p_{2,3}, \delta_{1,3}) = \{\{locked, \neg open, \neg closed\}\}$$

Therefore, we have

$$\widehat{\Phi}^0(p_2, \delta_1) = \widehat{\Phi}^0(p_{2,2}, \delta_{1,2}) \cup \widehat{\Phi}^0(p_{2,3}, \delta_{1,3}) = \{\{locked, \neg open, \neg closed\}\}$$

Since *locked* is true in $\{locked, \neg open, \neg closed\}$, we have (4.23) holds. On the other hand, because *closed* is false in $\{locked, \neg open, \neg closed\}$, we have

$$(D_1, O_1) \not\models^0 \textbf{Knows } closed \textbf{ after } p_2 \qquad \text{but} \qquad (D_1, O_1) \models^0 \textbf{Knows } \neg closed \textbf{ after } p_2.$$

Likewise, we can prove that

$$(D_1, O_1) \models^0 \textbf{Knows } \mathit{locked} \textbf{ after } p_3 \qquad \text{and} \qquad (D_1, O_1) \models^0 \textbf{Knows } \mathit{locked} \textbf{ after } p_4.$$

where

$$p_3 = \mathit{check};\ \textbf{cases}\begin{pmatrix} \mathit{open} & \rightarrow & [\mathit{push\_down}; \mathit{flip\_lock}] \\ \mathit{closed} & \rightarrow & [\mathit{flip\_lock}; \mathit{flip\_lock}; \mathit{flip\_lock}] \\ \mathit{locked} & \rightarrow & [] \end{pmatrix}$$

$$p_4 = \mathit{check};\ \textbf{cases}\begin{pmatrix} \mathit{open} & \rightarrow & [] \\ \mathit{closed} & \rightarrow & p_2 \\ \mathit{locked} & \rightarrow & [] \end{pmatrix}$$

4.5.2 Properties of the Approximation Semantics

We will now discuss some properties of the approximation semantics defined by Φ^0, which we will refer to as the 0-approximation. Theorem 17 allows us to conclude the following property.

Theorem 18 Let (D, O) be a consistent $\mathcal{B}_K$ action theory, c be a conditional plan, and φ be a fluent formula. It holds that

$$(D, O) \models^0 \varphi \textbf{ after } c \quad \text{implies that} \quad (D, O) \models^r_{\mathcal{B}_K} \varphi \textbf{ after } c.$$

One salient feature of the 0-approximation is that it is deterministic and it can be computed in polynomial time in the size of the domain specification.

Theorem 19 *For a domain specification D, an action a, and an a-state δ, computing $\Phi^0(a, \delta)$ can be done in polynomial time in the size of D.*

Proof. This follows from the fact that computing $Cl_D(\sigma)$ and $pc(a, \delta)$ is polynomial for an arbitrary set of fluent literals σ and an a-state δ respectively. □

This leads to the following result:

Theorem 20 *$PI^{\mathcal{B}_K}$ is NP-complete with respect to Φ^0.*

The above theorem shows that reasoning and planning using the approximation transition function Φ^0 has lower complexity than planning with respect to the full semantics. Yet, the price one has to pay is the incompleteness of this approximation as shown in the following example.

Example 34 Consider the action theory (D, O) with

$$D = \begin{cases} a \textbf{ causes } f \textbf{ if } g \\ a \textbf{ causes } f \textbf{ if } \neg g \end{cases}$$

We can easily check that $(D, \emptyset) \models^r_{\mathcal{B}_K} \textbf{Knows } f \textbf{ after } [a]$ and $(D, \emptyset) \not\models^0 \textbf{Knows } f \textbf{ after } [a]$. □

The above example highlights the main weakness of the 0-approximation in that it does not allow for reasoning by cases for non-sensing actions or in the presence of disjunctive initial situation.

4.6 Narrative Revisited—Diagnosis, Diagnostic Planning, and Repairing

Chapter 3 develops a language for representing and reasoning about observations. The discussion on narratives is limited to domain specifications in $\mathcal{B}$ and the definition of the notion of a model of a narrative. We will now consider narratives with $\mathcal{B}_K$ domain specifications and define the notion of *diagnosis*, *diagnostic planning*, and *repair* for dynamic systems.

Let us now illustrate a few problems that the process of diagnosis problem solving needs to consider through an modified version of the introductory story in Chapter 3.

> John gets up in the morning. He turns on the switch of his lamp, and reads the morning newspaper. He then turns off the switch and does other things before going to work. After he gets home from work, he enters his room and turns on the switch of his lamp again. This time, *the lamp does not turn on*. John thinks that maybe either the bulb is broken, or the switch of the lamp is broken, or the power cord is broken, or there is no power at the outlet. He does nothing about it and goes to his bathroom and turns on the light switch, observing that even that light does not turn on. He thinks perhaps there is no power at home, but then he notices that his electric clock is working, so he figures that there is power in at least part of his home. Now he is worried and goes to his garage to check his fuse box and finds that one of the fuses is blown. He replaces that fuse and comes back to his room. He turns on his lamp switch and voila it works.

The narrative illustrates that diagnostic problem solving must involve reasoning about the evolution of a dynamical system. Triggered by an observation of system behavior that is inconsistent with expected behavior – in this case, the fact that when John turned on the lamp it did not emit light, diagnostic problem solving involves:

- generating candidate diagnoses based on an incomplete history of events that have occurred and observations that have been made.
- in the event of multiple candidate diagnoses, performing actions to enable observations that will discriminate candidate diagnoses. The selection of a particular action is often biased towards confirming the most likely diagnosis, or the one that is easiest to test.
- generating (possibly conditional) plans, comprising both world-altering actions and sensing actions, to discriminate candidate diagnoses.
- updating the space of diagnoses in the face of changes in the state of the world, and in the face of new observations.

We will now formulate the notion of diagnosis with respect to a narrative that represents the evolution of the system of our interest. We assume that the system is composed of a distinguished set of components that can malfunction. Associated with each component c, is the distinguished fluent $ab(c)$, denoting that the component c is abnormal or broken. Also associated with each component is the distinguished action $miracle(c)$, a wildcard or *miracle* action that may be used to explain unexpected observations relating to $ab(c)$. Note that all representation of all nontrivial domains will involve other actions and static causal axioms that affect the truth of $ab(c)$.

Definition 32 (System) A *system Sys* is a tuple $(SD, COMPS, \Gamma)$ where

- $COMPS = \{c_1, \ldots, c_n\}$ is a finite set of components.
- SD is a domain specification characterizing the behavior of the system, and augmented with fluent effect actions of the form $miracle(c)$ **causes** $ab(c)$, for each component c in $COMPS$.
- Γ is a collection of observations about the system.

Example 35 The story in Example 23 can be described by a system description $Sys_1 = (SD_{bulb}, \{bulb\}, \Gamma_{bulb})$ where

$$SD_{bulb} = D_{bulb} \cup \{miracle(bulb) \textbf{ causes } ab(bulb)\}$$

D_{bulb} and Γ_{bulb} are as specified in Example 23. □

4.6.1 Models of a System

According to Definition 32, a system $Sys = (SD, COMPS, \Gamma)$ includes a narrative (SD, Γ) and a set of components $COMPS$. As such, models of the narrative (SD, Γ) (Chapter 3, Definition 17) reflects the evolution of the system and thus can be viewed as a diagnosis for its evolution. We will introduce several types of models of a system and discuss situations in which we may prefer one type of model over the other.

Definition 33 (Abnormality-Free Model) Given a system $Sys = (SD, COMPS, \Gamma)$, $\mathcal{M}$ is an *abnormality-free model* of Sys if $\mathcal{M}$ is a model of the narrative (SD, Γ) and for every $c \in COMPS$ and situation $\mathsf{s} \in \mathbf{S}$,

$$M \models ab(c) \textbf{ at } \mathsf{s} \textit{ iff } M \models ab(c) \textbf{ at } \mathsf{s}_0.$$

Abnormality-free models of our system are those models of the narrative comprising the system description and observations that dictate *no new abnormal components*. I.e., the states and action occurrences entailed by the model only designate components abnormal if they were already abnormal in the initial situation, s_0.

Example 36 To illustrate, consider the system $Sys_2 = (SD_1, \{bulb\}, \Gamma_2)$ where SD_1 is the domain specification in Example 35 but Γ_2 contains only two fluent facts stating that initially the light was on and it was off at a later situation:

$$\Gamma_2 = \begin{cases} light\_on \textbf{ at } \mathsf{s}_0 \\ \neg light\_on \textbf{ at } \mathsf{s}_1 \end{cases}$$

An abnormality-free model $\mathcal{M}$ of the system would be (Ψ, Σ) with $\Psi(\mathsf{s}_0) = \{light\_on, \neg ab(bulb)\}$, $\Psi(\mathsf{s}_1) = \{\neg light\_on, \neg ab(bulb)\}$, $\Sigma(\mathsf{s}_0) = []$, and $\Sigma(\mathsf{s}_1) = turn\_off$. The model is represented graphically as follows.

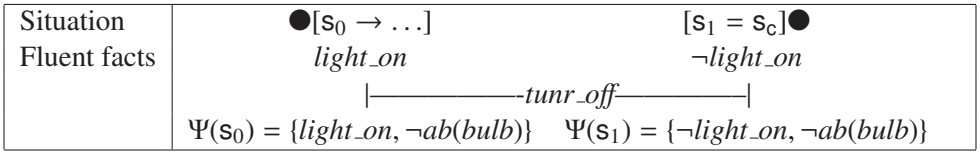

Figure 4.1: An abnormality-free model of the system Sys_2

□

We contrast abnormality-free models, with *abnormality models* that do indeed entail new abnormal components.

Definition 34 (Abnormality Model) Given a system $Sys = (SD, COMPS, \Gamma)$, $\mathcal{M}$ is an *abnormality model* of *Sys* if $\mathcal{M}$ is a model of the narrative (SD, Γ) and for some $c \in COMPS$ and $\mathsf{s} \in \mathbf{S}$,

$$M \models ab(c) \textbf{ at } \mathsf{s} \textit{ and } M \models \neg ab(c) \textbf{ at } \mathsf{s}_0.$$

Example 37 Returning to our previous example, an abnormality model of $Sys_2 = (SD_1, \{bulb\}, \Gamma_2)$ is (Ψ, Σ) with $\Psi(\mathsf{s}_0) = \{light\_on, \neg ab(bilb)\}$, $\Psi(\mathsf{s}_1) = \{\neg light\_on, ab(bulb)\}$, $\Sigma(\mathsf{s}_0) = []$, and $\Sigma(\mathsf{s}_1) = burn\_out(bulb)$. The model is represented graphically as follows.

Situation	$\bullet[\mathsf{s}_0 \rightarrow \ldots]$	$[\mathsf{s}_1 = \mathsf{s}_c]\bullet$
Fluent facts	$light\_on$	$\neg light\_on$
	$\vert$——————$burn\_out(bulb)$——————$\vert$	
	$\Psi(\mathsf{s}_0) = \{light\_on, \neg ab(bulb)\}$	$\Psi(\mathsf{s}_1) = \{\neg light\_on, ab(bulb)\}$

Figure 4.2: An abnormality model of the system Sys_2

□

We often prefer abnormality-free models over abnormality models because they preclude the introduction of new faults, however this is not universally the case. Preference criteria to select the best account for the observations may be dependent on other factors such as parsimony or probabilistic information. For example, imagine a variant of our simple light bulb example in which the lights are office lights that stay on permanently. We no longer have actions *turn_on* and $turn\_off$. Instead, the lights remain on indefinitely unless a bulb burns out, or someone gets a ladder, removes the protective panels and removes the bulbs from the light fixtures. In such a system, the preferred accounting for the lights being off is the abnormality model that entails that the bulb burned out, rather than the abnormality-free model that entails the sequence of actions involving the ladder, panel removal and bulb removal.

As with any axiomatization of a physical system, it is often difficult to create an axiomatization that has anticipated and includes all possible actions that can lead to an abnormal component. As such, we often include a so-called *miracle* action, that is always possible, but that we only wish to consider when there is no other accounting for the observations. In our diagnosis domain, such miracle actions are included in our axiomatization as fluent effect axioms of the form

$$miracle(c) \textbf{ causes } ab(c).$$

We refer to models that exclude such miracle actions as *miracle-free models*, and we universally prefer abnormality or abnormality-free models that are also miracle-free.

Definition 35 (Miracle-Free Model) Given a system $Sys = (SD, COMPS, \Gamma)$, $\mathcal{M}$ is a *miracle-free model* of *Sys* if $\mathcal{M}$ is a model of the narrative (SD, Γ) and for all $c \in COMPS$ and for all $\mathsf{s} \in \mathbf{S}$,

$$M \not\models miracle(c) \textbf{ occurs\_at } \mathsf{s}.$$

It is easy to see that the models in Examples 36 and 37 are miracle-free models of the system $(SD_1, \{bulb\}, \Gamma_2)$.

4.6.2 Diagnoses and Explanations

From the models identified in the previous section, we can extract some compact accounting for our observations. We distinguish between the notion of an diagnosis and explanation.

