Copyright

by

Norman Clayton McCain

1997

Causality in Commonsense Reasoning about Actions

by

Norman Clayton McCain, B.A., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 1997

Causality in Commonsense Reasoning about Actions

Approved by
Dissertation Committee:

To my parents

Acknowledgments

The central ideas in this dissertation are the product of a long and close collaboration with my advisor, Vladimir Lifschitz, and my colleague, Hudson Turner. I do not claim any of these ideas as exclusively my own; in all cases they have a complicated "genealogy" and could not have arisen without many major contributions by Hudson and Vladimir. On the other hand, the presentation is finally my own, so I take full responsibility for any errors.

I wish to thank Vladimir for excellent advice and direction throughout my career as a graduate student, and both Vladimir and Hudson for many helpful suggestions. I also wish to thank Bob Boyer and Rob Koons for their advice and encouragement, especially during the early stages of my research. My interest in causation began in fact many years ago during a study of the logic of conditionals under Rob.

Norman Clayton McCain

The University of Texas at Austin
May 1997

Causality in Commonsense Reasoning about Actions

Publication No. ____________________

Norman Clayton McCain, Ph.D.

The University of Texas at Austin, 1997

Co-Supervisors: Vladimir Lifschitz

Robert Koons

In this dissertation, we investigate the role of causal knowledge in commonsense reasoning about action and change. We define a language in which a relatively simple form of causal knowledge is expressed. Using this language, we describe a novel approach to formalizing action domains as "causal theories"—including domains that involve concurrency, nondeterminism, and things that change by themselves. We show that a subclass of causal theories can be translated into propositional logic by a generalization of Clark's completion procedure for logic programs. Finally, we describe an implemented approach to automated query answering and "satisfiability planning" which is based on this translation.

Contents

List of Figures

Chapter 1

Introduction

The cause "is the sum total of the conditions positive and negative taken together . . . which being realized, the consequent invariably follows."[1]
John Stuart Mill

Beginning with John McCarthy's classic paper, "Programs with Common Sense" [1959], one of the main goals of artificial intelligence (AI) research has been to develop a computer program capable of commonsense reasoning about action and change—specifically, a program that is able to perform such tasks as prediction, explanation, and planning. McCarthy envisioned that a program of this kind might "reason" by manipulating explicit, declarative representations of the relevant knowledge. Following this broadly logicist approach, the central problems are to discover what this knowledge is and how it might be formalized. The aim of this dissertation is to advance our understanding of these issues.

[1]The quotation is from *A System of Logic*, page 332, Volume VII, *The Collected Works of John Stuart Mill*, [Robson, 1973].

1.1 The Role of Causal Knowledge

It has always been clear that causal knowledge plays a central role in commonsense reasoning about action and change. However, it has not always been clear what this role is, or that it cannot be played by non-causal knowledge as well. In the AI literature, this is evident in the use of state constraints for deriving the indirect effects of actions. Intuitively, a state constraint is a proposition that rules out certain states of the world as impossible but says nothing about causation.

The attempt to use state constraints to infer the indirect effects of actions has led to various difficulties in formalizing action domains. As an informal illustration of one of these difficulties, imagine a domain in which there is a light and a single switch. Ignoring such possibilities as a power failure or a burned out bulb, let us suppose that the following sentence is true.

(1) *The switch being closed causes the light to be on.*[2]

Let us also suppose that the causal relationship described in (1) is static, i.e., that there is no difference in time between the cause and the effect. In reality, there is a very small difference in time, but let us pretend that there is not.

Now let us compare the meaning of (1) with the meaning of the following state constraint.

(2) *In every state in which the switch is closed, the light is on.*

Given that the causal relationship described in (1) is static, intuitively it is clear that (1) implies (2). However (2) is compatible not only with the causal relationship described in (1) but also with other possibilities in which (1) may be false; for example, with the possibility that the light being off causes the switch to be open, with

[2]We understand (1) in a general sense, so that it is implicitly about all possible worlds and all times. In this respect, it is different from

The switch being closed caused the light to be on,

which intuitively relates the two mentioned facts only on a specific occasion.

the possibility that the switch being closed and the light being on are always joint effects of common causes, and even with the possibility that there is no causation at all. Therefore, (2) does not imply (1).

Now consider the following propositions about what can be achieved by performing actions.

(3) *The light can be turned on by closing the switch.*

(4) *The switch can be opened by turning off the light.*

Intuitively, from (1) we can infer (3), but not (4). On the other hand, from (2) we can infer neither (3) nor (4). The latter conclusion is not, however, the one usually drawn in the AI literature on reasoning about action and change. Instead, it is held that (3) in effect follows from (2), just as it does from (1). This conclusion is based on the assumption that whatever follows from the explicitly described effects of an action and known state constraints is an indirect effect. Conceptually, this is a mistake, one which rests on a rather common confusion of causal and non-causal grounds.[3] Technically, it leads to unintuitive results; for example, to the conclusion that not only (3), but also (4), follows from (2). It is clear, however, that (4) does not follow from (2). The explicit effect of turning off the light is that the light is off. From this and (2), it indeed follows that the the switch is open. But it does not follow that the switch being open is caused by turning off the light. So, intuitively,

[3]As evidence that the confusion is indeed common, we note that until recently a similar confusion infected standard decision theory [Jeffrey, 1965], where conditional probabilities, e.g., $Pr(S|A)$, were used in place of probabilities of causal connections, e.g., $Pr(A \text{ causes } S)$, in the definition of expected utility. The difference between the two probabilities is illustrated by the following example from [Stalnaker, 1981]. Suppose we discover a gene that causes both cancer and a predisposition to smoking, and as a result we come to view smoking as evidence for cancer, but no longer as a cause. Assuming that the probability function Pr maps propositions to degrees of belief, then in this situation $Pr(Cancer|Smoking)$ will be high, but $Pr(Smoking \text{ causes } Cancer)$ will be low. Intuitively, however, it is the latter probability, not the former, that is relevant in deciding whether or not to smoke. It is rational to act so as to change the facts from what they would have been had we not acted. But it is irrational to act merely in order to change our evidence for what the facts are. The confusion of evidential and causal conditions in standard decision theory has been set right in newer systems of "causal decision theory." See, for example, [Gibbard and Harper, 1981] and [Lewis, 1986a].

it does not follow that the switch can be opened by turning off the light.[4]

In the AI literature, Judea Pearl [1988] has emphasized the importance of the distinction between causal and non-causal grounds in general default reasoning. Some of the difficulties encountered in using state constraints for determining the indirect effects of actions were recognized by Ginsberg and Smith [1988b], Lifschitz [1990], and Lin and Reiter [1995]. Several authors have proposed to overcome these difficulties by replacing state constraints by representations of causal knowledge of one kind or another. These include Geffner [1989,1990], Elkan [1992], Brewka and Hertzberg [1993], Baral [1995], Lin [1995], McCain and Turner [1995], and Thielscher [1995a,1996]. The representation of causal knowledge presented in this dissertation is a simplification and generalization of the proposal of McCain and Turner.

1.2 What Kind of Causal Knowledge?

In view of the central role that causal knowledge plays in commonsense reasoning about action and change, the reader may expect to find as the centerpiece of this dissertation an analysis of the causal relation, and a logic of it. However, neither of these things is attempted here.[5] These omissions may reasonably seem to doom our project to failure, but in fact they do not.

To see why, consider, for example, the task of prediction. Suppose that ϕ is a proposition about the state of the world at a future time and that Γ is a set of true propositions about matters of particular fact in the actual world. Intuitively, in order to show that the prediction expressed by ϕ is also true in the actual world, it suffices to show that ϕ holds in every "causally possible world history" in which Γ is true. Thus, the task of prediction comes down to showing something about the

[4]Similar intuitions are appealed to in [Goldman, 1970]. If the reader does not share these intuitions, the technical difficulties illustrated in Chapter 3 may still convince him of the ill effects of using state constraints to infer the indirect of effects of actions.

[5]The problem of finding an adequate analysis and logic of the causal relation is a notoriously difficult one. The first attempt at a logic of causality was [Burks, 1951]. More recent attempts can be found in [von Wright, 1975], [Faye *et al.*, 1994], and [Koons, 1995].

set of causally possible world histories. As we will see in Chapter 8, planning does as well.

Our question, therefore, is this: for the purpose of determining the causally possible world histories, is it necessary to be able to represent and reason about *relations* of cause and effect? Apparently, it is not. Instead, it is sufficient to be able to represent and reason about the conditions under which facts are caused. Knowledge of the latter kind may be thought of as intermediate between non-causal knowledge (such as is expressed by state constraints) and knowledge of causal relations. The differences are revealed in the following sentence forms, where ϕ and ψ stand in place of sentences.

(5) *The fact that ϕ causes the fact that ψ.*

(6) *Necessarily, if ϕ then the fact that ψ is caused.*

(7) *Necessarily, if ϕ then ψ.*

A sentence of form (5) asserts that there is a causal relationship between the facts associated with ϕ and ψ. A sentence of form (6) makes no such assertion, but instead says only that whenever ϕ is true, something or other (possibly not the fact that ϕ itself) causes the fact that ψ. A sentence of form (7) says nothing about causation at all. Intuitively, (5) implies (6), and (6) implies (7), but neither implication holds in the other direction.

To see that (6) does not imply (5), consider a domain in which there is a switch S that controls two lights, A and B. Let t be a time. Suppose that S being closed at t causes both A and B to be on at t, and S being open at t causes both A and B to be off at t.[6] Then, intuitively, the following is true: necessarily, if A is on at t then S is closed at t. Since S being closed at t causes B to be on at t, the following sentence (which is easily rendered in form (6)) must also be true:

[6]In the regimented style of (5), we would say: The fact that S is closed at t causes the fact that A and B are on at t. We will continue to use less awkward phrasings such as those introduced above.

necessarily, if A is on at t, B is caused to be on at t. However, A being on at t does not cause B to be on at t, so the corresponding sentence of form (5) is false. Thus, in general, (6) does not imply (5).

We have argued that non-causal knowledge in the form of state constraints is an inadequate basis for reasoning about action and change. The same point can be made about knowledge of form (7). Causal knowledge is required. Our present point, however, is that for certain commonsense reasoning tasks—such as prediction and planning—whose solutions are definable in terms of the causally possible world histories, the strongest form of causal knowledge, i.e., knowledge of the form (5), is not required. Knowledge of form (6) is enough.

Although (6) is weaker than (5), this apparently makes no difference for the purpose of specifying the causally possible world histories. Intuitively, substituting knowledge of form (6) for (5) would have no effect on the facts that we judge to be caused in a world history, and thus also would have no effect on the world histories that we judge to be causally possible. Accordingly, we are in a very fortunate position indeed: even though we lack a satisfactory formal account of the causal relation itself, it seems we may nevertheless possess—in a relatively simple logic of sentences of form (6)—a way of representing our commonsense knowledge about causal relationships that is adequate for some our most important goals.

Unfortunately, not all commonsense reasoning tasks associated with domains of action and change come down to showing something about the causally possible world histories. Consider, for example, the task of finding a causal explanation for a known fact ψ. Intuitively, in order to find a causal explanation for ψ one would need to show that (5) holds for some true formula ϕ. (A good causal explanation might also have to satisfy other conditions, such as that ϕ not include irrelevant or redundant facts.) A similar conclusion of form (6), intuitively, would not suffice.[7]

[7]For example, in our two lights example, using knowledge of form (6) might lead us to answer the query "Why is light B on?" by replying "Because light A is on," since, as we have observed, in every world in which A is on, B is caused to be on. But this is not a correct causal explanation.

Until we understand the logic of the relation "causes," and not just the logic of the predicate "is caused," the goal of formalizing the task of causal explanation will continue to lie beyond our reach.[8] We are fortunate that for at least some important commonsense reasoning tasks less is required.

Causal knowledge of form (6) has been used by Geffner [1990] and, in a restricted form, by Lin [1995] in previous work on formalizing action domains. We owe a considerable debt to these authors.

1.3 A Brief Preview

As a preview of things to come, in this section we present our formalization of a simple action domain—the famous "Yale Shooting" domain [Hanks and McDermott, 1987]—and use it to illustrate our approach to query answering and planning. The example, which involves a turkey and a gun, is discussed in greater detail in later chapters.

In Figure 1.1, we present an input file describing the Yale Shooting domain. The file begins with the directive `declare_types`, which is used to define the specific language in which the domain is formalized. In this case, we declare the existence of three actions, two fluents, and four times. The atoms of the language include expressions such as `h(loaded,1)` and `o(shoot,2)`, which we read as "the fluent `loaded` holds at time 1" and "the action `shoot` occurs at time 2," respectively. In all there are twenty atoms. The second directive `declare_variables` is used to declare the types of the variables used in the specification of the domain. The remaining lines (a)–(i) in the file are the causal rules that we use to describe the domain. Each of these rules can be understood to express knowledge of form (6).

According to rule (a), necessarily, if the load action occurs at a time t then the gun is caused to be loaded at $t+1$. According to rule (b), necessarily, if the gun is

[8]The same is true of the the problem of abduction, insofar as this is understood as the problem of reasoning to the best causal explanation for given facts.

```
:- declare_types
        type(action,[load,wait,shoot]),
        type(fluent,[loaded,alive]),
        type(time,[0..3]),
        type(atom,[o(action,time),h(fluent,time)]).

:- declare_variables
        var(A,action), var(T,time), var(F,fluent).

h(loaded,T+1) <- o(load,T).                          % (a)
-h(alive,T+1) <- o(shoot,T), h(loaded,T).            % (b)
-h(loaded,T+1) <- o(shoot,T).                        % (c)

o(A,T) <- o(A,T).                                    % (d)
-o(A,T) <- -o(A,T).                                  % (e)
h(F,0) <- h(F,0).                                    % (f)
-h(F,0) <- -h(F,0).                                  % (g)

h(F,T+1) <- h(F,T), h(F,T+1).                        % (h)
-h(F,T+1) <- -h(F,T), -h(F,T+1).                     % (i)
```

Figure 1.1: The Yale Shooting domain

already loaded at a time t and the shoot action occurs at t then the turkey is caused to be dead at $t+1$. According to rule (c), necessarily, if the shoot action occurs at a time t then the gun is caused to be unloaded at $t+1$. According to rules (d)–(g), all facts about action occurrences at all times and about the values of fluents at time 0 are to be regarded as caused simply by virtue of their obtaining. In effect, these rules express that all such facts are exogenous to our formalization. Finally, rules (h) and (i) express the so-called "commonsense law of inertia." According to these rules, it is necessarily the case that if a fluent preserves its value from one time to the next, then its value of at the second time is caused.

In Figure 1.2, we illustrate query answering and planning with respect to the input file of Figure 1.1.[9] In the query, we assume that the turkey is initially alive, the gun is initially unloaded, and that the actions load, wait and shoot are performed at times 0, 1, and 2, respectively. From these premises, it can be shown that the turkey is not alive at time 3.[10] In the planning problem, we again assume that initially the turkey is alive and the gun is unloaded, and we pose the goal of killing the turkey. A plan is found that consists in performing exactly the actions load and shoot in that order. The goal is realized at time 2. (The action of loading the gun at time 2, which also appears in the output displayed in Figure 1.2, is not an essential part of the plan.)

The representational and computational ideas illustrated in the preceding example are developed in Chapters 5–9.

[9]Notice that according to the second line of Figure 1.1 there are 22 atoms rather than 20. The two extra atoms are `true` and `false`, which, for convenience, are automatically added to every domain description.

[10]This is not the standard "temporal projection problem" associated with the Yale Shooting domain, because we do not assume that load, wait, and shoot are the only actions performed at times 0, 1, and 2.

```
| ?- load_file(yale).
% 22 atoms, 51 rules, 26 clauses loaded.
yes
| ?- query.
enter facts (then ctrl-d)
|: h(alive,0).
|: -h(loaded,0).
|: o(load,0).
|: o(wait,1).
|: o(shoot,2).
|:
enter query
|: -h(alive,3).

yes
| ?- plan.
enter facts (then ctrl-d)
|: h(alive,0).
|: -h(loaded,0).
|:
enter goal
|: -h(alive,T).

0. -loaded  alive
Action(s): load

1.  loaded  alive
Action(s): shoot

2. -loaded -alive
Action(s): load

3.  loaded -alive
Action(s):

Verify plan?  y
plan verified.

yes
```

Figure 1.2: Examples of Query Answering and Planning

1.4 Organization

We begin in Chapter 2 by reviewing the background literature on reasoning about actions that is related to the present work.

The remaining chapters (up to the concluding one) can be divided into two parts—Chapters 3 through 4 and Chapters 5 through 9—corresponding to the two related formalisms that we will define.

In Chapter 3, we deepen our analysis of the inadequacies of state constraints. We propose a representation for static causal knowledge and a new definition of the possible next states. In Chapter 4, we describe an action description language which is based upon the causal framework of Chapter 3.

In Chapter 5, we define a general language of causal theories in which both static and dynamic causal laws are expressible. The semantics for the language of causal theories is obtained by simplifying and generalizing the definition of possible next states given in Chapter 3. In Chapter 6, we study the special class of causal theories that syntactically correspond to logic programs. We call such theories "objective programs." In Chapter 7, we describe how action domains can be formalized in the language of causal theories. For this purpose, we define the action description language $\mathcal{L}_{\mathrm{CL}}$ as a specialization of the language of causal theories. In Chapter 8, we describe two action query languages for use in conjunction with the language $\mathcal{L}_{\mathrm{CL}}$. We also lay the theoretical groundwork for a satisfiability-based approach to query answering and planning with respect to objective programs. In Chapter 9, we describe the program *satp*, which implements our approach to query answering and planning. We also present a number of examples.

Finally, in Chapter 10 we summarize our results and indicate some possible directions for future work.

In Appendix A, we discuss two other formalisms that are related to those defined in this dissertation. We compare the causal framework defined in Chapter 3 with the earlier framework of [McCain and Turner, 1995], and we generalize the language of causal theories in a modal framework. In Appendix B, we include a listing of the program `satp`, which is written in SICStus Prolog.

Chapter 2

Background

Although McCarthy began the study of commonsense reasoning about actions in 1959, it was later that he invented an adequate notation. This notation, called "the situation calculus," was first described in an unpublished memo [McCarthy, 1963] and was first published in [McCarthy and Hayes, 1969]. We will not use the situation calculus in this dissertation. However, since it is the most widely used notation for describing action domains, it is an appropriate place to begin our review.

A situation, according to McCarthy, is a complete state of the universe at an instant of time. A (propositional) fluent is a function whose domain is the set of situations and whose co-domain is the set of truth values. The situation calculus is a many-sorted, first-order language based on the ontology of situations, actions, and fluents. Given a fluent F and a situation S, one writes $F(S)$ or $\mathit{Holds}(F, S)$ in the situation calculus to express that F is true in the situation S. (In $\mathit{Holds}(F, S)$, the fluent F is reified, i.e., treated as an object.) Given an action A and a situation S, one writes $\mathit{Result}(A, S)$ to designate the situation that *would* result from performing the action A in the situation S. As an example, the sentence

$$\forall s[\mathit{Holds}(\mathit{Loaded}, s) \supset \neg \mathit{Holds}(\mathit{Alive}, \mathit{Result}(\mathit{Shoot}, s))] \tag{2.1}$$

expresses the fact that in every situation s, if the gun is loaded in s then the turkey is not alive in the situation that would result from shooting the gun in s. Here *Loaded* and *Alive* are constants of sort fluent, *Shoot* is a constant of sort action, and s is a variable of sort situation.[1]

The syntax of the situation calculus is simple and attractive but has seemed to many researchers to be expressively weak. It has been claimed, for example, that the situation calculus is unable to express the duration of actions, continuous time, or concurrency. In [Gelfond *et al.*, 1991b], these particular criticisms have been met and overcome. However, one limitation of the situation calculus remains, that is, its inability to represent nondeterministic actions—actions that can lead from a single situation to more than one possible next situation.[2]

Besides the introduction of the situation calculus, another important contribution of [McCarthy and Hayes, 1969] is the identification of the "frame problem." Typically, actions affect only a small number of fluents, while the values of the remaining fluents are assumed not to change. The frame problem is the problem of finding a way to avoid the need to mention the fluents that actions do not affect. As an example, a solution to the frame problem should make it unnecessary to include such axioms as the following.

$$\forall s[\mathit{Holds}(\mathit{Cloudy}, s) \supset \mathit{Holds}(\mathit{Cloudy}, \mathit{Result}(\mathit{Shoot}, s))]$$

$$\forall s[\neg \mathit{Holds}(\mathit{Cloudy}, s) \supset \neg \mathit{Holds}(\mathit{Cloudy}, \mathit{Result}(\mathit{Shoot}, s))]$$

[1]The sentence (2.1) corresponds to the causal rule (b) of Figure 1.1. Our description of the situation calculus deviates from [McCarthy and Hayes, 1969] in several inessential details.

[2]It may seem that the fact that *Result* is a function—one that maps an action and a situation to a unique resulting situation—precludes representing nondeterministic actions. However, this is so only if we understand $\mathit{Result}(A, S)$ to be the unique situation that *could* result from doing the action A in the situation S. If instead we understand it, as in fact we do, to be the unique situation that *would* result from doing A in S, then the functional nature of *Result* is compatible with nondeterminism. It may be possible, therefore, to extend the situation calculus so that nondeterministic actions can be represented without changing the *Result* function. (We will not explore this possibility here.) From this perspective, what is presupposed by the situation calculus is not that all actions are deterministic, but rather that even when an action is nondeterministic—so that more than one situation could result from performing it—still a unique situation would result. In other words, what is presupposed is a theory of counterfactuals like Stalnaker's [1968], which employs a "selection function" to pick out *the* world that would obtain under a possible counterfactual supposition, as opposed to a theory like Lewis's [1973], which does not presuppose that a unique such world exists.