Definition 36 (Diagnosis) A diagnosis for system $Sys = (SD, COMPS, \Gamma)$ with respect to model M of (SD, Γ) and situation $\mathtt{s}$ is the set of components $\Delta \subseteq COMPS$ such that

$$M \models ab(c) \textbf{ at } \mathtt{s}$$

for all $c \in \Delta \subseteq COMPS$.

A diagnosis with respect to $\mathtt{s_c}$ is called a *current fluent diagnosis*.

A diagnosis Δ (wrt. a situation $\mathtt{s}$) is *minimal* if there exists no diagnosis Δ' (wrt. $\mathtt{s}$) such that $\Delta' \subset \Delta$.

Observe that when M is an abnormality-free model then the diagnosis is empty or equals the set of abnormal components which existed in the initial situation. In such cases, the observations in a narrative can be *explained* by the sequence of actions (possibly exogenous) that have occurred. We will call this type of explanations consistency-based explanations. When it is sufficient for us to explain the observations in every possible case, we call it an abductive explanations.

Definition 37 (Consistency-Based and Abductive Explanations) Given a system $Sys = (SD, COMPS, \Gamma)$ and a model M of (SD, Γ). Let $actions(M)$ be the set of occurrence facts and precedence facts of the forms (3.2), (3.3), and (3.4), (i.e., facts of the forms α **between** $\mathtt{s}_1, \mathtt{s}_2$, α **occurs_at** $\mathtt{s}_1$, $\mathtt{s}_1$ **preceeds** $\mathtt{s}_2$) that are true in M.

We call $action(M)$ a consistency-based explanation for the system Sys with respect to the model M.

If additionally, $(SD,\ actions(M)) \models \Gamma$, then we refer to $actions(M)$ as an abductive explanation for our observations.

The difference between consistency-based and abductive explanations is highlighted next.

Example 38 Consider the system $Sys_2 = (SD_2, \{c_1, c_2\}, \Gamma_2)$ with

$$SD_2 = \begin{cases} drop(glass) & \textbf{causes} & broken(glass) \vee \neg broken(glass) \\ destroy(glass) & \textbf{causes} & broken(glass) \\ glass\_shards & \textbf{if} & broken(glass) \end{cases}$$

and

$$\Gamma_2 = \begin{cases} \neg glass\_shards & \textbf{at} & \mathtt{s}_0 \\ glass\_shards & \textbf{at} & \mathtt{s}_1 \end{cases}$$

Notive that the action $drop(glass)$ is a non-deterministic action. It may or may not break the glass. We can simulate this effect by introducing three additional fluents may_broken, $broken_yes$, and $broken\_no$, replacing the fluent effect axiom of $drop(glass)$ with

$$drop(glass) \textbf{ causes } may\_broken$$

and add to SD_2 the following static causal axioms

$$\begin{gathered} may\_broken, broken\_yes \textbf{ s\_causes } \neg broken\_glass \\ may\_broken, broken\_no \textbf{ s\_causes } broken\_glass \\ broken\_yes \oplus broken\_no \end{gathered}$$

Situation	$\bullet[s_0 \rightarrow \ldots]$	$[s_1 = s_c]\bullet$
Fluent facts	$\neg glass\_shards$	$glass\_shards$
	$\vert$——————$drop(glass)$——————$\vert$	
	$\Psi(s_0) = \{\neg broken(glass), \neg glass\_shards\}$	$\Psi(s_1) = \{broken(glass), glass\_shards\}$

Figure 4.3: An abnormality model of the system Sys_2

Obviously, $M = (\Psi_1, \Sigma_1)$ with $\Psi_1(s_0) = \{\neg broken(glass), \neg glass\_shards\}$, $\Psi_1(s_1) = \{broken(glass), glass\_shards\}$ and $\Sigma_1(s_0) = []$, $\Sigma_1(s_1) = drop(glass)$ is an model for Sys_2 (Figure 4.3).
This implies that $action(M) = \{drop(glass)$ **between** s_0, s_0 **preceeds** $s_1\}$ is a consistency-based explanation for Sys_2.

However, the narrative $(SD_2, action(M))$ does not entail "$glass\_shards$ **at** s_1" because it has a model (Ψ_2, Σ_2) with $\Psi_2(s_0) = \Psi_2(s_1) = \{\neg broken(glass), \neg glass\_shards\}$, and $\Sigma_2(s_0) = []$, $\Sigma_2(s_1) = drop(glass)$ in which "$glass\_shards$ **at** s_1" does not hold. □

There are situations in which consistency-based explanations and abductive explanations coincide.

Theorem 21 Let $Sys = (SD, COMPS, \Gamma)$ be a deterministic system with complete information about the initial situation. Then, $actions(M)$ is a consistency-based explanation iff $actions(M)$ is an abductive explanation.

Proof. By definition each abductive explanation is a consistency-based explanation. Thus, to prove the theorem, we need to show that if $M = (\Psi, \Sigma)$ is a model of Sys, then $actions(M)$ is an abductive explanation for Γ.

Assume that $M' = (\Psi', \Sigma')$ is an arbitrary model of the narrative $Q = (SD, actions(M))$. We need to prove that

$$M' \models \Gamma.$$

Since we have complete information about the initial situation, we have that $\Psi([]) = \Psi'([])$. This, together with the fact that actions in SD are deterministic imply that $\Psi(\alpha) = \Psi'(\alpha)$ for every sequence of actions α. Furthermore, since $\Sigma(s_i)$ **between** s_0, s_i is true in M for every $i = 0, \ldots, C$, we have that $\Sigma(s_i)$ **between** s_0, s_i belongs to $action(M)$. Since M' is a model of Q, we have that $\Sigma'(s_i) = \Sigma(s_0) \circ \Sigma(s_i) = \Sigma(s_i)$ for every $i = 0, \ldots, C$. This shows that M' is identical to M, i.e., M is a unique model of Q. (1)

Since M is also a model of Sys, we have that $M \models \Gamma$. (2)

It follows from (1) and (2) that $Q \models \Gamma$. The theorem is proved. □

Example 39 (Continuation of Example 35) From the observation "$light\_on$ **at** s_1" in Γ_1 we conclude that $\neg ab(bulb)$ **at** s_0 holds in every model M of Sys_1. It is easy to see that:

- There exists no abnormality-free model of Sys_1 (due to the observations "$turn\_on$ **between** s_2, s_3" and "$\neg light\_on$ **at** s_3").
- Sys_1 has one abnormality, miracle-free model $M = (\Psi, \Sigma)$ where $\Psi([]) = \emptyset$, and

  $\Sigma(s_0) = []$,
  $\Sigma(s_1) = turn\_on$,

$\Sigma(\mathtt{s}_2) = \mathit{turn\_on} \circ \mathit{turn\_off} \circ \mathit{burn\_out}(\mathit{bulb})$, and
$\Sigma(\mathtt{s}_3) = \Sigma(\mathtt{s}_c) = \mathit{turn\_on} \circ \mathit{turn\_off} \circ \mathit{burn\_out}(\mathit{bulb}) \circ \mathit{turn\_on}$.

- Sys_1 has one abnormality model that is not miracle-free, $M' = (\Psi', \Sigma')$ where $\Psi'([]) = \emptyset$, and

  $\Sigma'(\mathtt{s}_0) = []$,
  $\Sigma'(\mathtt{s}_1) = \mathit{turn\_on}$,
  $\Sigma'(\mathtt{s}_2) = \mathit{turn\_on} \circ \mathit{turn\_off} \circ \mathit{miracle}(\mathit{bulb})$, and
  $\Sigma'(\mathtt{s}_3) = \Sigma'(\mathtt{s}_c) = \mathit{turn\_on} \circ \mathit{turn\_off} \circ \mathit{miracle}(\mathit{bulb}) \circ \mathit{turn\_on}$.

Even though there are different models of Sys_1, it has only one current fluent diagnosis $\Delta = \{\mathit{bulb}\}$ because $\mathit{ab}(\mathit{bulb})$ **at** $\mathtt{s}_c$ holds in every model of the system. □

It is easy to see that every component of Sys_1 is okay initially. In the next example, we consider a variation of an example from the literature in which some components are broken in the initial situation.

Example 40 (Adapted from [SD89]) Let us consider a variation of the three bulbs example from [SD89]. The system consists of two bulbs and a battery source. The two bulbs are connected parallel to the battery source. The contact is used to connect/disconnect the battery. When the battery and the bulbs are okay, connecting the battery will turn on the light on both bulbs. Initially, we observe that the battery is disconnected and that the light is off on both bulbs. Connecting the switch and we observe that the light does not turn on in one bulb. The system is represented in the following picture.

Situation	●$[\mathtt{s}_0 \rightarrow \ldots]$	$[\mathtt{s}_1 = \mathtt{s}_c]$●
Fluent facts	$\neg on(b_1),\ \neg on(b_2)$	$on(b_1),\ \neg on(b_2)$
	|——————*turn_on*——————|	
	$\Psi(\mathtt{s}_0) = \{\neg on(b_1), \neg one(b_2)$	$\Psi(\mathtt{s}_1) = \{on(b_1), \neg one(b_2)$
	$.3\neg ab(b_1), ab(b_2)\}$	$.3\neg ab(b_1), ab(b_2)\}$

Figure 4.4: Connect the battery and only one bulb is on

The system can be represented by $Sys_2 = (SD_2, \{b_1, b_2, \mathit{battery}\}, \Gamma_2)$ with

$$SD_2 = \begin{cases} \mathit{turn\_on} \textbf{ causes } \mathit{on}(b_1) \textbf{ if } \neg\mathit{ab}(b_1) \wedge \neg\mathit{ab}(\mathit{battery}) \\ \mathit{turn\_on} \textbf{ causes } \mathit{on}(b_2) \textbf{ if } \neg\mathit{ab}(b_2) \wedge \neg\mathit{ab}(\mathit{battery}) \\ \mathit{ab}(b_1) \textbf{ s\_causes } \neg\mathit{on}(b_1) \\ \mathit{ab}(\mathit{battery}) \textbf{ s\_causes } \neg\mathit{on}(b_1) \\ \mathit{ab}(b_2) \textbf{ s\_causes } \neg\mathit{on}(b_2) \\ \mathit{ab}(\mathit{battery}) \textbf{ s\_causes } \neg\mathit{on}(b_2) \end{cases}$$

and

$$\Gamma_2 = \begin{cases} \neg\mathit{on}(b_1) \wedge \neg\mathit{on}(b_2) \textbf{ at } \mathtt{s}_0 \\ \mathit{on}(b_1) \wedge \neg\mathit{on}(b_2) \textbf{ at } \mathtt{s}_1 \\ \mathit{turn\_on} \textbf{ between } \mathtt{s}_0, \mathtt{s}_1 \end{cases}$$

It is easy to check that Sys_2 has only one abnormality- and miracle-free model $M = (\Psi, \Sigma)$ where

$\Psi([]) = \{\mathit{ab}(b_2), \neg\mathit{ab}(b_1), \neg\mathit{ab}(\mathit{battery})\}$, and

$\Sigma(s_0) = []$,
$\Sigma(s_1) = \Sigma(s_c) = turn\_on$.

Since $ab(b_2)$ **at** s_c holds in M, we have that $\Delta = \{b_2\}$ is a current fluent diagnosis of Sys_2. □

4.6.3 Diagnostic and Repair Planning

The diagnostic process discussed in the previous subsection will generate a set of candidate diagnoses. However, diagnosis is only the first step in dealing with an errant system. In most cases we will attempt to discriminate these diagnoses with the objective of identifying a unique diagnosis and/or reducing our space of candidate diagnoses to a point where a repair plan can be conceived. We are operating under the assumption that we cannot directly observe the state of abnormality of the various components of the system. Nevertheless, we can make other observations about the system, add them to Γ, and then refine our diagnoses.

In general, the fluents in the system are of two kinds: *observable* and *unobservable*. A simple *generic observation*[5] leads the agent to know the value of the observable fluents. By knowing the relationship between the observable and unobservable fluents, and the values of the observable fluents, we can sometimes deduce the values of unobservable fluents. We can also use direct sensing actions to sometimes determine their value. Given a set of candidate diagnoses, we can execute a plan – perhaps including some sensing actions and conditional branches – and make the generic observations to obtain additional information that will help reduce the space of possible diagnoses. Such plans are distinguished in that they can have knowledge goals in addition to goals relating to the state of the world. We refer to plans that attempt to reduce our space of diagnoses as *diagnostic plans*. A diagnostic plan that includes some repair is called a *repair plan*.

The main goal of this section is to formally define a diagnostic plan. In the process, we will see the need for sensing actions in and the notions of *observable* and *unobservable fluents*. But first we will illustrate the role of diagnostic planning through a simple example.