Intuitively, these so-called "frame axioms" express that shooting the gun has no effect on whether or not the sky is cloudy.

The frame problem can be straightforwardly solved when the syntactic form of the represented knowledge is highly constrained. This is illustrated by the approaches described by Pednault [1989] and Reiter [1991]. Pednault automatically generates the frame axioms from "effect axioms" (axioms, such as (2.1), describing the effects of actions) in classical logic. This is possible if it is assumed that the effect axioms describe all of the effects of the actions. Reiter builds on Pednault's proposal but achieves additional parsimony by quantifying over actions, as recommended by Haas [1987] and Schubert [1990]. Elkan [1992] proposes an approach to formalizing action domains directly in classical logic.

The reasoning formalized in classical logic is monotonic in the sense that acquiring new information can only lead to additional conclusions being drawn, not to earlier conclusions being retracted. Commonsense reasoning, on the other hand, is not monotonic in this sense. Several general systems of nonmonotonic reasoning were introduced in the 1980's: McCarthy's [1980,1986] method of circumscription, Reiter's [1980] default logic, McDermott and Doyle's [1980] nonmonotonic logic, and Moore's [1985b] autoepistemic logic.

Because of the "negation as failure" rule, logic programming is also a system of nonmonotonic reasoning. The first semantics for logic programs with negation as failure was given by Clark [1978]. Many other semantics followed, the most influential being the well-founded semantics [Van Gelder *et al.*, 1990] and the stable model semantics [Gelfond and Lifschitz, 1988]. The connection between logic programming and nonmonotonic reasoning was clarified by translations from logic programming into the general nonmonotonic formalisms. Gelfond [1987], for example, showed how to translate logic programs into autoepistemic logic.

Even assuming rather strongly restricted forms of knowledge, the frame problem turned out to be unexpectedly difficult to solve in some of the above-mentioned systems of nonmonotonic reasoning. This was shown in the cases of circumscription and default logic by Hanks and McDermott [1986,1987], using the Yale Shooting problem as an example (cf. Section 1.3). In particular, they showed that an apparently natural formalization in circcumscription turned out to have consequences that were unexpectedly weak.

Various solutions to the Yale Shooting problem were proposed in the subsequent literature. Some, such as [Kautz, 1986] and [Shoham, 1986], employed alternatives to the above-mentioned systems of nonmonotonic reasoning. Others, such as [Lifschitz, 1987], [Gelfond, 1988], [Morris, 1988], [Baker, 1989], [Apt and Bezem, 1990], [Baker, 1991], and [Lifschitz, 1991], discovered workable formalizations within these systems.

In the early 1990's two frameworks were proposed for the systematic study of the problem of representing action domains, namely, the frameworks of Sandewall [1992a] and Gelfond and Lifschitz [1993]. The aim in both cases was to supplement the example-based methodology illustrated by earlier work on the Yale Shooting problem. Sandewall defined the general notion of an "inhabited dynamical system" and a extensive taxonomy of features and approaches to formalizing action domains. Gelfond and Lifschitz proposed to define specific high-level action description languages—each with its own special syntax and semantics—and to study the mathematical properties (for example, soundness and completeness) of translations from these languages into various approaches to formalizing actions. As an illustration, Gelfond and Lifschitz defined a simple high-level action description language $\mathcal{A}$ and proved the soundness of a translation from $\mathcal{A}$ into the language of extended logic programs. Subsequently, Kartha [1993] proved the soundness and completeness of translations from $\mathcal{A}$ into the above-mentioned approaches of Pednault, Reiter, and Baker. Sound and complete translations from $\mathcal{A}$ were also given by Denecker and De Schreye [1993] into the language of abductive logic programming and by Turner [1994] into the language of disjunctive logic programming.

Other high-level action description languages have been defined as approximate extensions of $\mathcal{A}$; for example, [Baral and Gelfond, 1993], [Kartha and Lifschitz, 1994], [Thielscher, 1994], [Giunchiglia and Lifschitz, 1995], [Giunchiglia *et al.*, 1995], [Baral *et al.*, 1995], and [Turner, 1997b]. The languages defined in [Kartha and Lifschitz, 1994] and [Giunchiglia *et al.*, 1995], $\mathcal{AR}_0$ and $\mathcal{AR}$, respectively, are particularly important in the present context, since they extend $\mathcal{A}$ to allow the representation of the indirect effects of actions. (The language $\mathcal{A}_{\rm CL}$ that we will define in Chapter 4 is closely related to $\mathcal{AR}$ and $\mathcal{AR}_0$, and even more closely related to the somewhat simpler language $\mathcal{AR}^-$ of [Kartha, 1995].) In [Giunchiglia and Lifschitz, 1995], a sound and complete translation is given from $\mathcal{AR}$ into the formalism of nested abnormality theories [Lifschitz, 1994]. Nested abnormality theories allow circumscription to be applied to parts of a theory, rather than only to the theory as a whole. The idea of circumscribing parts of an action theory was advocated by Crawford and Etherington [1992].

The central semantic definition in $\mathcal{AR}$ specifies the set of states that *could* result when an action is performed in a given state. This definition is an elaboration of a definition given by Winslett [1988], which in turn corrected an earlier definition given by Ginsberg and Smith [1988a]. Winslett's definition will be the starting point for our own investigations. As such, it will be carefully analyzed in Chapter 3.

As remarked in the previous chapter, there have recently been several proposals to replace the use of state constraints by causal knowledge of one form or another. A list of references is given at the end of Section 1.1. In Chapter 5 we will define a language of "causal theories," which is based on many of the same intuitions as Geffner's [1990] formalism. (See also [Geffner, 1989].) In particular, in describing action domains, both formalisms are used to represent knowledge of the form: *Necessarily, if ϕ then the fact that ψ is caused.* Like Geffner, we do not use the situation calculus but instead refer explicitly to times. In other respects, however, our approaches to formalizing action domains—for example, our solutions to the frame problem—are vastly different.

The proposals of Brewka and Hertzberg [1993] and McCain and Turner [1995] represent causal laws by inference rules. Baral [1995] represents static and dynamic causal laws in a notation of "state specifications," which, like the proposal of McCain and Turner, is closely related to the formalism of revision programs [Marek and Truszczyński, 1994]. In Chapter 3, we present a precursor to the language of causal theories, which is a modification of the causal framework of [McCain and Turner, 1995]. This new framework is compared to the earlier one in Appendix A.

Thielscher [1995a,1996] proposes to compute the indirect effects of an action by first modifying the initial state by the action's direct effects and then, as a "postprocessing" step, initiating a series of additional changes in accordance with the dynamic causal laws of the domain. The changes continue until a given set of state constraints is satisfied. Thielscher defines a procedure for generating causal laws from the given state constraints and a binary "influence relation" which specifies pairs of fluents such that the first can influence the second.

A simplified version of Lin's [1995] formalism and the causal framework defined in Chapter 3 are formally related in [McCain and Turner, 1997]. A variant of the language of causal theories is formally related to Lin's formalism in [Giunchiglia and Lifschitz, 1997].

Chapter 3

A Causal Approach to Ramifications and Qualifications

This chapter is concerned with the problem of determining the ramifications or indirect effects of actions. The problem is usually investigated in a framework in which action domains are described in part by state constraints. Informally, a state constraint is a formula that says of a proposition that it is true in every possible state of the world.[1] Our main objective in this chapter is to argue that an adequate theory of ramifications requires the representation of information of a kind that is not conveyed by state constraints, specifically, information about the conditions under which facts are caused. It turns out that this is also the information that is needed for an adequate theory of qualifications or derived action preconditions.

Previous approaches to the problem of ramifications have assumed a definition of the following kind: A ramification, roughly speaking, is a change (not

[1] In the situation calculus, a state constraint is a formula of form

$$\forall s \, \phi(s)$$

where s is the only siutation-valued term in the formula ϕ. In modal logic, a state constraint is a formula of the form

$$\Box \, \phi$$

where ϕ is a non-modal formula.

explicitly described) that is implied by the performance of an action. In our approach, we impose a stronger requirement, namely, that it be implied not only that the change occurs, but also that it is caused to occur. As we will see, this stronger requirement makes it possible to avoid unintended ramifications and to infer qualifications. (The importance of the latter is argued in [Ginsberg and Smith, 1988a] and [Lin and Reiter, 1994].) Again roughly speaking, our theory of qualifications is this: An action cannot be performed if the performance of the action implies that a change occurs that is not caused.

The main points can be illustrated by the following example. Imagine that Fred the turkey is on a walk. Consider the action of making Fred dead. Intuitively, as an indirect effect of performing the action, Fred will no longer be walking. The reason is that Fred's being dead is a causally sufficient condition for his not walking. Now consider the action of making Fred walk, but suppose that Fred is dead. Intuitively, the action cannot be performed. The reason is as follows: Fred can walk only if he is alive, but making him walk does not cause him to be alive; so unless he is already alive (or something in addition is done to cause him to become alive), he cannot be made to walk.

The conclusions reached in the previous paragraph are supported by the following facts about the so-called "by" relation. (The "by" relation is discussed in [Goldman, 1970] and [Davidson, 1980].) Intuitively, Fred can be made to not walk *by* making him be not alive, but he cannot be made to be alive *by* making him walk. If the indirect effects of an action are the facts made true by making the explicit effects of the action true, then we should expect Fred's not walking to be an indirect effect of making him not alive, but we should not expect Fred's being alive to be an indirect effect of making him walk.

In general, what can and cannot be done by doing something else is contingent upon the underlying causal connections and other relations of determination that hold among facts or states of affairs. (For a discussion of non-causal determination relations see [Kim, 1974].) Intuitively, state constraints say nothing about these, so it is not surprising that background knowledge in the form of state constraints should prove to be inadequate. The present chapter is closely related to [McCain and Turner, 1995]. However, the central definition in that paper is here replaced by a different one. The two definitions are compared in Appendix A.

3.1 Notation and Terminology

We begin with a language of propositional logic whose signature is given by a nonempty set of atoms. We view the atoms as propositions whose truth values may vary from one state of the world to the next. Accordingly, we refer to the atoms also as (propositional) *fluents*. By a *literal* we mean either an atom F or its negation $\neg F$. Throughout this dissertation the symbol L will be used to stand exclusively for literals. We use the expressions *True* and *False* as abbreviations for $(F \vee \neg F)$ and $(F \wedge \neg F)$, respectively, for some atom F. By an *interpretation* we mean a function that maps each atom to a truth value. We identify an interpretation I with the set of literals L such that $I \models L$.

We will consider a number of different frameworks in which the notion of a "state" is defined. In each of these frameworks, by a *state* we will mean an interpretation that "satisfies" (in some sense) the given background knowledge. What this means precisely will depend upon the kind of the background knowledge that is given. We will consider two kinds of background knowledge, constraints and static causal laws.