Example 41 Let SD_3 be the system $(SD_3 \cup SD_{ab}, \{bulb, switch\}, \Gamma_3)$ where

$$SD_3 = \begin{cases} turn\_on \textbf{ causes } light\_on \textbf{ if } \neg ab(bulb), \neg ab(switch), connected(bulb, switch) \\ turn\_off \textbf{ causes } \neg light\_on \textbf{ if } connected(bulb, switch) \\ disconnect(bulb, switch) \textbf{ causes } \neg connected(bulb, switch) \textbf{ if } connected(bulb, switch) \\ connect(bulb, test\_device) \textbf{ causes } connected(bulb, test\_device) \textbf{ if } \neg connected(bulb, switch) \\ light\_on \textbf{ if } connected(bulb, test\_device), \neg ab(bulb) \end{cases}$$

[5]Here we distinguish between generic observations and sensing actions. We assume that the agent is constantly performing 'generic observations' and thus knows the truth value of the observable fluents at all times. In contrast, sensing actions require the agent's effort.

$$\Gamma_3 = \begin{cases} \textit{turn\_on} \textbf{ occurs\_at } \mathtt{s}_0 \\ \textit{turn\_off} \textbf{ occurs\_at } \mathtt{s}_1 \\ \textit{turn\_on} \textbf{ between } \mathtt{s}_2, \mathtt{s}_3 \\ \mathtt{s}_0 \textbf{ preceeds } \mathtt{s}_1 \\ \mathtt{s}_1 \textbf{ preceeds } \mathtt{s}_2 \\ \mathtt{s}_2 \textbf{ preceeds } \mathtt{s}_3 \\ \neg \textit{light\_on} \textbf{ at } \mathtt{s}_0 \\ \textit{connected}(\textit{bulb}, \textit{switch}) \textbf{ at } \mathtt{s}_0 \\ \neg \textit{connected}(\textit{bulb}, \textit{test\_device}) \textbf{ at } \mathtt{s}_0 \\ \textit{light\_on} \textbf{ at } \mathtt{s}_1 \\ \neg \textit{light\_on} \textbf{ at } \mathtt{s}_2 \\ \neg \textit{light\_on} \textbf{ at } \mathtt{s}_3 \end{cases}$$

and

$$SD_{ab} = \begin{cases} \textit{burn\_out}(\textit{bulb}) \textbf{ causes } \textit{ab}(\textit{bulb}) \\ \textit{miracle}(\textit{bulb}) \textbf{ causes } \textit{ab}(\textit{bulb}) \\ \textit{break}(\textit{switch}) \textbf{ causes } \textit{ab}(\textit{switch}) \\ \textit{miracle}(\textit{switch}) \textbf{ causes } \textit{ab}(\textit{switch}) \end{cases}$$

Here, the only observable fluent is *light_on*.

It is easy to see that there are two current fluent diagnoses for SD_3: $\Delta_1 = \{\textit{bulb}\}$ and $\Delta_2 = \{\textit{switch}\}$ which correspond to the models (Ψ, Σ_1) and (Ψ, Σ_2) of $(SD_3 \cup SD_{ab}, \Gamma_3 \cup \{\neg \textit{ab}(\textit{bulb}) \textbf{ at } \mathtt{s}_0, \neg \textit{ab}(\textit{switch}) \textbf{ at } \mathtt{s}_0\})$ where

$\Psi([]) = \{\textit{connected}(\textit{bulb}, \textit{switch}), \neg \textit{light\_on}, \neg \textit{ab}(\textit{switch}), \neg \textit{ab}(\textit{bulb})\}$, and

$\Sigma_1(\mathtt{s}_0) = []$,
$\Sigma_1(\mathtt{s}_1) = \textit{turn\_on}$,
$\Sigma_1(\mathtt{s}_2) = \textit{turn\_on} \circ \textit{turn\_off} \circ \textit{burn\_out}(\textit{bulb})$,
$\Sigma_1(\mathtt{s}_3) = \Sigma_1(\mathtt{s}_c) = \textit{turn\_on} \circ \textit{turn\_off} \circ \textit{burn\_out}(\textit{bulb}) \circ \textit{turn\_on}$, and

$\Sigma_2(\mathtt{s}_0) = []$,
$\Sigma_2(\mathtt{s}_1) = \textit{turn\_on}$,
$\Sigma_2(\mathtt{s}_2) = \textit{turn\_on} \circ \textit{turn\_off} \circ \textit{break}(\textit{switch})$,
$\Sigma_2(\mathtt{s}_3) = \Sigma_2(\mathtt{s}_c) = \textit{turn\_on} \circ \textit{turn\_off} \circ \textit{break}(\textit{switch}) \circ \textit{turn\_on}$. □

In our example, there are two components: *bulb* and *switch*. When the light does not go on after turning the switch on, the diagnosis is that either the bulb is abnormal or the switch is broken (i.e., abnormal). To get to a unique diagnosis we can take the bulb out and connect it to a testing device and observe if the bulb works with the testing device. If it does, it means that the bulb is fine and hence the switch must be broken. (Now to fix the system we have to fix the switch and also remember to put the bulb back on.) If the bulb does not work then the bulb is broken. If we employ the notion of minimal diagnosis we will conclude that the switch is fine.

The above example shows that we could identify the defective components of the system by executing a simple plan (a sequence of actions) and observing its result. In the above example, the plan is [*turn_off* ∘ *get_bulb* ∘ *put_bulb*] and the observation is *light_on*.

Our next goal is to extend the notion of entailment of queries w.r.t narratives (Definition 18) to consider diagnostic queries. This entailment gives insight into what planning with incompleteness and sensing w.r.t a narrative means. Intuitively, since the narrative may not be complete (or does not have sufficient observations) to arrive at a unique model, multiple models may tell us that a situation s may correspond to many different states, only one of which corresponds to s in reality. Thus we have a set of c-states from which we need to verify the correctness of a conditional plan with respect to a goal. More formally,

Definition 38 (Possible State wrt. a Situation) Let $N = (D, \Gamma)$ be a narrative. We say s is a possible state corresponding to situation s, if there exists a model (Ψ, Σ) of N such that $\Psi(\Sigma(\mathsf{s})) = s$. We say $\langle s, \Sigma\rangle$ is a c-state corresponding to situation s, if s is a possible state corresponding to situation s and Σ is the set of all possible states corresponding to situation s.

A natural generalization of the query languages $\mathcal{L}_Q$ for narratives with $\mathcal{B}_K$ domain specification are queries of the following form

$$\textbf{Knows } \varphi \textbf{ after } c \textbf{ at } \mathsf{s} \tag{4.24}$$

$$\textbf{Kwhether } \varphi \textbf{ after } c \textbf{ at } \mathsf{s} \tag{4.25}$$

$$\varphi \textbf{ during } c \textbf{ at } \mathsf{s} \tag{4.26}$$

where φ is a fluent formula, c is a conditional plan, and s is a situation. For a query q of the above form, c and s are said to be its *plan* and *situation* respectively. The entailment between a narrative and the above types of query is defined next.

Definition 39 (Entailment of Narrative (Revisited)) Let (D, Γ) be a narrative. Let q be a query of $\mathcal{L}_Q$ whose plan is c and whose situation is s. q is said to be *entailed* by (D, Γ), denoted by $(D, \Gamma) \models q$, if for every c-state $\langle s, \Sigma\rangle$ corresponding to s, $\widehat{\Phi}(c, \langle s, \Sigma\rangle) \neq \bot$ and

- if $q =$ **Knows** φ **after** c **at** s then φ is known to be true in every c-state belonging to $\widehat{\Phi}(c, \langle s, \Sigma\rangle)$; or
- if $q =$ **Kwhether** φ **after** c **at** s, then for every c-state $\langle s', \Sigma'\rangle$ belonging to $\widehat{\Phi}(c, \langle s, \Sigma\rangle)$ and for every state $s'' \in \Sigma'$, $s' \sim_\varphi s''$; or
- if $q = \varphi$ **during** c **at** s then for all trajectories of the form $\langle s_1, \Sigma_1\rangle, \ldots, \langle s_n, \Sigma_n\rangle$ of c wrt. $\langle s, \Sigma\rangle$, $s_i \sim_\varphi s_j$ for $1 \leq i \neq j \leq n$.

In the following definition, we define the notion of a temporal knowledge plan that will be useful for the introduction of a diagnostic plan.

Definition 40 (Temporal Knowledge Plan) Let (D, Γ) be a narrative, L_M, L_W, and L_K be sets of fluent literals. A plan c is a *temporal knowledge plan* of (D, Γ) with respect to (L_M, L_W, L_K) if

- $(D, \Gamma) \models l$ **during** c **at** $\mathsf{s_c}$ for every $l \in L_M$
- $(D, \Gamma) \models$ **Kwhether** l **after** c **at** $\mathsf{s_c}$, for every $l \in L_W$ and
- $(D, \Gamma) \models$ **Knows** l **after** c **at** $\mathsf{s_c}$, for every $l \in L_K$.

Observe that if a fluent literal is known to be true, it is also known to be true or known to be false. Therefore, we have

Remark 6 If c is a temporal knowledge plan of (D, Γ) with respect to $(L_M,\ L_W,\ L_K)$ then $L_K \subseteq L_W$.

We are now ready to define what a diagnostic plan is. Intuitively, it is a conditional plan, possibly with sensing actions which when executed in the current situation gives us enough information to reach a unique diagnosis. In addition, we may have certain restrictions, such as:

- Certain literals are not allowed to change their values during the execution of the plan. We will refer to such literals as *protected literals.*
- Certain literals are allowed to change during the execution of the plan, but we require that at the end of the execution of the plan, their value be the same as it was before the plan was executed. We refer to such literals as *restored literals.*
- Certain literals are allowed to change during the execution of the plan, but we require that at the end of the plan, their value be either the same as it was before the plan was executed or be $false$. Such literals will be referred to as *fixable literals.* (This accommodates repair, where *ab* fluents can be made $\neg ab$.)

The intuition behind the above restrictions are as follows. Sometimes we do not want to affect certain aspects of a system while diagnosing. These aspects may be too dangerous or too valuable to tinker with, even if our intention is to revert them to their original state after changing their state. Also, changing their state may defeat the whole purpose. For example, to stabilize the leaning tower of Pisa we may not break it and rebuild it. Thus our option is to do unobtrusive sensing (perhaps through X-rays) to diagnose. Fluent literals corresponding to such aspects are labeled as *protected literals*. For certain other aspects of the system, we are allowed to tinker with them while diagnosing but we need to bring them back to their original stage. Examples of such aspects include, opening a flashlight to differentiate between if the bulb is bad, or if the batteries are down, or if the connection is bad. But while opening the flashlight, even if the connection was initially fine, we change the state of the connection. Thus we are required to put the flashlight back so that the state of the connection reverts back to its original state. Another similar example is the disassembling of a car engine to diagnose it. We need to put it back. Fluent literals corresponding to such aspects are labeled as *restored literals* . If we allow integration of repair with diagnostic planning, then while diagnosis or execution of the diagnostic plan we may allow that certain abnormal components are fixed but not vice-versa. Such literals are labeled as *fixable* literals.

We now define diagnostic plans as follows.

Definition 41 (Diagnostic Plan) Given a system $SD = (SD, COMPS, \Gamma)$. A temporal knowledge plan p of (SD, Γ) with respecto to (L_M, L_W, L_K), where

$$L_M,\ L_W,\ L_K \subseteq \{ab(c) \mid c \in COMPS\} \cup \{\neg ab(c) \mid c \in COMPS\}$$

is

- a *diagnostic plan* of SD wrt. (L_M, L_W, L_K) if
  - $\{ab(c) \mid c \in COMPS\} \cup \{\neg ab(c) \mid c \in COMPS\} \subseteq L_M$ and
  - $L_W \subseteq \{ab(c) \mid c \in COMPS\}$.

  In particular,

- if $L_W = \{ab(c) \mid c \in COMPS\}$ and $L_K = \emptyset$, we say that p is a *purely diagnostic plan* of SD.
- if $L_W = \{ab(c)\}$ for some $c \in COMPS$, we say that p is a *discriminating diagnostic plan* of SD for c.

- is a *repair plan* of SD wrt. (L_M, L_W, L_K) if $\{\neg ab(c) \mid c \in COMPS\} \subseteq L_K$.

  If $L_K = \{\neg ab(c) \mid c \in COMPS\}$ and $L_M = \emptyset$, p is called a *purely repair plan*.

The main difference between a repair plan and a diagnostic plan lies in the requirement that a diagnostic plan maintains the abnormality status of the system components and a repair plan is allowed to change it. It is worth mentioning that to be able to construct a repair plan, we will need at our disposal actions that can change the status of the system components, i.e., the domain speficification has to contain repair actions whose effects are of the form $\neg ab(c)$. For example, replacing a broken bulb with a new one will restore $\neg ab(bulb)$. When repair actions are present, a conditional plan achieving a subset of $\{\neg ab(c) \mid c \in COMPS\}$ is a repair plan.

We illustrate this definition by considering diagnostic plans for the systems SD_1, SD_2, and SD_3 in the next example.

Example 42 Since SD_1 and SD_2 have only one diagnosis, the empty plan (i.e. []) is a purely diagnostic plan for SD_1 and SD_2. We will prove that the empty plan is not a purely diagnostic plan for SD_3.

Recall that the only observable fluent in SD_3 is *light_on*.

It follows from Example 41 that there are two possible current states:

$$s_1 = \{connected(bulb, switch), ab(bulb)\} \quad \text{and} \quad s_2 = \{connected(bulb, switch), ab(switch)\}.$$

Let $\Sigma = \{s_1, s_2\}$ and $L_W = \{ab(switch), ab(bulb)\}$. Since $\widehat{\Phi}([], \langle s_1, \Sigma\rangle) = \{\langle s_1, \Sigma\rangle, \langle s_2, \Sigma\rangle\}$ and $s_1 \not\sim_{L_W} s_2$, we conclude that the empty plan is not a purely diagnostic plan for SD_3.

We will now show that the plan

$$c = turn\_off \circ disconnect(bulb, switch) \circ connect(bulb, test\_device)$$

is a purely diagnostic plan for SD_3.

The trajectories of c with respect to the two possible states are given in the above picture. We have that

$$\widehat{\Phi}(c, \langle s_1, \Sigma\rangle) = \langle s_1', \{s_1'\}\rangle \quad \text{and} \quad \widehat{\Phi}(c, \langle s_2, \Sigma\rangle) = \langle s_2', \{s_2'\}\rangle$$

where

$$s_1' = \{connected(bulb, test\_device), ab(bulb)\} \quad \text{and} \quad s_2' = \{connected(bulb, test\_device), ab(switch), light\_on\}$$

In both cases, the values of $ab(bulb)$ and $ab(switch)$ do not change while c is executed. Furthermore, we have that $s_1 \sim_{L_W} s_1'$ and $s_2 \sim_{L_W} s_2'$. This implies that c is a purely diagnostic plan of SD_3. □

In the next example, we consider a more complicated diagnostic plan that involves sensing actions and conditional plans.

Example 43 (Electro-magnetic Door) Consider an electro-magnetic control door. The door has two led-indicators (*red* and *yellow*) To enter, an agent needs to put its electro-magnetic card, containing its id-number and password, into the slot connected to the door's controller. The door will open only if the card is valid, the id-number and the password are not corrupted, and the door is not malfunctioning. While the card is in the slot, if it is invalid, the red led will be on; and if the id-number or the password is corrupted or the door is defective, the yellow led will be on. The yellow led is on only if the red led is not. In this case, pushing the button "message" will print out a message. Reading it, the agent will know whether the door is defective or its card is unreadable.

Our agent, Jack, comes to work, and as usual, puts his card into the slot. The door does not open. What is wrong?

The story can be represented by the system $SD_4 = (SD_4, \{card, door, id\_pwd\}, \Gamma_4)$ as follows.

The actions of the domain specification SD_4 are: *insert_card*, *push_button*, *take_out_card*, *look* (look at the leds), or *read_msg* (read the message).

The fluents of SD_4 are: *ab(card)*, *ab(door)*, *ab(id_pwd)*, *has_card, card_in_slot*, *open, has_msg, red*, and *nolight, red, yellow*. The three fluents *red*, *yellow*, or *nolight* indicate that the red, yellow led, or none of them is on, respectively.

SD_4 comprises the following axioms:

$$
\begin{array}{l}
\mathit{insert\_card} \textbf{ causes } \mathit{open} \textbf{ if } \neg \mathit{ab}(\mathit{card}), \neg \mathit{ab}(\mathit{door}), \neg \mathit{ab}(\mathit{id\_pwd}) \\
\mathit{insert\_card} \textbf{ causes } \neg \mathit{has\_card} \wedge \mathit{card\_in\_slot} \\
\mathit{push\_button} \textbf{ causes } \mathit{has\_msg} \\
\mathit{take\_out\_card} \textbf{ causes } \mathit{has\_card} \wedge \neg \mathit{card\_in\_slot} \textbf{ if } \mathit{card\_in\_slot} \\
\mathit{look} \textbf{ determines } \mathit{red}, \mathit{yellow}, \mathit{nolight} \\
\mathit{read\_msg} \textbf{ determines } \mathit{ab}(\mathit{id\_pwd}), \mathit{ab}(\mathit{door}) \\
\textbf{executable } \mathit{insert\_card} \textbf{ if } \mathit{has\_card} \\
\textbf{executable } \mathit{push\_button} \textbf{ if } \mathit{yellow} \\
\textbf{executable } \mathit{read\_msg} \textbf{ if } \mathit{has\_msg} \\
\mathit{ab}(\mathit{card}) \wedge \mathit{card\_in\_slot} \textbf{ s\_causes } \mathit{red} \\
\mathit{yellow} \wedge \mathit{card\_in\_slot} \textbf{ s\_causes } \neg \mathit{red} \\
(\mathit{ab}(\mathit{id\_pwd}) \vee \mathit{ab}(\mathit{door})) \wedge \neg \mathit{red} \wedge \mathit{card\_in\_slot} \textbf{ s\_causes } \mathit{yellow}
\end{array}
$$

and the set of miracle actions

$$
\begin{array}{l}
\mathit{break}(\mathit{card}) \textbf{ causes } \mathit{ab}(\mathit{card}) \\
\mathit{break}(\mathit{door}) \textbf{ causes } \mathit{ab}(\mathit{door}) \\
\mathit{break}(\mathit{id\_pwd}) \textbf{ causes } \mathit{ab}(\mathit{id\_pwd})
\end{array}
$$

The set Γ_4 consists of the following observations,

$$
\begin{array}{l}
\neg \mathit{red} \wedge \neg \mathit{yellow} \wedge \neg \mathit{open} \wedge \mathit{has\_card} \wedge \neg \mathit{has\_msg} \wedge \neg \mathit{card\_in\_slot} \textbf{ at } \mathtt{s}_0 \\
\mathit{insert\_card} \textbf{ between } \mathtt{s}_0, \mathtt{s}_1 \\
\neg \mathit{open} \textbf{ at } \mathtt{s}_1 \\
\mathtt{s}_0 \textbf{ preceeds } \mathtt{s}_1
\end{array}
$$

The first observation describes the first observable situation, s_0. The second observation states that Jack puts his card into the slot, while the third states that the door is not open after Jack puts his card into the slot.

Intuitively, when Jack observes that the door does not open as the result of putting his card into the slot, he should realize that at least one of the three components: the card, the door, or the information on the card is no longer valid. Our diagnostic reasoning system does likewise. Indeed, the narrative

$$(SD_4 \setminus SD_{ab}, \Gamma_4 \cup \{\neg ab(c) \textbf{ at } \mathsf{s}_0 \mid c \in COMPS\})$$

does not have a model and there are three diagnoses for SD_4: $\Delta_1 = \{ab(id\_pwd)\}$, $\Delta_2 = \{ab(door)\}$, and $\Delta_3 = \{ab(card)\}$ which correspond to the models M_1, M_2, and M_3 of

$$\begin{array}{l}(SD_4, \Gamma_4 \cup \{\neg ab(card) \textbf{ at } \mathsf{s}_0,\ \neg ab(door) \textbf{ at } \mathsf{s}_0,\ ab(id\_pwd) \textbf{ at } \mathsf{s}_0\}),\\ (SD_4, \Gamma_4 \cup \{\neg ab(card) \textbf{ at } \mathsf{s}_0,\ ab(door) \textbf{ at } \mathsf{s}_0,\ \neg ab(id\_pwd) \textbf{ at } \mathsf{s}_0\}), \text{ and}\\ (SD_4, \Gamma_4 \cup \{ab(card) \textbf{ at } \mathsf{s}_0,\ \neg ab(door) \textbf{ at } \mathsf{s}_0,\ \neg ab(id\_pwd) \textbf{ at } \mathsf{s}_0\})\end{array}$$

defined as follows. $M_1 = (\Psi_1, \Sigma_1)$, $M_2 = (\Psi_2, \Sigma_2)$, and $M_3 = (\Psi_3, \Sigma_3)$, where $\Psi_1([]) = s_0 \cup \{ab(id\_pwd)\}$, $\Psi_2([]) = s_0 \cup \{ab(door)\}$, $\Psi_3([]) = s_0 \cup \{ab(card)\}$,

- $\Sigma_1(\mathsf{s}_0) = []$, $\Sigma_1(\mathsf{s}_1) = \Sigma_1(\mathsf{s}_c) = insert\_card$,
- $\Sigma_2(\mathsf{s}_0) = []$, $\Sigma_2(\mathsf{s}_1) = \Sigma_2(\mathsf{s}_c) = insert\_card$,
- $\Sigma_3(\mathsf{s}_0) = []$, $\Sigma_3(\mathsf{s}_1) = \Sigma_3(\mathsf{s}_c) = insert\_card$,

with $s_0 = \{has\_card, nolight, \neg red, \neg yellow, \neg open, \neg card\_in\_slot, \neg has\_msg\}$,

- $\Psi_1(insert\_card) = \{card\_in\_slot, ab(id\_pwd), yellow, \neg has\_card, \neg nolight, \neg red, \neg open,$
  $\neg ab(card), \neg ab(door)\} = s_1$,
- $\Psi_2(insert\_card) = \{card\_in\_slot, ab(door), yellow, \neg has\_card, \neg nolight, \neg red, \neg open,$
  $\neg ab(card), \neg ab(id\_pwd)\} = s_2$,
- $\Psi_3(insert\_card) = \{card\_in\_slot, ab(card), red, \neg has\_card, \neg nolight, \neg yellow, \neg open,$
  $\neg ab(card), \neg ab(door)\} = s_3$.

To narrow the list of the possible diagnoses of the system, Jack can find out the status of the LEDs. If the red led is on, he knows for sure that the card is no longer valid. Otherwise, the yellow led must be on. In that case, he can get the message and read it to know if the door is broken or the information on the card is corrupted. This process is captured by the following plan.

$$c = [look \circ \textbf{cases}(red \rightarrow [\,], yellow \rightarrow push\_button \circ read\_msg, nolight \rightarrow [\,])]$$

We will now show that c is a purely diagnostic plan for SD_4.

Let $\Sigma = \{s_1, s_2, s_3\}$. There are three possible current situations of SD_4: $\sigma_1 = \langle s_1, \Sigma\rangle$, $\sigma_2 = \langle s_2, \Sigma\rangle$, and $\sigma_3 = \langle s_3, \Sigma\rangle$. Let $s_i' = s_i \cup \{has\_msg\} \setminus \{\neg has\_msg\}$, $i = 1, 2$, then

- $\widehat{\Phi}(c, \sigma_1) = \widehat{\Phi}(push\_button \circ read\_msg, \langle s_1, \{s_1, s_2\}\rangle) = \widehat{\Phi}(read\_msg, \langle s_1', \{s_1', s_2'\}\rangle) = \{\langle s_1', \{s_1'\}\rangle\}$;
- $\widehat{\Phi}(c, \sigma_2) = \widehat{\Phi}(push\_button \circ read\_msg, \langle s_2, \{s_1, s_2\}\rangle) = \widehat{\Phi}(read\_msg, \langle s_2', \{s_1', s_2'\}\rangle) = \{\langle s_2', \{s_2'\}\rangle\}$;

- $\widehat{\Phi}(c, \sigma_3) = \{\langle s_3, \{s_3\}\rangle\}$.

The above computations also represent all trajectories of c wrt σ_1, σ_2, and σ_3. Obviously, $\widehat{\Phi}(c, \sigma_i) \neq \perp$ for $i = 1, 2, 3$. Furthermore, it is easy to check that the values of ab-literals remain unchanged on all trajectories and in each c-state $\langle s', \Sigma'\rangle$ belonging to $\widehat{\Phi}(c, \sigma_i)$ and $s'' \in \Sigma'$, $s' \sim_{L_W} s''$ where $L_W = \{ab(card), ab(id\_pwd), ab(door)\}$. For example, $\langle s'_1, \{s'_1\}\rangle$ is the only c-state in $\widehat{\Phi}(c, \sigma_1)$, and trivially, $s'_1 \sim_{L_W} s'_1$. Thus, c is a purely diagnostic plan for SD_4. □

4.7 Bibliographical Notes

- sensing actions in planning
- sensing actions in rac (previous work)
- sensing actions in domains with multi-valued fluents
- different approximations present in [SB01a]

Thus sensing actions are very important for planning in presence of incomplete information. In the past, sensing actions have been formalized in [Moo79, Moo85, SL93, Haa86, LTM97] and planning in presence of incomplete information has been studied in [EHW+92, PS92, PC96, GB94, GEW96, GW96, Lev96, KOG92, SW98, WAS98, GB96]. To motivate our work we now briefly review the earlier formalizations of sensing actions.

Sensing actions were first formalized by Moore in his dissertation [Moo79] and in some of his later papers; for example, [Moo85].

Moore uses possible world semantics to represent knowledge and treats the accessibility relation between worlds as a fluent when reasoning about sensing and non-sensing actions.

Scherl and Levesque [SL93] adapted Moore's formulation to situation calculus and proved several important results about their formulation such as: knowledge-producing actions do not affect fluents other than the knowledge fluent; and that actions that are not knowledge-producing only affect the knowledge fluent as appropriate. They also showed how regression can be applied to knowledge-producing actions.