By a *constraint* we mean a formula. A standard example of a constraint (from [Baker, 1991]) is the formula $(Walking \supset Alive)$. Intuitively, by saying that a formula is a constraint, we say that it holds in every possible state of the world.[2]

[2]In a fully semantic treatment, this would be expressed by explicitly quantifying over all possible states, or by a modal operator with a similar meaning; for example, by a formula of one of the following forms:

$$\forall s(Walking(s) \supset Alive(s))$$

Formally, in a framework in which all background knowledge is given in the form of constraints, the states will be identified with the interpretations in which every constraint is true.

By a *static causal law* we mean an expression of the form

$$\phi \Rightarrow \psi \tag{3.1}$$

where ϕ and ψ are formulas (of the underlying propositional language). The expression (3.1) itself is not such a formula. In particular, it is different from the corresponding material conditional

$$\phi \supset \psi.$$

We call ϕ the *antecedent* of (3.1) and ψ the *consequent*.

Intuitively, (3.1) can be read as: *in every state in which ϕ is true, ψ is caused to be true.* An alternative reading is: *ϕ's being true (in a state) causes ψ to be true (in the same state).* Intuitively, the first reading is weaker than the second one. Roughly speaking, it says that ψ is caused whenever ϕ is true, but it does not say that ϕ causes ψ. Something else that is true whenever ϕ is true may cause ψ instead.[3]

Formally, in a framework in which all background knowledge is given in the form of static causal laws, the states will be identified with the interpretations I such that for every static causal law $\phi \Rightarrow \psi$, the corresponding material conditional $\phi \supset \psi$ is true in I.

$$\Box(Walking \supset Alive).$$

There are advantages to such fully semantic treatments. However, for the sake of simplicity and in order to ease comparison with previous work (specifically, [Winslett, 1988]), we have elected not to use them. Because of this decision, we have had to say something about constraints that in a fully semantic treatment the formulas say themselves, namely, that the constraints hold in every possible state of the world.

[3]In this chapter, we will not define truth conditions for static causal laws. Accordingly, whenever we say that a static causal law is true or false, we refer only to the truth or falsity of its informal reading. (Normally, it will not matter which of the two readings we adopt, though the weaker reading is always safe.) In Appendix A, we will describe an encoding of "causal laws" (not necessarily static) in modal logic. The truth conditions given by this encoding will correspond to the weaker reading given above.

3.2 Defining the Possible Next States

In this section our ultimate aim is to define $Res_D(E, S)$, the set of states that can result from performing an action with the explicit effect E in the state S, given background knowledge in the form of a set D of static causal laws. Our intention is that each state in $Res_D(E, S)$ should reflect the direct and indirect effects of performing the action. We will begin by reviewing Winslett's [1988] definition for the case in which the set D of static causal laws is replaced by a set of constraints.

3.2.1 Constraints

Using the terminology of section 3.1, Winslett defines $Res_B^W(E, S)$, the set of states that can result from performing an action with the explicit effect E in the state S, given the set B of constraints. (The superscript W is used merely to distinguish Winslett's definition from other definitions of *Res* to follow.) The states are the interpretations that satisfy each of the constraints in B.

Definition W For any set B of constraints, any explicit effect E, and any state S, $Res_B^W(E, S)$ is the set of states S' such that

(1) S' satisfies E, and

(2) no other state that satisfies E differs from S on fewer atoms, where "fewer" is defined by set inclusion.[4]

[4]The following definition differs from the one given by Winslett in one respect. Whereas *Res* is here defined for states S and S', Winslett defines *Res* for arbitrary interpretations. The following is a more faithful restatement of Winslett's definition. For any set B of constraints, explicit effect E, and interpretation I, $Res_B^W(E, I)$ is the set of interpretations I' such that

($1'$) I' satisfies $E \cup B$, and

($2'$) no other interpretation that satisfies $E \cup B$ differs from I on fewer atoms, where "fewer" is defined by set inclusion.

The difference between this definition and Definition W is unimportant, because (i) for every I, by ($1'$), every element of $Res_B^W(E, I)$ is a state, and (ii) we have no interest in the value of $Res_B^W(E, I)$ when I is not a state.

Example 3.1 Let us suppose that Fred the turkey is presently alive and walking, and that it is a cloudy day. The explicit effect of, say, shooting a gun is to make Fred be not alive. We also have the constraint that in every state Fred is walking only if he is alive.

$$\begin{aligned} S &= \{Alive, Walking, Cloudy\} \\ E &= \{\neg Alive\} \\ B &= \{Walking \supset Alive\}. \end{aligned}$$

Then

$$Res_B^W(E,S) = \{\{\neg Alive, \neg Walking, Cloudy\}\}.$$

Here, $\neg Walking$ is a ramification. The requirement in Winslett's definition that the elements of $Res_B^W(E,S)$ differ minimally from S rules out

$$\{\neg Alive, \neg Walking, \neg Cloudy\}$$

as a possible next state. The truth of the fluent $Cloudy$ is, as we say, "preserved by inertia." ◇

Intuitively, Winslett's definition behaves reasonably in the preceding example, but in many cases it does not. We will give two examples.

Example 3.2 In the following variation on Example 3.1, we consider the action of enticing Fred to walk. We suppose that this action has the explicit effect of making Fred walk whenever it is performed. Consequently, if Fred does not walk, it is certain that one cannot have enticed him to do so. (Of course, this does not mean that one did not *attempt* to entice him to walk, but that is a different action.) As in the introduction to this chapter, let us suppose that Fred is not alive.

$$\begin{aligned} S &= \{\neg Alive, \neg Walking, Cloudy\} \\ E &= \{Walking\} \\ B &= \{Walking \supset Alive\} \end{aligned}$$

It is easy to see that $Res_B^W(E,S) = \{\{Walking, Alive, Cloudy\}\}$. Hence, according to Winslett's definition, in this case *Alive* becomes true as a ramification of making *Walking* true. Intuitively, of course, this is not the desired result. We should instead conclude that, because Fred is dead, he cannot be enticed to walk. $\diamond$

In the preceding example, Winslett's definition yields a ramification—*Alive* becoming true—when the intuitively correct result would be a qualification—the action of enticing Fred to walk cannot be performed. The next example shows that Winslett's definition can also yield unintended ramifications in cases in which the action can be performed.

Example 3.3 The following domain is described in [Lifschitz, 1990]. Imagine that there are two ("three-way") switches that control the state of a single light. The switches may be up or down, and the light may be on or off. The light is on just in case the positions of the two switches agree, i.e., both are up or both are down.

$$\begin{aligned} S &= \{\neg\, Up1,\, Up2, \neg\, On\} \\ E &= \{\, Up1\} \\ B &= \{On \equiv (Up1 \equiv Up2)\}. \end{aligned}$$

Then

$$Res_B^W(E,S) = \{\,\{\,Up1,\, Up2,\, On\},\, \{\,Up1, \neg\, Up2, \neg\, On\}\,\}.$$

Each of the states in $Res_B^W(E,S)$ satisfies E and differs minimally from S among such states. However, the second state, which includes the unintended ramification $\neg\, Up2$, is anomalous. Intuitively, toggling switch 1 in state S will cause a change in the state of the light; it cannot cause a change in the state of switch 2. $\diamond$

Examples 3.2 and 3.3 show that Winslett's definition sometimes gives incorrect results. Later we will analyze what is wrong with Winslett's definition and show how it can be fixed by replacing background knowledge in the form of constraints by static causal laws. For this purpose, however, it will be useful to first recast Winslett's definition in an equivalent form given in [McCain and Turner, 1995].

In order to motivate this reformulation, we will present a series of definitions in which we introduce elements of Winslett's definition in successive steps. First, we introduce the assumption of inertia, which is needed to solve the frame problem. (In Winslett's definition, inertia is realized by condition (2).) Secondly, we introduce background knowledge in the form of constraints. Each definition will take the following form: For any state S and explicit effect E, $Res(E, S)$ is the set of states S' such that S' is precisely the set of literals that are entailed by E and the available background knowledge, together with any information provided by the assumption of inertia. Recall that the symbol L stands exclusively for literals.

In Definition 1, we do not admit the assumption of inertia or background knowledge of any kind. The set of states is simply the set of interpretations.

Definition 1 For any explicit effect E and state S, $Res^1(E, S)$ is the set of states S' such that

$$S' = \{L : E \models L\}.$$

According to Definition 1, every literal in a possible next state has to be entailed by the explicit effect E. Consider, for example, the state $S = \{p, q\}$ and an action that makes p false. Choosing $E = \{\neg p\}$, $Res^1(E, S) = \emptyset$. This is because E neither implies that q is true nor implies that q is false. On the other hand, choosing $E = \{\neg p \wedge q\}$, $Res^1(E, S) = \{\{\neg p, q\}\}$. Since E is required to imply not only the values of the fluents that change but also the values of those that do not, the frame problem is unsolved in Definition 1.

In Definition 2, we add to Definition 1 the assumption of inertia, according to which it is unnecessary to explain the values of fluents that do not change in the transition from state S to next state S'. These fluents and their values are represented by the set $S \cap S'$. We obtain Definition 2 by adding the literals in $S \cap S'$

as additional premises to the consequence relation in Definition 1. We still do not include background knowledge of any kind, so again the set of states is simply the set of interpretations.

Definition 2 For any explicit effect E and state S, $Res^2(E,S)$ is the set of states S' such that

$$S' = \{L : (S \cap S') \cup E \models L\}.$$

Consider again the state $S = \{p,q\}$ and an action that makes p false. Choosing $E = \{\neg p\}$, we find that $Res^2(E,S) = \{\{\neg p, q\}\}$. Because of the assumption of inertia, E now only has to specify the *changes* in the values of fluents. Consequently, in the restricted setting of Definition 2—one without background knowledge—the frame problem is solved.

Given a state S and an explicit effect E, the solutions to the equation in Definition 2 are the states S' that are fixpoints of the function

$$\lambda X.\{L : (S \cap X) \cup E \models L\}.$$

The following example shows that there may be more than one such fixpoint. Let $S = \{\neg p, \neg q\}$ and $E = \{p \vee q\}$. Then

$$Res^2(E,S) = \{\{\neg p, q\}, \{p, \neg q\}\}.$$

Notice, for example, in the case of the possible next state $\{\neg p, q\}$, the literal q is not entailed by E alone, nor is it preserved by inertia. Rather, it is entailed only by the union $(S \cap S') \cup E$, i.e., by $\{\neg p, p \vee q\}$. Intuitively, in determining what is caused in a candidate next state, facts preserved by inertia play the same role as facts belonging to the explicit effect.

The definition of the transition function for the language $\mathcal{A}$ of Gelfond and Lifschitz [1993] corresponds to the special case of Definition 2 in which the explicit effect E is required to be a consistent set of literals. Under this restriction, it is easy to see that $Res^2(E,S)$ will always be a singleton.

We are now ready to reformulate Winslett's definition for the framework with inertia and background knowledge in the form of a set B of constraints. The states will be the interpretations that satisfy every constraint in B. We obtain Definition 3 by adding the constraints to the premises of the consequence relation in Definition 2.