Notice that our definition of the transition function Φ does not stipulate any special requirement on how the Φ function is defined. Thus, any action description language [BG97, KL94, Tur97] with a semantics depending on a state transition function like *Res* can be extended to allow sensing actions. Therefore, several of the other features of action description languages such as multi-valued fluents [GKL97], ramification [LR94b, KL94], causality [Lin95, MT95, Bar95], concurrent actions [LS92, BG93, BG97], can be directly added to our framework. For example, to extend our formulation to multi-valued fluents, we have to: (i) extend our propositions to be able to denote different values of the fluents, and (ii) extend our notion of states to be interpretations of the fluents. The definition of transition function will remain the same, except that the notion of s and s' agreeing on a fluent f would now mean that s and s' have the same value of f. To keep our focus on the main issue of formalizing sensing actions, we do not include these features in our formulation, as they can be directly added when desired.

Chapter 5

Reasoning About Actions in a Probabilistic Setting

Static causal axioms and formulas enable us to represent and reason with nondeterministic actions (Chapter 2). They can also be used in modeling actions with probabilistic effects.

Example 44 Consider the statement:

Tossing the coin C causes head with the probability of 2/3.

Let h_1, h_2 and t be three fluents represent *head, head* and *tail* respectively. The statement can be represented using the following axioms:

$$\begin{aligned} toss \quad & \textbf{causes} \quad tossed \\ tossed \quad & \textbf{s\_causes} \quad h_1 \vee h_2 \vee h_3 \\ h_1 \vee h_2 \quad & \textbf{s\_causes} \quad head \\ t \quad & \textbf{s\_causes} \quad tail \\ & head \oplus tail \\ & h_1 \oplus h_2 \oplus t \end{aligned}$$

The last two statical axioms state that *head* and *tail* (respectively, h_1, h_2, and t) are mutual exclusive. Let s be an arbitrary state of this domain. Let

$$\begin{aligned} s_1 &= \{tossed, h_1, \neg h_2, \neg t, head, \neg tail\}, \\ s_2 &= \{tossed, \neg h_1, h_2, \neg t, head, \neg tail\}, \text{ and} \\ s_3 &= \{tossed, \neg h_1, \neg h_2, t, tail, head\}. \end{aligned}$$

We can show that

$$\Phi(toss, s) = \{s_1, s_2, s_3\}$$

Notice that among the three states resulting from the execution of the action *toss*, two contain *head*. As this holds for every state, we can say that performing *toss* causes *head* to appear withthe probability of 2/3. □

The above exampe shows that it is feasilbe to model actions with probabilistic effects using formulas and statica causal axioms, the representation requires additional symbols and is somtimes cumbersome[1].

[1] It is clear that an effect with irrational probability cannot be represented using the method described in Example 44.

In this chapter, we will develop a language for representing and reasoning with actions with probabilitic effects. The language, called $\mathcal{B}_\mathcal{P}$, is an extension of the language $\mathcal{B}$. Naturally, $\mathcal{B}_\mathcal{P}$ needs to allow us to specify the probabily of effects of actions and to define the transitions between states when actions are executed. Following our approach, the following issues need to be addressed:

1. How to represent probabilistic effects of actions?
2. What is a state?
3. How are the transitions between states defined?

To represent probabilistic effects of actions, we will add a new sort called *variables* to $\mathcal{B}$. Variables are used to encode the probability associated to the effects of actions and the unpredictable nature of actions in the probabilistic settings. We tackle each of the above issues in a separate section.

5.1 Syntax of $\mathcal{B}_\mathcal{P}$

We extend the syntax of $\mathcal{B}$ with two additional non-empty disjoin sets of symbols $\mathbf{U}_N$ and $\mathbf{U}_I$. These two sets are also disjoin from the set of fluents $\mathbf{F}$ and actions $\mathbf{A}$. Members of $\mathbf{U}_N$ and $\mathbf{U}_I$ are referred to as *non-inertial variables* and *inertial unknown variables* respectively. Like fluent literals, we define unknown variable literals as either an unknown variable v or its negation, denoted by $\neg v$. In $\mathcal{B}_\mathcal{P}$, a *literal* is either a fluent literal or an unknown variable literal. A *formula* is a propositional formula constructed from literals. $\mathcal{B}_\mathcal{P}$ has two components, a domain specification language and a proability specification component. The domain specification language of $\mathcal{B}_\mathcal{P}$ also have three types of axioms: fluent effect axioms, static causal axioms, and executability axiom. They are very similar to axioms in $\mathcal{B}$ and are of the following form:

$$a \textbf{ causes } \psi \textbf{ if } \varphi \tag{5.1}$$

$$\theta \textbf{ causes } \psi \tag{5.2}$$

$$\textbf{executable } a \textbf{ if } \varphi \tag{5.3}$$

where a is an action, ψ is a *fluent* formula, θ is a formula of fluents and *inertial* unknown variables, and φ is a formula of fluents and *unknown* variables. Similar to $\mathcal{B}$, axioms of the form (5.1) describe the direct effects of actions on the world and are called *fluent effect axiom*. Axioms of the form (5.2), called *static causal axioms*, describe causal relationship between fluents and unknown variables. Finally, axioms of the form (5.3), called *executability conditions*, state when actions are executable. The main difference between axioms in $\mathcal{B}$ and $\mathcal{B}_\mathcal{P}$ is the presence of the unknown variables in $\mathcal{B}_\mathcal{P}$'s axioms.

To specify the probability distribution associated to a unknown variable, we use axiom of the following form:

$$\textbf{probability of } u \textbf{ is } p \tag{5.4}$$

where u is an unknown variable, and p is a real number between 0 and 1.

Observe that the specification of the axioms stick to the following principles:

- The values of unknown variables are not affected by actions;
- The values of unknown variables are independent on the fluents;

- The effect of an action on a fluent may depend on unknown variables; and
- Only inertial unknown variables may have direct effects on the values of fluents, i.e., only inertial unknown variables can occur in θ of (5.2).

A domain specification in $\mathcal{B}_\mathcal{P}$ is a set of $\mathcal{B}_\mathcal{P}$ axioms. We illustrate the syntax of $\mathcal{B}_\mathcal{P}$ in a few examples.

Example 45 (Ball drawing) We draw a ball from an infinitely large black box. The color of the ball is either *red* or *blue* (or $\neg red$). The probability of drawing a red ball is said to be equal 0.5.

This domain can be represented by a domain specification with one action *draw*, one fluent *red*, and an unknown variable u which affects the outcome. The axioms are:

$$D_{ball} = \begin{cases} draw \textbf{ causes } red \textbf{ if } u \\ draw \textbf{ causes } \neg red \textbf{ if } \neg u \\ \textbf{probability of } u \textbf{ is } 0.5 \end{cases}$$

Intuitively, the first fluent effect axiom states that we draw a *red* ball with the probability u; the second one is the converse: we draw a $\neg red$ ball with the probability $1 - u$; the last axiom says that $u = 0.5$. For now, we do not need to declare whether or not u is a inertial or non-inertial unknown variable. We will do so whenever this distinction is needed. □

Example 46 (The Yale Shooting with Probability) Let us assume that the effects of two actions *load* and *shoot* in the Yale shooting Example 1 are probabilistic: *load* successes with a probability of u_1 while *shoot* does with a probability of u_2. The $\mathcal{B}_\mathcal{P}$ representation of the story is given as follows.

$$D_{shoot} = \begin{cases} shoot \textbf{ causes } \neg alive \textbf{ if } loaded, u_1 \\ load \textbf{ causes } loaded \textbf{ if } u_2 \\ \textbf{probability of } u_1 \textbf{ is } p_1 \\ \textbf{probability of } u_2 \textbf{ is } p_2 \end{cases}$$

where $0 \leq p1, p_2 \leq 1$. Again, we do not need to specify whether or not u_1 (resp. u_2) is inertial or non-inertial unknown variable. □

5.2 Transitions in $\mathcal{B}_\mathcal{P}$

The semantics of $\mathcal{B}_\mathcal{P}$ is defined similar to the semantics of $\mathcal{B}$, by specifying a transitiion function which maps pairs of actions and states into a set of states. In fact, if we were to consider random/unknown variables as fluents, the transition function of $\mathcal{B}_\mathcal{P}$ can be specified by Definition 9.

Let D be a domain specification in $\mathcal{B}_\mathcal{P}$. A set of literals σ in D can contain fluent literals and/or unknown variable literals. A *state* $\mathbf{s}$ of D is a complete set of literals[2] (fluent and unknown variable literals) that is closed under the set of static causal axioms of D. Each state can be viewed as a pair of a *n-state* $\mathbf{s}_F$ and an *u-state* $\mathbf{s}_u$, i.e., $\mathbf{s} = \mathbf{s}_F \cup \mathbf{s}_u$, where $\mathbf{s}_F$ and $\mathbf{s}_u$ are the set of fluent literals and unknown variable literals in $\mathbf{s}$, respectively. By $\mathbf{s}_I$ and $\mathbf{s}_N$, we denote the set or inertial unknown literals and and non-inertial unknown literals in $\mathbf{s}$, i.e., $\mathbf{s}_u = \mathbf{s}_I \cup \mathbf{s}_N$. For later use, let $\mathbf{s}_{F,I} = \mathbf{s}_F \cup \mathbf{s}_I$. For any u-state u, $\mathcal{S}(u)$ denotes the set of states $\mathbf{s}$, such that $\mathbf{s}_u = u$. For a state $\mathbf{s}$ and a n-state s, $\mathbf{s} \models s$ if $\mathbf{s}_F = s$, i.e., if the interpretation of fluents in $\mathbf{s}$ is same as in $\mathbf{s}$.

[2]We use a different type setting for states to highlight the different between states in $\mathcal{B}_\mathcal{P}$ and $\mathcal{B}$: $\mathbf{s}$ and s respectively.

Given a domain specification D and a state $\mathbf{s}$, the truth value of a formula φ with respect to $\mathbf{s}$ is defined as usual. The executability of an action a in a state $\mathbf{s}$ and the direct effect of a in $\mathbf{s}$ are defined as in Chapter 2:

- An action a is *possibly executable* in $\mathbf{s}$ if there exists an executability condition

  $$\textbf{executable } a \textbf{ if } \varphi$$

  in D such that φ holds in $\mathbf{s}$. The direct effects of a in $\mathbf{s}$ is defined by

- The direct effect of a in $\mathbf{s}$

  $$e_D(a, \mathbf{s}) = \{\psi \mid a \textbf{ causes } \psi \textbf{ if } \varphi,\ \mathbf{s} \models \varphi\}.$$

We will next define a transition function Φ that specifies the transitions between states of D. Φ needs to satisfy the following properties:

- it behaves exactly as a transition function for $\mathcal{B}$ domains (Definition 9) if axioms of the form (5.4) were not present in D; and
- it does not change the values of inertial unknown variables of D.

The first property is obvious since the set of axioms of the form (5.1)-(5.3) in D could be viewed as a domain specification in $\mathcal{B}$. The second property stems from the fact that inertial unknown variables cannot be controlled by the agent's actions. This leads to the following definition:

Definition 42 Let D be a domain specification in $\mathcal{B}_{\mathcal{P}}$, a be an action, and $\mathbf{s}$ be a state. A function Φ that maps pairs of actions and states into sets of states is a *transition function* of D if

$$\Phi(a, \mathbf{s}) = \begin{cases} \{\mathbf{s}' \mid \mathbf{s}' \text{ is a state and } \mathbf{s}'_{F,I} = Cl_D(e_D(a, \mathbf{s}) \cup (\mathbf{s}_{F,I} \cap \mathbf{s}'_{F,I}))\} & \text{if } a \text{ is possibly executable in } \mathbf{s} \\ \emptyset & \text{otherwise} \end{cases}$$

The following properties can be proved.

Proposition 3 For a $\mathcal{B}_{\mathcal{P}}$ domain specification D,

1. D posses a unique transition function;
2. For every $\mathbf{s}' \in \Phi(a, \mathbf{s})$, $\mathbf{s}'_I = \mathbf{s}_I$.
3. For every $\mathbf{s}' \in \Phi(a, \mathbf{s})$ and for every interpretation w of U_N, we have that $(\mathbf{s}'_{F,I} \cup w) \in \Phi(a, \mathbf{s})$.

Since every domain specification D has a unique transition function Φ, we will write Φ_D to denote the transition function of D. Φ_D is extended to define the function $\widehat{\Phi}_D$, which determines the set of states reached after the execution of a sequence of actons $[a_1 \circ a_2 \ldots \circ a_n]$ in a state $\mathbf{s}$, as in Definition 4. We write

$$(D, \mathbf{s}) \models_{\mathcal{B}_{\mathcal{P}}} \varphi \textbf{ after } [a_1 \circ \ldots \circ a_n]$$

if $\widehat{\Phi}_D([a_1 \circ \ldots \circ a_n], \mathbf{s})$ is not empty and φ is true in all states belonging to it.