Definition 3 For any set B of constraints, explicit effect E, and state S, $Res_B^3(E, S)$ is the set of states S' such that

$$S' = \{L : (S \cap S') \cup E \cup B \models L\}.$$

Winslett's definition explicitly expresses the idea of minimizing change. Definition 3 has a very different form; it is given in terms of a fixpoint condition. Despite this difference, the two definitions are equivalent, as the following proposition shows.

Proposition 3.1 *[McCain and Turner, 1995] For any set B of constraints, explicit effect E, and state S, $Res_B^W(E, S) = Res_B^3(E, S)$.*

By Proposition 3.1, we know that Definition 3—like Winslett's definition—behaves incorrectly on Examples 3.2 and 3.3. (The reader is encouraged to check that this is so.) We are, however, in a better position to diagnose what is wrong with Definition 3, and we are in a better position to fix it.

3.2.2 Analysis

Let us consider, with respect to each of the Definitions 1–3, whether or not it requires every change in the value of a fluent to be caused.

According to Definition 1, in any possible next state S', every literal in S' is entailed by the explicit effect E. Since the formulas in E are understood to be the effects of an action, we see that Definition 1 does indeed require every change to be caused.

According to Definition 2, in any possible next state S', every literal in S' is entailed by $(S \cap S') \cup E$, i.e., by the literals preserved by inertia and formulas caused to hold in S'. As in the case of Definition 1, we again see that Definition 2 requires every change to be caused.

According to Definition 3, in any possible next state S', every literal in S' is entailed by $(S \cap S') \cup E \cup B$, i.e., by the literals preserved by inertia, formulas caused to hold in S', and by the constraints. Can we say that Definition 3 requires every change to be caused? Intuitively, we cannot. The reason for this is that the constraints in B (in contrast, for example, to the formulas in E) are not assumed to be caused in every candidate next state S'. They are, indeed, assumed to hold in S'—they are assumed to hold in every state—but they are not assumed to be caused to do so. As a result, the consequences of $(S \cap S') \cup E \cup B$ are not in general consequences only of what is preserved by inertia or caused to hold in S'.

As an example, consider the constraint

$$Walking \supset Alive \tag{3.2}$$

from Example 3.2. Intuitively, (3.2) is true in every possible state. However, we do not know that (3.2) is caused to be true in every candidate next state. Rather, what we know is that in certain states—namely, those in which $\neg Alive$ is true—$\neg Walking$ is caused to be true. Indeed, it is presumably on the basis of *this* causal knowledge that we know that (3.2) is true in every possible state.

As a second example, consider the constraint

$$On \equiv (Up1 \equiv Up2) \tag{3.3}$$

from Example 3.3. Intuitively, (3.3) also is true in every possible state. However, again we do not know that (3.3) is caused to be true in every candidate next state. Rather, what we know is that in certain states—namely, those in which $(Up1 \equiv Up2)$ is true—On is caused to be true, and that in certain other states—namely, those in which $(Up1 \equiv Up2)$ is false—On is caused to be false.

In order to ensure that every change is caused, we should modify Definition 3 so that instead of including the non-causal constraints in B as premises, we include (in addition to the literals preserved by inertia) only formulas that we know to be caused to be true in S'. Thus, in Example 3.2, we should include $\neg Walking$ as a premise if $\neg Alive$ is true in S', while in Example 3.3, we should include On as a premise if $(Up1 \equiv Up2)$ is true in S', and $\neg On$ as a premise if $(Up1 \equiv Up2)$ is false in S'. In the next section, we will modify Definition 3 to work in this way.

According to our analysis, the defect in Definition 3 is that it permits uncaused premises (namely, the constraints in B) where caused premises are required. We conclude, therefore, that Definition 3 rests on a confusion of causal and non-causal grounds. We should observe, however, that this conclusion assumes that by classifying a formula as a constraint we intend only to convey that the formula is true in every state, and not that it is, moreover, caused to be true. If we were to revise our view of the intended meaning of constraints to say that constraints are formulas that are caused to be true in every state (so that the constraint ψ is understood in the same manner in which we understand the static causal law $True \Rightarrow \psi$), then Definition 3 would no longer rest upon the above mentioned confusion. Instead, its less serious failing would be that it does not employ a sufficiently expressive language for describing the conditions under which facts are caused. In order to formalize Examples 3.2 and 3.3, we need to be able to write causal laws of the form $\phi \Rightarrow \psi$, where ϕ is other than $True$.

3.2.3 Static Causal Laws

Given the reformulation of Winslett's definition in Definition 3 and our analysis of its defects in the preceding section, it is now a simple matter to define the possible next states in the presence of background knowledge in the form of static causal laws.

The states will now be the interpretations that satisfy, for each static causal law $\phi \Rightarrow \psi$ in D, the corresponding material conditional $\phi \supset \psi$.

Given a set D of static causal laws and a state S, we define

$$D^S = \{\psi : \text{ for some } \phi,\ \phi \Rightarrow \psi \in D \text{ and } S \models \phi\}.$$

That is, D^S is the set of consequents of all causal laws in D whose antecedents are true in S.

Definition 4 For any set D of static causal laws, explicit effect E, and state S, $Res_D^4(E, S)$ is the set of states S' such that

$$S' = \{L : (S \cap S') \cup E \cup D^{S'} \models L\}.$$

Example 3.4 Consider the following variation on Example 3.1 in which the constraint is replaced by the static causal law in D below.

$$\begin{aligned} S &= \{Alive, Walking, Cloudy\} \\ E &= \{\neg Alive\} \\ D &= \{\neg Alive \Rightarrow \neg Walking\} \end{aligned}$$

We find that

$$Res_D^4(E, S) = \{\{\neg Alive, \neg Walking, Cloudy\}\}.$$

Again, $\neg Walking$ is a ramification. $\diamond$

The superiority of Definition 4 over Definition 3, and of static causal laws over constraints, is illustrated by the following example.

Example 3.5 As in Example 3.2, let us again consider the case in which we attempt to entice Fred the turkey to walk. However, let us replace the constraint of Example 3.2 by the static causal law in D below.

$$\begin{aligned} S &= \{\neg Alive, \neg Walking, Cloudy\} \\ E &= \{Walking\} \\ D &= \{\neg Alive \Rightarrow \neg Walking\} \end{aligned}$$

Now $Res_D^4(E,S)$ is empty, which means that the action cannot be performed in S.

Intuitively, this new result is correct. We cannot make *Alive* true by making *Walking* true. Therefore, in the state S we cannot perform an action whose entire explicit effect is $\{Walking\}$. The reason is that making *Walking* true implies a change—namely, making *Alive* true—that the action does not cause.[5] This is an example of a derived qualification. $\diamond$

Another advantage of using static causal laws is illustrated by the following variation on Example 3.3.

Example 3.6 Again imagine that there are two switches and a light. The switches may be up or down, and the light may be on or off. The light is caused to be on if the positions of the two switches agree, i.e., both are up or both are down, and caused to be off otherwise.

$$\begin{aligned} S &= \{\neg Up1, Up2, \neg On\} \\ E &= \{Up1\} \\ D &= \{(Up1 \equiv Up2) \Rightarrow On,\ \ \neg(Up1 \equiv Up2) \Rightarrow \neg On\}. \end{aligned}$$

Then

$$Res_D^4(E,S) = \{\{Up1, Up2, On\}\}.$$

$\diamond$

Example 3.3 is identical to Example 3.6, except that in the former example, instead of the set D of static causal laws, we had the set B of state constraints

$$\{On \equiv (Up1 \equiv Up2)\}.$$

We found that $Res_B^3(E,S)$ contained in addition to the state in $Res_D^4(E,S)$ also the state

$$\{Up1, \neg Up2, \neg On\} \tag{3.4}$$

[5]Since D contains $\neg Alive \Rightarrow \neg Walking$, we know that the material conditional $(Walking \supset Alive)$ holds in every state. This is the sense in which *Walking* implies *Alive*.

which, as we observed, is anomalous and results from the unintended ramification $\neg Up2$. In [Lifschitz, 1990] and [Kartha and Lifschitz, 1994], this ramification is blocked by declaring *Up1* and *Up2* to be "in the frame" and *On* to be "not in the frame." (The assumption of inertia is applied only to the fluents that are in the frame.) By contrast, the use of static causal laws in place of constraints makes the frame/nonframe distinction unnecessary for the purpose of limiting possible ramifications.[6] Intuitively, (3.4) is ruled out as a possible next state according to Definition 4 by the fact that there is no causal explanation for $\neg Up2$.

The next example illustrates a possibly unexpected feature of Definition 4.

Example 3.7 Consider the following extension to an example that we considered earlier in relation to Definition 2.

$$\begin{aligned} S &= \{\neg p, \neg q\} \\ E &= \{p \vee q\} \\ D &= \{p \Rightarrow q\} \end{aligned}$$

Previously, we observed that

$$Res^2(E,S) = \{\{\neg p, q\}, \{p, \neg q\}\}.$$

Before calculating $Res^4_D(E,S)$, one might, therefore, expect that $Res^4_D(E,S)$ is $\{\{\neg p, q\}, \{p, q\}\}$; the expectation being that the static causal law $p \Rightarrow q$ will leave $\{\neg p, q\}$ unchanged (since p is not satisfied in this state) but will transform $\{p, \neg q\}$ into $\{p, q\}$. However, this expectation derives from mistakenly treating static causal laws as if they were dynamic, that is, as if they related facts in successive states rather than contemporaneously. Intuitively, given D, the state $\{p, \neg q\}$ does not describe a causally possible state. Thus, $\{p, \neg q\}$ cannot first come into existence

[6]The frame/nonframe distinction is one of several closely related approaches to controlling ramifications by dividing fluents into different categories. These approaches are compared in [Sandewall, 1995]. An insightful analysis of the limitations of such categorization-based approaches is given in [Thielscher, 1996].

and then be transformed into $\{p, q\}$, as the reasoning leading to this expectation suggests.

In fact, $Res^4_D(E, S)$ includes only $\{\neg p, q\}$. Intuitively, the state $\{p, q\}$ is not a possible next state because there is no causal explanation for the change in the value of p. $\Diamond$

3.3 Ramification and Qualification Constraints

Lin and Reiter [1994] draw a pragmatic distinction between two kinds of state constraints: *ramification constraints*, which yield indirect effects, and *qualification constraints*, which yield action preconditions. As they observe, the same distinction was drawn earlier by Ginsberg and Smith [1988b]. In the language of static causal laws, we can give a syntactic form to this distinction. Suppose that ϕ is a constraint. If we wish ϕ to function as a ramification constraint, we write the rule

$$\mathit{True} \Rightarrow \phi.$$

If instead we wish ϕ to function as a qualification constraint, we write the rule

$$\neg\phi \Rightarrow \mathit{False}.$$

In Definition 3, constraints function exclusively as ramification constraints. The correctness of our encoding of ramification constraints is, therefore, corroborated by the following proposition.[7]

Proposition 3.2 *Let B be a set of constraints, and*

$$D = \{\mathit{True} \Rightarrow \psi : \psi \in B\}.$$

For any state S and explicit effect E, $Res^3_B(E, S) = Res^4_D(E, S)$.

[7]The corroboration is peculiar because a constraint is properly understood not to convey causal knowledge at all. Its intuitive meaning is expressed by a qualification constraint rather than a ramification constraint. Nevertheless, constraints do function as ramification constraints in Definition 3. This peculiar state of affairs is explained by the fact that Definition 3 only requires the literals in a possible next state to be implied, not caused. Because of this weaker requirement on possible next states, non-causal knowledge behaves in Definition 3 like causal knowledge.

Proof. It suffices to observe that for any state S', $D^{S'} = B$. $\square$

As an example of a domain in which a state constraint is intended to function as a qualification constraint, we consider a simplified version of a domain from [Lin and Reiter, 1994].

Example 3.8 Imagine an ancient kingdom in which there are two blocks. Either block may be painted yellow, but by order of the emperor at most one of the blocks is permitted to be yellow at a time. Consider a state in which the second block is yellow. Intuitively, in this state it is impossible to only perform the action of painting the first block yellow. However, representing the emperor's decree by a ramification constraint does not conform to this intuition. Indeed, let

$$\begin{aligned} S &= \{\neg Yellow1, Yellow2\} \\ E &= \{Yellow1\} \\ D &= \{\mathit{True} \Rightarrow \neg(Yellow1 \wedge Yellow2)\}. \end{aligned}$$

Then

$$Res^4_D(E,S) = \{\{Yellow1, \neg Yellow2\}\}.$$

So painting the first block yellow changes the color of the second block! On the other hand, if we represent the emperor's decree as a qualification constraint by redefining D as

$$D = \{(Yellow1 \wedge Yellow2) \Rightarrow \mathit{False}\}$$

then $Res^4_D(E,S)$ is empty. This result, unlike the previous one, agrees with our intuition that it is impossible to (only) paint the first block yellow in state S. $\diamond$

The following straightforward proposition shows that static causal laws of the form we write for qualification constraints cannot lead to ramifications, but can only rule them out.

Proposition 3.3 *Let D be a set of static causal laws and ϕ be a formula. Let $D' = D \cup \{\neg\phi \Rightarrow \mathit{False}\}$. For any explicit effect E, and states S and S', $S' \in \mathit{Res}^4_{D'}(E,S)$ if and only if $S' \in \mathit{Res}^4_D(E,S)$ and $S' \models \phi$.*

Proof. For the left-to-right direction, suppose $S' \in \mathit{Res}^4_{D'}(E,S)$. Then we know that $\mathit{False} \notin D'^{S'}$. So $S' \models \phi$. Therefore, $D^{S'} = D'^{S'}$. So $S' \in \mathit{Res}^4_D(E,S)$. For the right-to-left direction, suppose $S' \in \mathit{Res}^4_D(E,S)$ and $S' \models \phi$. Then again we know that $D'^{S'} = D^{S'}$. So $S' \in \mathit{Res}^4_{D'}(E,S)$. $\Box$

Brewka and Hertzberg [1993] propose a modification of Winslett's definition [1988] in which static causal laws (represented as inference rules) play a role in the definition of minimal change between states. Roughly speaking, in their definition *uncaused* changes in the values of fluents are minimized. In our definition, on the other hand, uncaused changes are strictly forbidden. Because of the role that minimal change continues to play in their definition, Brewka and Hertzberg cannot express qualification constraints in the manner shown in Example 3.8. Nor do they obtain derived qualifications of the kind illustrated by Example 3.5. Notice that the static causal law $\neg \mathit{Alive} \Rightarrow \neg \mathit{Walking}$ has neither the form of a ramification constraint nor the form of a qualification constraint. In fact, as illustrated in Section 3.2.3, it sometimes leads to ramifications and sometimes qualifications.

3.4 The Art of Causal Formalization

In formulating the static causal laws of a domain, we are faced with many alternatives that we do not face in writing constraints. Deciding among these alternatives requires one to be sensitive to the existence and nature of the causal dependencies. As an example, consider the following constraint from Example 3.1.

$$\mathit{Walking} \supset \mathit{Alive} \tag{3.5}$$

Each of the following causal laws rules out the same possible states as (3.5), but in other respects their meanings diverge.

$$\mathit{True} \Rightarrow (\mathit{Walking} \supset \mathit{Alive}) \tag{3.6}$$

$$\neg(\mathit{Walking} \supset \mathit{Alive}) \Rightarrow \mathit{False} \tag{3.7}$$

$$\mathit{Walking} \Rightarrow \mathit{Alive} \tag{3.8}$$

$$\neg \mathit{Alive} \Rightarrow \neg \mathit{Walking}. \tag{3.9}$$

The static causal law (3.6) functions as a ramification constraint, and (3.7) as a qualification constraint, while (3.8) and (3.9) are "proper" causal laws, with non-trivial antecedents and consequents. Although all of these laws rule out the same possible states, each has a potentially different impact on the possible next states according to Definition 4. The art of causal formalization often lies in deciding among such alternatives as these.

In this case, we know, by Proposition 3.2, that writing (3.6) would yield from Definition 4 the same result that we have obtained from Definition 3 by writing the constraint (3.5), namely, that Fred can be brought back to life by enticing him to walk. (See Example 3.2.) It is easy to see that (3.8) would yield the same unintuitive result. On the other hand, by Proposition 3.3 we know that writing (3.7) would not yield any ramifications. Only (3.9) gives the desired results.

It is vital, however, that formal considerations such as these not be the only basis we have for deciding among different formulations. A formalism that required one to work out the mathematical consequences of alternative formulations in order to decide among them would be of little use for representing commonsense knowledge. Instead, it must be sufficient to consider which formulations are true.

Intuitively, not being alive *is* a causally sufficient condition for not walking, so (3.9) satisfies the truth test. On the other hand, it is not the case that not walking is a causally sufficient condition for being not alive, so (3.8) does not. The informal readings of (3.6) and (3.7) are unusual, due to the appearance of *True* and

False in these expressions, so the truth test is difficult to apply to these. However, it is not difficult to see that a qualification constraint such as (3.7) rules out possible states without saying anything about causation. Accordingly, (3.7) has the same intuitive meaning as (3.5), that is, it expresses a state constraint. When viewed in this light, it is clear that (3.7) also satisfies the truth test, although, assuming (3.9), it is redundant.[8]

Definition 4 implicitly assumes that the set D of static causal laws is complete, in the sense that in every state S, D^S entails exactly the formulas that are statically caused to be true in S. Accordingly, our aim in formalizing a domain is to include in D any true static causal law that is non-redundant relative to D. This explains why it would be insufficient to include (3.7) in our formalization but not (3.9).

One of the most difficult problems in formalizing action domains is deciding when to write a ramification constraint, such as (3.6), and when to write a "proper" causal law, such as (3.9). In this regard, it is useful to ask oneself whether there is a fixed causal ordering among the fluents involved, or whether the ordering might vary from one possible occasion to the next. Applying this consideration to the case at hand, we would ask does Fred's being not alive always (on every imaginable occasion) cause his not walking, or can Fred's walking also sometimes cause his being alive. Whereas (3.6) recognizes the latter possibility, (3.9) does not. Since we know that Fred's walking can never cause his being alive, we know in this case—even before considering the mathematical consequences of the two formulations—that we should write (3.9) rather than (3.6).

Von Wright [1975] distinguishes cases in which "the cause-effect distinction refers to the history of an individual occasion" from cases in which it "resides in the relation between the generic factors themselves." Using this terminology, a proper causal law, such as (3.9), describes a causal relationship between generic factors,

[8]Given a set D of static causal laws, a static causal law R is "redundant" if adding R to D has no effect on the set of states and for every state S, D^S and $(D \cup \{R\})^S$ have the same models.

e.g., the fluents $\neg Alive$ and $\neg Walking$. A ramification constraint, such as (3.6), does not. It may give rise to different relations of cause and effect on different occasions.

The importance of ramification constraints is illustrated by the following example.

Example 3.9 Imagine a seesaw whose two ends are labeled A and B. Whenever A is up, B is down, and *vice versa*. Imagine that we are capable of directly raising and lowering both A and B. The causal relationship between the positions of the two ends of the seesaw can be formalized by the ramification constraint

$$True \Rightarrow Up(A) \equiv \neg Up(B). \tag{3.10}$$

Consider the following cases.

$$\begin{aligned} S_1 &= \{Up(A), \neg Up(B)\} \\ E_1 &= \{\neg Up(A)\} \\ D &= \{True \Rightarrow Up(A) \equiv \neg Up(B)\} \end{aligned}$$

$$\begin{aligned} S_2 &= \{Up(A), \neg Up(B)\} \\ E_2 &= \{Up(B)\} \\ D &= \{True \Rightarrow Up(A) \equiv \neg Up(B)\} \end{aligned}$$

Both cases yield, by Definition 4, the same unique possible next state, namely,

$$\{\neg Up(A), Up(B)\}.$$

Intuitively, in the first case we can say that lowering A caused B to go up, and in the second case we can say that raising B caused A to go down. $\diamond$

Example 3.10 Now, let us attempt to reformalize the Seesaw domain, representing the causal relationship between the positions of the two ends of the seesaw, A

and B, not by a ramification constraint, but by static causal laws with non-trivial antecedents and consequents. We will need the following four laws to cover all of the possibilities of raising and lowering the two ends of the seesaw.

$$Up(A) \Rightarrow \neg Up(B) \tag{3.11}$$

$$\neg Up(A) \Rightarrow Up(B) \tag{3.12}$$

$$Up(B) \Rightarrow \neg Up(A) \tag{3.13}$$

$$\neg Up(B) \Rightarrow Up(A). \tag{3.14}$$

Is this also a reasonable formalization? At first, it may appear that it is. For instance, if we substitute (3.11)–(3.14) for D in the two cases above, the results are unchanged. However, the following case shows that the formalization is not reasonable after all.

$$\begin{aligned} S_3 &= \{Up(A), \neg Up(B)\} \\ E_3 &= \emptyset \\ D_3 &= (3.11)\text{–}(3.14) \end{aligned}$$

Here we suppose that an action with no explicit effect (for example, the action of waiting) is performed. Nevertheless, we find that $Res^4_{D_3}(E_3, S_3)$ contains two possible next states—not only the state S_3 itself, which we expect, but also the state $S' = \{\neg Up(A), Up(B)\}$, which we do not. To see that S' is, indeed, a possible next state, notice that $D^{S'}$ is in fact S' itself. Since $(S_3 \cap S') \cup E_3 = \emptyset$, it follows that $S' \in Res^4_{D_3}(E_3, S_3)$. $\diamond$

According to the formalization of Example 3.10, one possible result of merely waiting is that the seesaw spontaneously changes position. Intuitively, the reason for this odd behavior is a loop through the laws (3.12) and (3.13). Intuitively, according to these two laws, $\neg Up(A)$ causes $Up(B)$, and *vice versa*. If such causal laws were true, it would make sense that $\neg Up(A)$ and $Up(B)$ could cause each other to be true, and, by so doing, also cause the seesaw to spontaneously change positions. Thus, the behavior of Definition 4 with respect to these static causal laws is arguably correct. Intuitively, however, the causal laws themselves are false; the causal relationship between $\neg Up(A)$ and $Up(B)$ is not between "the generic factors themselves," but in each case concerns "the history of an individual occasion."[9]

A better formalization of the seesaw domain uses the static causal law (3.10), which does not give rise to the possibility of spontaneous change.

$$\begin{aligned} S_4 &= \{Up(A), \neg Up(B)\} \\ E_4 &= \emptyset \\ D &= \{True \Rightarrow Up(A) \equiv \neg Up(B)\} \end{aligned}$$

We find that $Res^4_D(E_4, S_4) = \{\{Up(A), \neg Up(B)\}\}$.