5.3 Probabilistic Transitions in $\mathcal{B}_{\mathcal{P}}$

So far, we have not taken into consideration the probabilistic specification of $\mathcal{B}_{\mathcal{P}}$ domains. Given a domain specification D, what is the probability of a state $\mathbf{s}$ in D? What is the probability of a state $\mathbf{s}'$ after the execution of an action sequence $[a_1 \circ \ldots \circ a_n]$ in a state $\mathbf{s}$? This section provides answers for these questions. We begin with a formula of the joint probability distribution of the unknown variables. Since we assume that the values of the unknown variables are independent of each other, their joint probability distribution is given by:

$$P(u_1, \ldots, u_n) = P(u_1) \times \ldots \times P(u_n) \tag{5.5}$$

where $P(u_1)$ is a short hand for $P(U_1 = true)$. [3]

Since several states may have the same interpretation of the unknown variables and we do not have any unconditional preference of one state over another, the unconditional probability of the various states can now be defined as:

$$P(\mathbf{s}) = \frac{P(\mathbf{s}_u)}{|\mathcal{S}(\mathbf{s}_u)|} \tag{5.6}$$

Example 47 For the domain specification D_{ball}, we have that $\mathbf{F} = \{red\}$. Let us assume that $\mathbf{U}_N = \{u\}$ and $\mathbf{U}_I = \emptyset$, i.e., u is a non-inertial unknown variable. Consider $\mathbf{s} = \{red, u\}$, we have that $\mathbf{s}_u = \{u\}$. Thus, $P(\mathbf{s}_u) = P(u) = 0.5$. Furthermore, $S(\mathbf{s}_u) = \{\{red, u\}, \{\neg red, u\}\}$. Hence,

$$P(\{red, u\}) = \frac{0.5}{2} = \frac{1}{4}$$

□

The probability of a state after the execution of a sequence of actions is defined in several steps. First we define the transitional probability between states due to a single action. Since inertial variables do not change their value from one state to the next, $P_a(\mathbf{s}'|\mathbf{s})$ will depend only on the conditioning of fluents and non-inertial variables. Thus,

$$P_a(\mathbf{s}'|\mathbf{s}) = P_a(\mathbf{s}'_{F,N}|\mathbf{s}).$$

Since non-inertial variables are independent from the transition, we have

$$P_a(\mathbf{s}'_{F,N}|\mathbf{s}) = P_a(\mathbf{s}'_F|\mathbf{s}) * P(\mathbf{s}'_N).$$

Since there is no distribution associated with fluents, we assume that $P_a(\mathbf{s}'_F|\mathbf{s})$ is uniformly distributed. Then

$$P_a(\mathbf{s}'_F|\mathbf{s}) = \frac{|\Phi(a, \mathbf{s})|}{2^{|U_N|}},$$

because there are $\frac{|\Phi(a,\mathbf{s})|}{2^{|U_N|}}$ possible next states that share the same interpretation of unknown variables. Since our transition function maps an action and a state to a set of states without any preference among the states, we have the following.

$$P_{[a]}(\mathbf{s}'|\mathbf{s}) = P_a(\mathbf{s}'|\mathbf{s}) = \begin{cases} \frac{|\Phi(a,\mathbf{s})|}{2^{|U_N|}} P(\mathbf{s}'_N) & \text{if } \mathbf{s}' \in \Phi(a, \mathbf{s}); \\ 0 & \text{otherwise .} \end{cases} \tag{5.7}$$

[3] If we have multi-valued unknown variables then $P(u_1)$ will be a short hand for $P(U_1 = u_1)$.

Remark 7 At this point, it is instructive to point out that the probability $P_a(\mathbf{s}'|\mathbf{s})$ depends on the set of non-inertial unknown variables.

This allows us to define the transitional probability due to a sequence of actions as follows.

$$P_{[\ ]}(\mathbf{s}'|\mathbf{s}) = 1 \text{ if } \mathbf{s} = \mathbf{s}';\ \text{otherwise it is 0.} \tag{5.8}$$

$$P_{[a_1 \circ \ldots \circ a_n]}(\mathbf{s}'|\mathbf{s}) = \sum_{\mathbf{s}''} P_{[a_1 \circ \ldots \circ a_{n-1}]}(\mathbf{s}''|\mathbf{s}) P_{a_n}(\mathbf{s}'|\mathbf{s}'') \tag{5.9}$$

Example 48 Let us continue with Example 47. Let $\mathbf{s}' = \{red, u\}$. We would like to compute

$$P_{draw}(\mathbf{s}'|\mathbf{s})$$

in two situations.

- *u is a non-inertial unknown variable*. So, $\mathbf{U}_N = \{u\}$. First, it is easy to check that

  $$\Phi(draw, \mathbf{s}) = \{\{red, u\}\}$$

  By Equation (5.7),

  $$P_{draw}(\mathbf{s}'|\mathbf{s}) = \frac{1}{2} \times P(\mathbf{s}'_N) = \frac{1}{2} \times P(u) = \frac{1}{4}$$

- *u is an inertial unknown variable*. In this case, $\mathbf{U}_N = \emptyset$. Notice that $P(\mathbf{s})$ does not change it value (according to Equation (5.6)). Since u is now an inertial unknown variable, we have that $\mathbf{U}_N = \emptyset$. Equation (5.7) gives

  $$P_{draw}(\mathbf{s}'|\mathbf{s}) = 1$$

□

The above formulas allow us to define the (probabilistic) correctness of an action sequence α given that we are in a particular state $\mathbf{s}$ as follows:

$$P(\varphi \textbf{ after } \alpha \mid \mathbf{s}) = \sum_{\mathbf{s}' \in \widehat{\Phi}([\alpha], \mathbf{s}) \wedge \mathbf{s}' \models \varphi} P_{[\alpha]}(\mathbf{s}'|\mathbf{s}) \tag{5.10}$$

Equation (5.10) allows us to answer queries of the form:

$$\textbf{probability of } \varphi \textbf{ after } [a_1 \circ \ldots \circ a_n] \textbf{ is } p \tag{5.11}$$

where φ is a formula of fluents and unknown variables, a_i's are actions, and p is a real number between 0 and 1. Two special cases:

- φ **after** $[a_1 \circ \ldots \circ a_n]$ implies that $p = 1$, and
- $\neg\varphi$ **after** $[a_1 \circ \ldots \circ a_n]$ implies that $p = 0$.

Given a state $\mathbf{s}$ and a domain specification D in $\mathcal{B}_{\mathcal{P}}$, we say that

$$(D, \mathbf{s}) \models_{\mathcal{B}_{\mathcal{P}}} \textbf{ probability of } \varphi \textbf{ after } [a_1 \circ \ldots \circ a_n] \textbf{ is } p$$

if $P(\varphi \textbf{ after } \alpha \mid \mathbf{s}) = p$.

Example 49 Continuing with Example 48, let $\mathbf{s} = \mathbf{s}' = \{red, u\}$ and $\alpha = [draw \circ draw]$. We have that

$$\widehat{\Phi}(\alpha, \mathbf{s}) = \{\{red, u\}\}$$

This and the results in Example 48 give us the following probability:

- If u is a non-inertial unknown variable, then

$$P_\alpha(\mathbf{s}'|\mathbf{s}) = \sum_{\mathbf{s}''} P_{draw}(\mathbf{s}''|s) P_{draw}(\mathbf{s}'|\mathbf{s}'') == \frac{1}{4} \times \frac{1}{4} = \frac{1}{16}$$

- If u is an inertial unknown variable, then

$$P_\alpha(\mathbf{s}'|\mathbf{s}) = \sum_{\mathbf{s}''} P_{draw}(\mathbf{s}''|s) P_{draw}(\mathbf{s}'|\mathbf{s}'') == 1 \times 1 = 1$$

□

5.4 Hypothetical Reasoning in $\mathcal{B}_\mathcal{P}$

The previous section presents our formulation for computing the probability of a formula after the execution of an action sequence in a given state. We will now extend it to compute this probability given a set of observations of the form

$$\psi \ \textbf{obs\_after} \ \ [a_1 \circ \ldots \circ a_n] \tag{5.12}$$

where ψ is a fluent formula, and a_i's are actions. Intuitively, this observation states that ψ is observed after the action sequence $a_1 \circ \ldots \circ a_n$ is executed from the current state. When, $n = 0$, we simply write **initially** ψ. In this case, they are similar to observations in (2.3) in that they describe the initial state(s) of the world. When $n > 0$, an observation of the form (5.12) states that, hypothetically, ψ is observed to be true after the execution of the sequence $[a_1 \circ \ldots \circ a_n]$ in the initial state. Thus, this type of observations is more general than the one discussed in Chapter 2 (as in (2.3)). Observations of the form (5.12) are also different than the observations in a narrative as it is only hypothetical. In a later section, we will discuss the language for probabilistic reasoning in narratives. A *probabilistic action theory* is a pair (D, O) where D is a domain specification consisting of axioms of the form (5.1)-(5.3) and (5.4) and O is a set of observations of the form (5.12).

Since the probability $P(\varphi \ \textbf{obs\_after} \ \ \alpha \mid \mathbf{s})$ can be computed by Equation (5.10), we can use the Bayes' rule to define the conditional probability of a state given that a set of observations O by:

$$P(\mathbf{s}_i \mid O) = \begin{cases} \frac{P(O|\mathbf{s}_i)P(\mathbf{s}_i)}{\sum_{\mathbf{s}_j} P(O|\mathbf{s}_j)P(\mathbf{s}_j)} & \text{if } \sum_{\mathbf{s}_j} P(O|\mathbf{s}_j)P(\mathbf{s}_j) \neq 0 \\ 0 & \text{otherwise .} \end{cases} \tag{5.13}$$

This gives us the following equation:

$$P(\varphi \ \textbf{after} \ \alpha \mid O) = \sum_s P(\mathbf{s}|O) \times P(\varphi \ \textbf{after} \ \alpha \mid \mathbf{s}) \tag{5.14}$$

Using the above formula, we now define the entailment between a theory (consisting of a domain description and an set of observations) and queries :

Definition 43 Let (D, O) be a probabilistic action theory and Q be a query of the form (5.11). (D, O) entails Q, denoted by

$$(D, O) \models_{\mathcal{B}_\mathcal{P}} \textbf{ probability of } [\varphi \textbf{ after } a_1 \circ \ldots \circ a_n] \textbf{ is } p$$

iff

$$P(\varphi \textbf{ after } [a_1 \circ \ldots \circ a_n] \mid O) = p$$

Remark 8 Since our observations are hypothetical and are about a particular hypothetical execution, it is possible that $P(\varphi \textbf{ after } \alpha \mid \varphi \textbf{ obs\_after } \alpha) < 1$, when α contains some non-deterministic actions. Although it may appear unintuitive in the first glance, it is reasonable as just because a particular run of α makes φ true does not imply that *all* run of α would make φ true.

Example 50 (Ball Drawing, Cont.) Let us consider the domain specification D_{ball} from Example 45. Let

$$O = \{red \textbf{ obs\_after } draw\}$$

and

$$Q = red \textbf{ after } [draw \circ draw]$$

As in the previous examples, we will compute the values of $p = P(Q|O)$ under different assumptions about the variable u and about the color of the balls in the box.

Let $\mathbf{s}_1 = \{red, u\}$, $\mathbf{s}_2 = \{red, \neg u\}$, $\mathbf{s}_3 = \{\neg red, u\}$ and $\mathbf{s}_4 = \{\neg red, \neg u\}$.

1. Assume that the balls in the box are of the same color, i.e., the box contains either all red or all blue balls. Assume also that u is an inertial unknown variable.

   Since $\Phi(draw, \mathbf{s}_1) = \Phi(draw, \mathbf{s}_3) = \{\mathbf{s}_1\}$ and $\Phi(draw, \mathbf{s}_2) = \Phi(draw, \mathbf{s}_4) = \{\mathbf{s}_4\}$ and $\mathbf{U}_N = \emptyset$, we have that

   $$P_{draw}(\mathbf{s}_1|\mathbf{s}_1) = P_{draw}(\mathbf{s}_1|\mathbf{s}_3) = P_{draw}(\mathbf{s}_4|\mathbf{s}_4) = P_{draw}(\mathbf{s}_4|\mathbf{s}_2) = 1.$$

   By (5.13),

   $$P(O|\mathbf{s}_1) = \Sigma_{\mathbf{s}' \in \Phi(draw, \mathbf{s}_1),\, \mathbf{s}' \models red} P_{draw}(\mathbf{s}'|\mathbf{s}_1) = P_{draw}(\mathbf{s}_1|\mathbf{s}_1) = 1$$

   Similarly,

   $$P(O|\mathbf{s}_3) = 1 \quad \text{and} \quad P(O|\mathbf{s}_2) = P(O|\mathbf{s}_4) = 0$$

   This gives

   $$P(\mathbf{s}_1|O) = 0.5$$

   Furthermore,

   $$P(Q|\mathbf{s}_1) = \Sigma_{\mathbf{s}' \in \widehat{\Phi}([draw \circ draw], \mathbf{s}_1),\, \mathbf{s}' \models red} P_{draw \circ draw}(\mathbf{s}'|\mathbf{s}_1) = P_{draaw \circ draw}(\mathbf{s}_1|\mathbf{s}_1) = 1$$

   Similar computation yields $P(Q|\mathbf{s}_1) = 1$. Hence, we have that $P(Q|O) = 1$.

   Observe that under the mentioned assumptions, the initial observation tells us all about the future outcomes.

2. Assume that half of the balls in the box are red and the other half are blue. Assume also that u is a non-inertial unknown variable. We can show that $P(\mathbf{s}_1|O) = P(\mathbf{s}_3|O) = 0.5$ and $P(\mathbf{s}_2|O) = P(\mathbf{s}_4|O) = 0$. By (5.10), $P(Q|\mathbf{s}_j) = 0.5$ for $1 \leq j \leq 4$. By (5.14), $P(Q|O) = 0.5 \times \sum_{\mathbf{s}_j} P(\mathbf{s}_j|O) = 0.5$. Here, the observation O does not help in predicting the future. □

Example 51 (Yale Shooting, Cont.) Let us consider the domain specification D_{shoot} from Example 46. Assume that u_1 and u_2 are inertial unknown variables.

Now suppose we have the following observations

$$O_1 = \{\textbf{initially } alive,\ \textbf{initially } \neg loaded\}$$

We can now show that $(D_{shoot}, O_1) \models_{\mathcal{B}_\mathcal{P}}$ **probability of** $[alive \textbf{ after } load \circ shoot]$ **is** $1 - p_1 \times p_2$.

Let $l_1 \in \{u_1, \neg u_1\}$ and $l_2 \in \{u_2, \neg u_2\}$. Let $\mathbf{s}_{l_1,l_2}$ denote a state satisfying $\{l_1, l_2\} \subseteq \mathbf{s}_{l_1,l_2}$. Furthermore, let Q be the query $alive$ **after** $[load \circ shoot]$.

We can show that

$$\begin{array}{ll} P(\mathbf{s}_{u_1,u_2}) = \frac{p_1 p_2}{4} & P(\mathbf{s}_{u_1,\neg u_2}) = \frac{p_1(1-p_2)}{4} \\ P(\mathbf{s}_{\neg u_1,u_2}) = \frac{(1-p_1)p_2}{4} & P(\mathbf{s}_{\neg u_1,\neg u_2}) = \frac{(1-p_1)(1-p_2)}{4} \end{array}$$

Since $P_{[]}(\mathbf{s}'|\mathbf{s}) = 1$ iff $\mathbf{s}' = \mathbf{s}$, we have the following:

$$P(O_1|\mathbf{s}) = \begin{cases} 1 & \text{if } \{\neg loaded, alive\} \subseteq \mathbf{s} \\ \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

Let $\mathbf{s}_1 = \{\neg loaded, alive, u_1, u_2\}$, $\mathbf{s}_2 = \{\neg loaded, alive, u_1, \neg u_2\}$, $\mathbf{s}_3 = \{\neg loaded, alive, \neg u_1, u_2\}$, and $\mathbf{s}_4 = \{\neg loaded, alive, \neg u_1, \neg u_2\}$. Equation (5.15) gives the following value:

$$P(\mathbf{s}|O_1) = \begin{cases} 4 \times P(\mathbf{s}_i) & \text{if } \mathbf{s} = \mathbf{s}_i\ (i = 1, \ldots, 4) \\ \\ 0 & \text{if } \mathbf{s} \neq \mathbf{s}_i \end{cases} \tag{5.16}$$

Let $\mathbf{s}'_1 = \{loaded, alive, u_1, u_2\}$ and $\mathbf{s}'' = \{loaded, \neg alive, u_1, u_2\}$, we have that $\Phi(load, \mathbf{s}_1) = \{\mathbf{s}'_1\}$ and $\Phi(shoot, \mathbf{s}'_1) = \{\mathbf{s}''\}$. This implies that $P(Q|\mathbf{s}_1) = 0$.

On the other hand, $\Phi(load, \mathbf{s}_2) = \{\mathbf{s}_2\}$ and $\Phi(shoot, \mathbf{s}_2) = \{\mathbf{s}_2\}$, $P(Q|\mathbf{s}_2) = P_{load}(\mathbf{s}_2|\mathbf{s}_2) \times P_{shoot}(\mathbf{s}_2|\mathbf{s}_2)$. Equation (5.7) gives

$$P_{load}(\mathbf{s}_2|\mathbf{s}_2) = P_{shoot}(\mathbf{s}_2|\mathbf{s}_2) = 1.$$

Hence, $P(Q|\mathbf{s}_2) = 1$. Similarly, $P(Q|\mathbf{s}_3) = 1$ and $P(Q|\mathbf{s}_4) = 1$. This leads to the following

$$\begin{array}{lll} P(Q|O_1) = \Sigma_\mathbf{s} P(\mathbf{s}|O_1) \times P(Q|\mathbf{s}) & & \text{(Eq. (5.14))} \\ = \Sigma_{i=1}^4 P(\mathbf{s}_i|O_1) \times P(Q|\mathbf{s}_i) & & \text{(Eq. (5.16))} \\ = \Sigma_{i=1}^4 P(\mathbf{s}_i|O_1) \times P(Q|\mathbf{s}_i) & & \text{(Eq. (5.16))} \\ = p_1(1-p_2) + (1-p_1)p_2 + (1-p_1)(1-p_2) & & \text{(Eq. (5.16))} \\ = 1 - p_1 p_2 & & \end{array}$$

□

The next example, taken from [**?**], illustrates the uses of $\mathcal{B}_\mathcal{P}$ in formalizing counter-factual.

Example 52 (Counter Factual) The data obtained on a particular medical test where half the patients were treated and the other half were left untreated shows the following:

"Joe was treated and he died. Did Joe's death occur due to the treatment? In other words, would Joe have lived if he was not treated.

The story can be represented by two causal models, each resulting in different answers to the above question. They can be represented by two $\mathcal{B}_\mathcal{P}$ specifications as follows.

treated	true	true	false	false
alive	true	false	true	false
fraction	.25	.25	.25	.25

Table 5.1: Statistics

1. *Causal Model 1:* We have two actions, *treatment* and *no_treatment*, and an inertial unknown variable u. The domain specifications contains the following axioms:

$$D^1_{medical} = \begin{cases} \textit{treatment} \textbf{ causes } \textit{action\_occurred} \\ \textit{no\_treatment} \textbf{ causes } \textit{action\_occurred} \\ u \wedge \textit{action\_occurred} \textbf{ s\_causes } \neg \textit{alive} \\ \neg u \wedge \textit{action\_occurred} \textbf{ s\_causes } \textit{alive} \\ \textbf{probability of } u \textbf{ is } 0.5 \end{cases}$$

   The observation is given by

$$O^1 = \begin{cases} \textbf{initially } \neg \textit{action\_occurred} \\ \textbf{initially } \textit{alive} \\ \neg \textit{alive} \textbf{ obs\_after } \textit{treatment} \end{cases}$$

   Observe that $\mathcal{B}_{\mathcal{P}}$ does not provide the syntax for expressing the probability of an action occurrence. As this is not an important in the analysis that we are making, let us assume that the probability of the occurrence of *treatment* and *no_treatment* is 0.5 each.

   Assuming u is independent of the occurrence of *treatment* it is easy to see that the above specification agrees with the data in Table 5.1.

   It can be shown that

$$(D^1_{medical}, O^1) \not\models \textit{alive} \textbf{ after } \textit{no\_treatment}$$

   and

$$(D^1_{medical}, O^1) \models \neg \textit{alive} \textbf{ after } \textit{no\_treatment}$$

2. *Causal Model 2:* The second causal model can be expressed by the following domain specification:

$$D^2_{medical} = \begin{cases} \textit{treatment} \textbf{ causes } \neg \textit{alive} \textbf{ if } u \\ \textit{no\_treatment} \textbf{ causes } \neg \textit{alive} \textbf{ if } \neg u \\ \textbf{probability of } u \textbf{ is } 0.5 \end{cases}$$

   The observation is given by

$$O^2 = \{\textbf{initially } \textit{alive}, \ \neg \textit{alive} \textbf{ obs\_after } \textit{treatment}\}$$

   The probability of occurrence of *treatment* and *no_treatment* remains 0.5 each. Assuming that u is independent of the occurrence of *treatment*, it is easy to see that $D^2_{medical}$ represents the data in Table 5.1.

   We can now show that

$$(D^2_{medical}, O^2) \models \textit{alive} \textbf{ after } \textit{no\_treatment}$$

□

5.5 Reasoning about $\mathcal{B}_{\mathcal{P}}$ Narratives

Observations of the form (5.12) are hypothetical in the sense that the actions did not actual occur. In this section, we will consider observations of the form (3.1)-(3.4). In other words, we will consider $\mathcal{B}_{\mathcal{P}}$ narratives. As in Chapter 3, we define a narrative as a pair (D, Γ) of a domain specification D and a set of observations, which include fluent facts, precedence facts, and occurrence facts. The key difference is that D is a probabilistic domain specification.

Recall that a model of a narrative (D, Γ) is a pair (Ψ, Σ) satisfying Γ, where Ψ is a causal model of D and Σ is a situation assignment of the set of situations in (D, Γ). We write $(D, \Gamma) \models M$ to denote that $M = (\Psi, \Sigma)$ is a model of (D, Γ). A narrative is *consistent* if it has a model. Otherwise, it is *inconsistent*.

Intuitively, a model $M = (\Psi, \Sigma)$ of (D, Γ) encodes possible histories of the narrative. Each history is a trajectory of the form $\mathbf{s}_0, a_1, \mathbf{s}_1, a_2, \ldots, a_n, \mathbf{s}_n$ in which $\mathbf{s}_0, \ldots, \mathbf{s}_n$ are states, $a_1, \ldots, a_n$ are actions and $\mathbf{s}_i \in \Phi_D(a_i, \mathbf{s}_{i-1})$ for $i = 1, \ldots, n$.

In Chapter 3, we stop by the defining the notion of a model of a narrative as all models are equally possible. The discussion in Chapter 4 (Section 4.6) focuses on single out a possible model of a narrative by executing additional actions. The presence of the probability distribution in $\mathcal{B}_{\mathcal{P}}$ domain specifications provides us with the information to define the conditional probability of a model given the set of observations Γ as follows.

Let (D, Γ) be a $\mathcal{B}_{\mathcal{P}}$ narrative and

$$M = (\Psi, \Sigma) = ([\mathbf{s}_0, a_1, \mathbf{s}_1, a_2, \ldots, a_n, \mathbf{s}_n], \Sigma)$$

where $\mathbf{s}_0, a_1, \mathbf{s}_1, a_2, \ldots, a_n, \mathbf{s}_n$ is a trajectory. We first define the weight of M, denoted by $Weight(M)$, as:

$$Weight(M) = \begin{cases} 0 & \text{if } \Sigma(\mathsf{s}_C) \neq [a_1, \ldots, a_n] \\ P(\mathbf{s}_0) \times P_{a_1}(\mathbf{s}_1 \mid \mathbf{s}_0) \times \ldots \times P_{a_n}(\mathbf{s}_n \mid \mathbf{s}_{n-1}) & \text{otherwise} \end{cases} \tag{5.17}$$

Given a nnarative (D, Γ), we then define

$$P(M \mid \Gamma) = \begin{cases} 0 & \text{if } \mathcal{M} \text{ is not a model of } (\mathcal{D}, O'); \\ \frac{Weight(M)}{\sum_{(D,\Gamma) \models M'} Weight(M')} & \text{otherwise.} \end{cases} \tag{5.18}$$

The probabilistic correctness of a plan from a situation s with respect to a model M can then be defined as

$$P(\varphi \textbf{ after } \alpha \textbf{ at } \mathsf{s} \mid M) = \sum_{\mathbf{s}' \in \Phi([\beta], \mathbf{s}_0) \wedge \mathbf{s}' \models \varphi} P_{[\beta]}(\mathbf{s}' | \mathbf{s}_0) \tag{5.19}$$

where $\beta = \Sigma(\mathsf{s}) \circ \alpha$.

Finally, we define the (probabilistic) correctness of a plan from a situation s given a set of observations. This corresponds to counter-factual queries of Pearl [**?**] when the observations are about a sequence of actions that is different from he one in the hypothetical plan.

$$P(\varphi \textbf{ after } \alpha \textbf{ at } \mathsf{s} \mid \Gamma') = \textstyle\sum_{(D,\Gamma') \models M} P(M | \Gamma') \times P(\varphi \textbf{ after } \alpha \textbf{ at } \mathsf{s} | M) \tag{5.20}$$

Example 53 Consider the tossing coin example, where even if we observe that the coin lands heads after a toss, we have no idea what it will be when we pick it up and toss it again. Let D be the domain specification

$$\begin{array}{l} toss \textbf{ causes } tossed \\ \textbf{executable } toss \textbf{ if } \neg tossed \\ pickup \textbf{ causes } \neg tossed \\ tossed, head \textbf{ s\_causes } \neg tail \\ tossed, tail \textbf{ s\_causes } \neg head \\ tossed, \neg head \textbf{ s\_causes } tail \\ tossed, \neg tail \textbf{ s\_causes } head \end{array}$$

It can be easily shown that

$$\Phi(toss, \{\neg tossed, \neg head, \neg tail\}) = \{\{tossed, head, \neg tail\}, \{tossed, \neg head, tail\}\}.$$

Now even if we observe that $head$ **after** $toss$, the observation does not affect Φ. That is

$$\Phi([toss \circ pickup \circ a], \{\neg tossed, \neg head, \neg tail\}) = \{\{tossed, head, \neg tail\}, \{tossed, \neg head, tail\}\}.$$

Now we consider an example where the transition function changes with observations:

Example 54 Here, in the domain description we have an unknown variable u.