[9] Under special syntactic conditions, which are violated in Example 3.10, spontaneous change is impossible. This is shown in Appendix A (Proposition A.4).

Chapter 4

A Simple Action Description Language

In this chapter, we define an action description language $\mathcal{A}_{\mathrm{CL}}$, which is based on Definition 4 of Chapter 3. In addition to static causal laws, the language $\mathcal{A}_{\mathrm{CL}}$ includes symbols that designate actions and a second type of proposition that is used to describe their explicit effects.

4.1 Action Languages

The first high-level action language was the language $\mathcal{A}$ of Gelfond and Lifschitz [1993]. Following $\mathcal{A}$, a number of other high-level action languages have been defined. References to some of these are given in Chapter 2.

4.1.1 Two Components

According to Lifschitz [1995], the language $\mathcal{A}$ and its successors can be viewed as a combination of two sublanguages. First, there is an *action description language* in which the effects of actions in states are described. A set of propositions in an action description language specifies a transition system in approximately the sense of finite automata theory. Secondly, there is an *action query language* in which assertions about the paths in a transition system are expressed. (Intuitively, the paths through a transition system represent causally possible world histories.) In $\mathcal{A}$ itself, the action description language consists of "effect propositions" such as

$$\mathit{Shoot}\ \textbf{causes}\ \neg \mathit{Alive}\ \textbf{if}\ \mathit{Loaded}$$

and the action query language consists of "value propositions" such as

$$\neg \mathit{Alive}\ \textbf{after}\ \mathit{Load}; \mathit{Wait}; \mathit{Shoot}.$$

An *action language* is a combination of an action description language and an action query language. A general method for combining two such languages with the same signature is defined in [Lifschitz, 1995].

4.1.2 Action Description Languages

The signature for an action description language consists of two nonempty sets of symbols: a set $\mathbf{A}$ of action names and a nonempty set $\mathbf{F}$ of fluent names. A set of propositions in an action description language specifies, as we have said, a structure called a transition system. Definitions similar to the following appear in [Boutilier and Friedman, 1995] and [Lifschitz, 1995].

A *transition system* for an action description language with the signature $\langle \mathbf{A}, \mathbf{F} \rangle$ consists of the following elements:

- a nonempty set S of objects called *states*,
- a function V from $S \times \mathbf{F}$ into the set of truth values, and
- a function R from $\mathbf{A} \times S$ into the powerset of S.

The function R is called the *transition function* of the system. The elements of $R(A, s)$ are the states that could result (nondeterministically) from performing the action A in the state s.

$$\begin{tikzcd}
\{\neg H, T\} \arrow[r, shift right, "\mathit{PickUp}"'] & \{\neg H, \neg T\} \arrow[l, shift right, "\mathit{Toss}"'] \arrow[r, shift left, "\mathit{Toss}"] & \{H, \neg T\} \arrow[l, shift left, "\mathit{PickUp}"]
\end{tikzcd}$$

Figure 4.1: A transition system for a Coin Tossing domain

We say that A is *executable* in s if $|R(A,s)| > 0$. We say that A is *deterministic in s* if $|R(A,s)| = 1$. We say that A is *nondeterministic in s* if $|R(A,s)| > 1$. We say that A is *nondeterministic* if, for some $s \in S$, A is nondeterministic in s. Otherwise, we say that A is *deterministic*.

We identify an interpretation I of $\mathbf{F}$ with the set of literals L such that $I \models L$. By $V(s)$ we denote the interpretation

$$\{F : F \in \mathbf{F} \text{ and } V(s,F)\} \cup \{\neg F : F \in \mathbf{F} \text{ and not } V(s,F)\}.$$

Employing terminology from [Carnap, 1947], we say that $V(s)$ is the *state description* of s, and we say that s *realizes* the state description $V(s)$.

It is common in giving the semantics of action description languages to take the set S to be a set of state descriptions and to specify that for every $s \in S$, $V(s) = s$. The significance of these decisions is discussed in Section 4.2.2.

As an example, let us suppose that the signature for a given action description language is

$$\langle \mathbf{A}, \mathbf{F} \rangle = \langle \{\mathit{Toss}, \mathit{Pickup}\}, \{\mathit{Heads}, \mathit{Tails}\} \rangle.$$

In the transition system pictured in Figure 4.1 we abbreviate *Heads* and *Tails* by H and T, respectively. The states are the interpretations shown. (We assume that for all $s \in S$, $V(s) = s$.) The transition function R is depicted by the action-labeled arcs between states.

Notice that the action *Toss* is a nondeterministic, while *Pickup* is deterministic. The path

$$\{\neg H, \neg T\} \xrightarrow{Toss} \{H, \neg T\} \xrightarrow{Pkp} \{\neg H, \neg T\} \xrightarrow{Toss} \{\neg H, T\} \xrightarrow{Pkp} \{\neg H, \neg T\}$$

represents one “causally possible world history.”

4.1.3 Action Query Languages

The signature for an action query language consists of the same two sets of symbols as an action description language: a set $\mathbf{A}$ of action names and a nonempty set $\mathbf{F}$ of fluent names. Two classes of expressions are defined, *axioms* and *queries*. The two classes may be the same.

According to [Lifschitz, 1995], the semantics of an action query language with signature σ is defined by specifying, for every transition system T of σ, every set Γ of axioms, and every query Q, whether Q is a consequence of Γ in T; in symbols, $\Gamma \vdash_T Q$.

An *action language* combines an action description and action query language with same signature. A *domain description* of the combined language is a set D of propositions from the action description language and a set Γ of axioms from the action query language. A query Q is a consequence of a domain description if

$$\Gamma \vdash_T Q$$

where T is the transition system specified by D.

The language $\mathcal{A}_{\text{CL}}$ described in this chapter is purely an action description language. No query language is specified for it. Many choices of query language are possible, including, for example, the query language of $\mathcal{A}$, the query language of $\mathcal{AR}$ [Giunchiglia and Lifschitz, 1995], the two query languages defined in [Lifschitz, 1995], and the language *MPL* of Boutilier and Friedman [1995].

4.2 The Language $\mathcal{A}_{\text{CL}}$

In this section, we define the syntax and semantics of the language $\mathcal{A}_{\text{CL}}$. We assume a fixed signature $\langle \mathbf{A}, \mathbf{F} \rangle$, where $\mathbf{A}$ is a set of action names and $\mathbf{F}$ is a nonempty set of fluent names.

4.2.1 Syntax

By a *fluent literal* we mean an expression either of the form F or $\neg F$, where F is a fluent name. By a *fluent formula* we mean a propositional combination of fluent names, that is, a formula of propositional logic whose atoms are fluent names. We regard the expressions *True* and *False* as abbreviations for $(F \vee \neg F)$ and $(F \wedge \neg F)$, respectively, for some $F \in \mathbf{F}$.

An *effect proposition* is an expression of the form

$$A \textbf{ causes } \psi \textbf{ if } \phi \tag{4.1}$$

where A is an action name, and ϕ and ψ are fluent formulas. If ϕ is *True*, we write simply A **causes** ψ. Intuitively, (4.1) says that in every state in which ϕ is true, doing A causes ψ to be true in every possible next state. The truth of the formula ϕ is a fluent precondition for A causing ψ.

We write the expression

$$\textbf{impossible } A \textbf{ if } \phi \tag{4.2}$$

where A is an action name and ϕ is a fluent formula, as an abbreviation for an effect proposition of the form A **causes** *False* **if** ϕ. Intuitively, (4.2) expresses an action precondition for A, since it implies that A cannot be performed in any state in which ϕ is true. The terminology of action and fluent preconditions is due to Reiter [1991]. The syntax of effect propositions and the method of expressing action preconditions as abbreviations are due to Kartha and Lifschitz [1994].

A *static causal law* is an expression of the form

$$\phi \Rightarrow \psi \tag{4.3}$$

where ϕ and ψ are fluent formulas. As in Chapter 3, when ϕ is *True*, (4.3) will function as a ramification constraint, and when ψ is *False*, (4.3) will function as a qualification constraint.

A *domain description* (or *domain*) in $\mathcal{A}_{\text{CL}}$ is a set of effect propositions and static causal laws.

4.2.2 Semantics

Let D be a domain description in the signature $\langle \mathbf{A}, \mathbf{F} \rangle$, A be an action name in $\mathbf{A}$, and I be an interpretation of $\mathbf{F}$. By the *explicit effect* of A in I according to D—in symbols, $E_D(A, I)$—we designate the set of fluent formulas ψ such that for some fluent formula ϕ, A **causes** ψ **if** ϕ is in D and $I \models \phi$. By $C(D)$ we designate the set of static causal laws in D. Finally, we define

$$C(D)^I = \{\psi : \text{ for some } \phi,\ \phi \Rightarrow \psi \in C(D) \text{ and } I \models \phi\}.$$

Intuitively, $C(D)^I$ is the set of fluent formulas that are explicitly caused to be true in I according to the static causal laws in D.

The domain description D specifies a transition system $\langle S, V, R \rangle$ as follows:

(1) S is the set of interpretations I of $\mathbf{F}$ such that for all $\phi \Rightarrow \psi \in C(D)$, I satisfies the corresponding material conditional $\phi \supset \psi$,

(2) for all states s, $V(s) = s$, and

(3) for all action names A, and states s and s', $s' \in R(A, s)$ if and only if

$$s' = \{L : (s \cap s') \cup E_D(A, s) \cup C(D)^{s'} \models L\}.$$

(i)

$$\begin{tikzcd}
s_1 \arrow[r, "A"] & s_3 \\
s_2 \arrow[r, "A"'] & s_4
\end{tikzcd}$$

(ii)

$$\begin{tikzcd}
s_1 \arrow[r, "A"] \arrow[dr, "A"] & s_3 \\
 & s_4
\end{tikzcd}$$

Figure 4.2: Indefiniteness vs. nondeterminism

It follows from conditions (1) and (2) that states are identified with state descriptions. It follows from condition (3) that every change must be caused according to the effect propositions and static causal laws in D. The fixpoint equation in (3) is essentially that in Definition 4 of Chapter 3. The only differences are that here the explicit effect and static causal laws are extracted from D (by the functions E and C, respectively) rather being given directly as parameters.