$toss$ **causes** $head$ **if** u
$toss$ **causes** $tail$ **if** $\neg u$
$head$ **causes** $\neg tail$
$tail$ **causes** $\neg head$
$\neg head$ **causes** $tail$
$\neg tail$ **causes** $head$

If we consider transition due to $toss$ on the n-state $\{\neg head, \neg tail\ \}$, we have $\Phi(toss, \{\neg head, \neg tail\}) = \{\{head, \neg tail\}, \{\neg head, tail\}\}$.

But now suppose we have the observation $O = \{head$ **after** $toss\}$. Then we will have $\Phi_O(toss, \{\neg head, \neg tail\}) = \{\{head, \neg tail\}\}$

5.6 Bibliographical Notes

Planning with actions with probabilistic effects has been discussed in [**?**, **?**]. Formalizing actions with probabilisitc effects using situation calculus has been done in [**?**, **?**, Rei01, **?**], nonmonotonic causal theory [**?**], independence choice logic [**?**, **?**], and probabilistic causal model [**?**, **?**, **?**].

In all these proposals, except in [**?**, **?**], actions are explicitly defined (with names) and their effects on the world are described by various means. In [**?**, **?**] the dynamics of the world is described through relationships between variables (which denote properties of objects in the world) that are expressed through functional relationships between them. Furthermore, probabilities are associated with a subset of variables called background (or exogenous) variables. Together they are referred to as *probabilistic causal models* (PCMs). The effect of actions are then formulated as "local surgery"on these models [**?**].

In this paper we showed how to integrate probabilistic reasoning into 'reasoning about actions'. the key idea behind our formulation is the use of two kinds of unknown variables: inertial and non-inertial. The inertial unknown variables are similar to the unknown variables used by Pearl. The non-inertial unknown variables plays a similar role as the role of nature's action in Reiter's formulation (Chapter 12 of [Rei01]) and are also similar to Lin's magic predicate in [**?**]. In Reiter's formulation a stochastic action is composed of a set of deterministic actions, and when an agent executes the stochastic action nature steps in and picks one of the component actions respecting certain probabilities. So if the same stochastic action is executed multiple times in a row an observation after the first execution does not add information about what the nature will pick the next time the stochastic action is executed. In a sense the nature's pick in our formulation is driven by a non-inertial unknown variable. We are still investigating if Reiter's formulation has a counterpart to our inertial unknown variables.

Earlier we mentioned the representation languages for probabilistic planning and the fact that their focus is not from the point of view of elaboration tolerance. We would like to add that even if we consider the Dynamic Bayes net representations as suggested by Boutilier and Goldszmidt, our approach is more general as we allow cycles in the causal laws, and by definition they are prohibited in Bayes nets.

5.6.1 Comparison with Pearls' notion of causality

Among the differences betweens his and our approaches are:

(**1**) Pearl represents causal relationships in the form of deterministic, functional equations of the form $v_i = f_i(pa_i, u_i)$, with $pa_i \subset U \cup V \setminus \{v_i\}$, and $u_i \in U$, where U is the set of unknown variables and V is the set of fluents. Such equations are only defined for v_i's from V.

In our formulation instead of using such equations we use static causal laws of the form (5.2), and restrict ψ to fluent formulas. I.e., it does not contain unknown variables. A set of such static causal laws define functional equations which are embedded inside the semantics. The advantage of using such causal laws over the equations used by Pearl is the ease with which we can add new static causal laws. We just add them and let the semantics take care of the rest. (This is one manifestation of the notion of 'elaboration tolerance'.) On the other hand Pearl would have to replace his older equation by a new equation. Moreover, if we did not restrict ψ to be a formula of only fluents, we could have written $v_i = f_i(pa_i, u_i)$ as the static causal law *true* **causes** $v_i = f_i(pa_i, u_i)$.

(**2**) We see one major problem with the way Pearl reasons about actions (which he calls 'interventions') in his formulation. To reason about the intervention which assigns a particular value v to a fluent f, he proposes to modify the original causal model by removing the link between f and its parents (i.e., just assigning v to f by completely forgetting the structural equation for f), and then reasoning with the modified model. This is fine in itself, except that if we need to reason about a sequence of actions, one of which may change values of the predecessors of f (in the original model) that may affect the value of f. Pearl's formulation will not allow us to do that, as the link between f and its predecessors has been removed when reasoning about the first action.

Since actions are first class citizens in our language we do not have such a problem. In addition, we are able to reason about executability of actions, and formulate indirect qualification, where static causal laws force an action to be in-executable in certain states. In Pearl's formulation, all interventions are always possible.

The works closely related to ours include [**?**], [**?**], [**?**]. [**?**] introduces *independent choice logic* (ICL) of independent choices and logic programs that specify the consequence of choices and actions. [**?**] has shown that many formalisms can be translated into ICL, including influence diagrams, Markov decision problems

(MDPs), strategic form of games and (dynamic) Bayesian networks. The relation between ICL and PAL is still unknown.

[**?**] translates causal models (without probabilities) into Fluent Calculus. Like the fluents $ab(V_i)$, their generic fluent $Denied(x)$ is used to simulate how an action inactivates respective functional equations. Nevertheless, the semantics of counterfactuals by [**?**] is essentially non-probabilistic. We also plan to relate this semantics to PAL in the future extension of our work.

[**?**] provide translations between PCM and ICL then extends the notions of structural causes and explanations to ICL-theories. The translations are restricted to a special class of PCMs whose causal graphs are *acyclic*. This restriction is probably due to the fact that ICL is defined with acyclic logic programs. Moreover, their and our method of analysis are orthogonal. In [**?**], the extended notions are first translated into PCM together with the ICL; then they are semantically interpreted in the translated PCM model. In our case, probabilistic queries can be interpreted separately in PCM and PAL semantics. Our contribution was showing that the translation from PCM to PAL is semantically correct.

[**?**] discuss differences between PAL and PCM in detail. With regard to our work in this paper, it is noticeable that we have used only inertial variables in the encoding of PCM. Another important aspect of the PAL encoding is that as the world progresses if we want to reactivate a previously inactivated functional equation $V_i = f_i(PA_i, U_i)$ all we need to do is make $ab(V_i)$ false. (Reactivation need to be done with care though.) In PCMs once a functional equation is inactivated it can no longer be activated.

5.7 Exercises

Exercise 2 Markov Decision Processes (MDP) can be used to specify sequential decision problems in fully observable environments in which actions have probabilistic effects. Given a domain with n fluents and m actions, a MDP specification will need to spcify the value $P_a(s'|s)$ for each triple of a state a and states s and s', which is the probability of reaching state s' if action a is done in state s.

Can a MDP be described by a $\mathcal{B}_\mathcal{P}$ domain specification? If so, what is the size of the specification in term of n and m?

Exercise 3 In our definitions of the transition function $\Phi(a, \mathbf{s})$ and the probability $P_a(\mathbf{s}'|\mathbf{s})$, observations do not play any role. However, they do affect the n-state transition function $\Phi(a, \mathbf{s}_F)$ and the probability $P_a(\mathbf{s}'_F|\mathbf{s}_F)$. Intuitively, this is because observations may tell us about the unknown variables. This additional information is monotonic in the sense that there value remains unchanged since actions do not affect unknown variables. Thus, in presence of observations O, we can define $\Phi_O(a, \mathbf{s}_F)$ as follows:

$$\Phi_O(a, \mathbf{s}_F) = \left\{ \mathbf{s}''_F \mid \mathbf{s}'' \in \bigcup_{\mathbf{s}' \models \mathbf{s}_F \& \mathbf{s}' \models O} \Phi(a, \mathbf{s}') \right\}$$

Is this true that if we have more observations, the transition function $\Phi_O(a, \mathbf{s}_F)$ becomes more deterministic? In other words, does $O \subseteq O'$ imply $|\Phi_{O'}(a, \mathbf{s}_F)| \leq |\Phi_{O'}(a, \mathbf{s}_F)|$?

Exercise 4 In Example 52, we briefly discussed the use of $\mathcal{B}_\mathcal{P}$ in encoding different causal models. In this exercise, we will work with *probabilistic causal model*, defined in []. We need the following definitions.

Definition 44 (Causal model) A *causal model* is a triple $M = \langle U, V, F \rangle$ where

- U is a set of *background* variables, (also called *exogenous*), that are determined by factors outside the model;
- V is a set $\{V_1, V_2, \ldots V_n\}$ of variables, called *endogenous*, that are determined by variables in the model - that is, variables in $U \cup V$; and
- F is a set of functions $\{f_1, f_2, \ldots f_n\}$, such that each f_i is a mapping from $U \cup (V \setminus V_i)$ to V_i, and such that the entire set F forms a mapping from U to V. In other words, each f_i tells us the value of V_i given the values of all other variables in $U \cup V$, and the entire set F has a unique solution for V, given a realization of U. Symbolically, the set of equations F can be represented by writing

$$V_i = f_i(PA_i, U_i)\ i = 1, \ldots, n$$

where PA_i is a subset of variables in $V \setminus V_i$ and U_i stands for a subset of variables in U.

Definition 45 (Submodel) Let M be a causal model, X be a set of variables in V, and x be a particular realization of X. A *submodel* M_x of M is the causal model $M_x = \langle U, V, F_x \rangle$ where $F_x = \{f_i : V_i \notin X\} \cup \{X = x\}$.

Definition 46 (Probabilistic Causal Model) A *probabilistic causal model* (PCM) is a pair $\langle M, P \rangle$ where M is a causal model and P is a probability distribution over the domain of U.

Definition 47 (Probabilistic queries and their entailment in PCM) Given a PCM $\mathcal{M} = \langle M, P \rangle$.

- The probability of x given an observation e is the conditional probability $P(x|e)$. If $P(x|e) = p$, we write $\mathcal{M} \models_C P(x|e) = p$.
- The probability of x given an intervention $do(y)$, denoted by $P(x|do(y))$, is the probability of x computed w.r.t the submodel $\mathcal{M}_y = \langle M_y, P \rangle$. If $P(x|do(y)) = p$, we write $\mathcal{M} \models_C P(x|do(y)) = p$.
- The probability of x given observation e and intervention $do(y)$, denoted by $P(x|e, do(y))$, is the probability $P(x|do(y))$ that is computed w.r.t the modified causal model $\mathcal{M}' = \langle M_y, P_e \rangle$. Here, P_e is the conditional probability $P(\_\,|e)$ computed w.r.t the model $\mathcal{M} = \langle M, P \rangle$. If $P(x|e, do(y)) = p$ we write $\mathcal{M} \models_C P(x|e, do(y)) = p$.

Thus, $\langle M, P \rangle \models_C P(x|do(y)) = p$ if and only if $\langle M_y, P \rangle \models_C P(x) = p$; and $\langle M, P \rangle \models_C P(x|e, do(y)) = p$ if and only if $\langle M_y, P_e \rangle \models_C P_e(x) = p$.

Given a PCM $\mathcal{M} = \langle M, P \rangle$ and assuming that the functions $f_i(PA_i, U_i)$ in M are logical functions, let $D(\mathcal{M})$ be a $\mathcal{B}_\mathcal{P}$ domain specification defined as follows.

- There are no non-inertial unknown variables.
- The inertial unknown variables are the exogenous variables in M with the same probability distributions:

probability of u **is** $P(u)$.

- The endogenous variables in M are fluents in $D(\mathcal{M})$. Moreover, for every fluent V_i, there is an additional fluent $ab(V_i)$ in $D(\mathcal{M})$.

- For each functional equation of the form $V_i = f_i(PA_i, U_i)$ in M, the following static causal axiom is in $D(\mathcal{M})$:

$$\neg ab(V_i) \quad \textbf{causes} \quad V_i \;\Leftrightarrow\; f_i(PA_i, U_i).$$

- For every fluent V_i, $D(\mathcal{M})$ has actions '$make(V_i)$', '$make(\neg V_i)$' with the following effects:

$$\begin{aligned} make(V_i) \quad &\textbf{causes} \quad \{ab(V_i), V_i\} \\ make(\neg V_i) \quad &\textbf{causes} \quad \{ab(V_i), \neg V_i\}. \end{aligned}$$

Let u be a set of background variable and v and w are subsets of endogenous variables. Show that

$$\mathcal{M} \models P(u|w) = p$$

if and only if

$$(D(\mathcal{M}), O_{ab}) \models_{\mathcal{BP}} P(\textbf{initially } u \mid O_{ab}, \textbf{initially } w) = p$$

where $O_{ab} = \{\textbf{initially } \neg ab(v') | v' \in v\}$