The identification of states and state descriptions means that it is impossible to specify in the language $\mathcal{A}_{\text{CL}}$ a transition system in which two distinct states realize the same state description. As a consequence, it is impossible to represent the possibility that an action may have different effects in two real-world states that agree on the values of all of the fluents in $\mathbf{F}$. Indefiniteness about the effects of actions is not expressible in $\mathcal{A}_{\text{CL}}$. On the other hand, nondeterminism is expressible. The distinction between indefiniteness and nondeterminism can be seen in the transitions depicted in Figure 4.2. We suppose that s_1 and s_2 realize the same state description, while s_3 and s_4 realize different state descriptions.

In (i), even though s_1 and s_2 realize the same state description, the action A nevertheless has different possible effects in the two states. Assuming that s_1

and s_2 correspond to distinct real-world states, this means that there are features of the two states which, although they cannot be described in the fluent language **F**, nevertheless are responsible for the different possible effects of A. The identification of states and state descriptions rules out the possibility of such hidden features. So (i) cannot be specified in $\mathcal{A}_{\text{CL}}$. It is, however, possible to specify (ii).

In (i), an agent who knew that he was either in state s_1 or s_2 could, by doing A and observing the results, experimentally determine which of the two states he was in. Later, if he knew that he was in the same state again, he would know precisely what the effect of doing A would be. On the other hand, in (ii) an agent who knew that he was in state s_1 would have nothing to learn by such an experiment. This illustrates the difference between indefiniteness and nondeterminism.[1]

4.3 Examples

In this section we show how several domains, including some of those discussed in the abstract framework of Chapter 3, are encoded in $\mathcal{A}_{\text{CL}}$.

Example 4.1 We begin with a domain, described in [Lifschitz, 1990], in which there are two three-way switches and a light (cf. Examples 3.3 and 3.6).

The signature contains two action names,

$$\mathit{Toggle}(\mathit{Switch1}),\ \ \mathit{Toggle}(\mathit{Switch2})$$

and three fluent names,

$$\mathit{Up}(\mathit{Switch1}),\ \ \mathit{Up}(\mathit{Switch2}),\ \ \mathit{Light}.$$

We specify the domain (in part) by means of schemas, in which s is a meta-variable standing for *Switch1* or *Switch2*.

[1]See [Sandewall, 1992b], page 118, for a closely related discussion. Giunchiglia and Lifschitz [1995] have defined an action description language in which distinct states may realize the same state description.

Domain $D_{4.1}$:

$$(\mathit{Up}(\mathit{Switch1}) \equiv \mathit{Up}(\mathit{Switch2})) \Rightarrow \mathit{Light} \tag{4.4}$$

$$\neg(\mathit{Up}(\mathit{Switch1}) \equiv \mathit{Up}(\mathit{Switch2})) \Rightarrow \neg \mathit{Light} \tag{4.5}$$

$$\mathit{Toggle}(s) \textbf{ causes } \mathit{Up}(s) \textbf{ if } \neg \mathit{Up}(s) \tag{4.6}$$

$$\mathit{Toggle}(s) \textbf{ causes } \neg \mathit{Up}(s) \textbf{ if } \mathit{Up}(s) \tag{4.7}$$

According to (4.4), the switches being in the same position—either both up or both down—is a causally sufficient condition for the light being on. According to (4.5), it is also a causally necessary condition. The effect propositions represented by schemas (4.6) and (4.7) describe the direct effects of toggling the switches.

The domain $D_{4.1}$ specifies the transition system $\langle S, V, R\rangle$, in which S contains four states,

$$\begin{aligned} s_1 &= \{\neg \mathit{Up}(\mathit{Switch1}), \neg \mathit{Up}(\mathit{Switch2}), \mathit{Light}\} \\ s_2 &= \{\neg \mathit{Up}(\mathit{Switch1}), \mathit{Up}(\mathit{Switch2}), \neg \mathit{Light}\} \\ s_3 &= \{\mathit{Up}(\mathit{Switch1}), \neg \mathit{Up}(\mathit{Switch2}), \neg \mathit{Light}\} \\ s_4 &= \{\mathit{Up}(\mathit{Switch1}), \mathit{Up}(\mathit{Switch2}), \mathit{Light}\} \end{aligned}$$

and R is defined as

$$\begin{aligned} R(\mathit{Toggle}(\mathit{Switch1}), s_1) = R(\mathit{Toggle}(\mathit{Switch2}), s_4) = \{s_3\} \\ R(\mathit{Toggle}(\mathit{Switch1}), s_2) = R(\mathit{Toggle}(\mathit{Switch2}), s_3) = \{s_4\} \\ R(\mathit{Toggle}(\mathit{Switch1}), s_3) = R(\mathit{Toggle}(\mathit{Switch2}), s_2) = \{s_1\} \\ R(\mathit{Toggle}(\mathit{Switch1}), s_4) = R(\mathit{Toggle}(\mathit{Switch2}), s_1) = \{s_2\}. \end{aligned}$$

The transition function is pictured in Figure 4.3. The action names *Toggle*(*Switch1*) and *Toggle*(*Switch2*) are abbreviated as T_1 and T_2, respectively. ◇

Example 4.2 The next example is a variant of the so-called Yale Shooting domain [Hanks and McDermott, 1987] (cf. Example 3.5).

$$\begin{tikzcd}
s_1 \arrow[r, shift right, "T_1"'] \arrow[d, shift left, "T_2"] & s_3 \arrow[l, shift right, "T_1"'] \arrow[d, shift right, "T_2"'] \\
s_2 \arrow[u, shift left, "T_2"] \arrow[r, shift left, "T_1"] & s_4 \arrow[l, shift left, "T_1"] \arrow[u, shift right, "T_2"']
\end{tikzcd}$$

Figure 4.3: A transition system for the Two-Switches domain

The language includes four action names,

$$Load,\ Wait,\ Shoot,\ Entice\_to\_Walk$$

and three fluent names,

$$Alive,\ Loaded,\ Walking.$$

Domain $D_{4.2}$**:**

impossible *Load* **if** *Loaded* (4.8)

Load **causes** *Loaded* (4.9)

Shoot **causes** $\neg$*Alive* **if** *Loaded* (4.10)

Shoot **causes** $\neg$*Loaded* (4.11)

Entice_to_Walk **causes** *Walking* (4.12)

$\neg Alive \Rightarrow \neg Walking$ (4.13)

The effect proposition (4.8) says that the gun's not being loaded is an action precondition for loading the gun. The motivating idea is that the action of loading the gun involves putting a bullet in it. (We assume that the gun holds at most one bullet). If the gun is already loaded, this cannot be done. The effect propositions (4.9)–(4.12) describe the explicit effects of the four actions. (Since the action *Wait* has no effects, it is not mentioned.) The effect proposition (4.10) is conditional; it says that the gun's being loaded is a fluent precondition for shooting having the

$$\begin{tikzcd}
s_4 \arrow[loop above, "{W,S,E}"] \arrow[d, "L"] & s_5 \arrow[l, "E"'] \arrow[loop above, "{W,S}"] \arrow[d, "L"'] \\
s_1 \arrow[loop left, "{W,E}"] \arrow[dr, "S"] & s_2 \arrow[l, "E"'] \arrow[loop right, "W"] \arrow[d, "S"] \\
s_3 \arrow[loop below, "W"] \arrow[r, shift right, "S"'] & s_6 \arrow[l, shift right, "L"'] \arrow[loop below, "{W,S}"]
\end{tikzcd}$$

Figure 4.4: A transition system for the Yale Shooting domain

effect of killing the turkey. The final propsosition (4.13) is our familiar static causal law.

The domain $D_{4.2}$ specifies the transition system $\langle S, V, R\rangle$ in which S contains six states,

$$\begin{aligned}
s_1 &= \{Alive, Loaded, Walking\} \\
s_2 &= \{Alive, Loaded, \neg Walking\} \\
s_3 &= \{\neg Alive, Loaded, \neg Walking\} \\
s_4 &= \{Alive, \neg Loaded, Walking\} \\
s_5 &= \{Alive, \neg Loaded, \neg Walking\} \\
s_6 &= \{\neg Alive, \neg Loaded, \neg Walking\}
\end{aligned}$$

and R is defined as pictured in Figure 4.4. (We abbreviate the action names by their first letters.) $\diamond$

Example 4.3 The next example is a variant of the emperor domain of Lin and Reiter [1994] (cf. Example 3.8). The language includes four action names of the form $Paint(x, c)$, and four fluent names of the form $Color(x, c)$, where x and c are meta-variables standing for $Block1$ or $Block2$, and $Yellow$ or Red, respectively.

Domain $D_{4.3}$**:**

$$Color(Block1, Yellow) \wedge Color(Block2, Yellow) \Rightarrow False \tag{4.14}$$

$$True \Rightarrow \neg Color(x, Red) \vee \neg Color(x, Yellow) \tag{4.15}$$

$$Paint(x, c) \textbf{ causes } Color(x, c) \tag{4.16}$$

$$\neg Color(x, Red) \wedge \neg Color(x, Yellow) \Rightarrow False \tag{4.17}$$

The static causal law (4.14) encodes the emperor's decree that there shall be at most one yellow block at a time. Since painting one block does not change the color of the other block, it is written as a qualification constraint. Schema (4.15) encodes, as a ramification constraint, the fact that a single block cannot be both Red and Yellow. Schema (4.16) describes the direct effect of painting a block. Schema (4.17) expresses the assumption that the blocks are always either red or yellow. (We make this assumption only in order to reduce the number of states.)

The domain of $D_{4.3}$ specifies the transition system $\langle S, V, R\rangle$ in which S contains three states. (Again, we use some natural abbreviations.)

$$\begin{aligned}
s_1 &= \{Color(B1, R), \neg Color(B1, Y), Color(B2, R), \neg Color(B2, Y)\}\\
s_2 &= \{Color(B1, R), \neg Color(B1, Y), \neg Color(B2, R), Color(B2, Y)\}\\
s_3 &= \{\neg Color(B1, R), Color(B1, Y), Color(B2, R), \neg Color(B2, Y)\}
\end{aligned}$$

The transition function R is defined as pictured in Figure 4.5. The action name $Paint(Block1, Yellow)$ is abbreviated as Y_1. The other action names are abbreviated similarly. ◇

In each of the preceding domains that we have formalized, all actions have been deterministic. In the following domain, we attempt to formalize a nondeterministic action.

Example 4.4 Let us formalize a simple coin tossing domain. The language contains two action names—*Toss* and *Pickup*—and two fluent names—*Heads* and *Tails*.

$$\begin{tikzcd}
s_1 \arrow[loop above, "{R_1, R_2}"] \arrow[r, shift left, "Y_2"] \arrow[d, shift left, "Y_1"] & s_2 \arrow[loop above, "{R_1, Y_2}"] \arrow[l, shift left, "R_2"] \\
s_3 \arrow[loop below, "{R_2, Y_1}"] \arrow[u, shift left, "R_1"] &
\end{tikzcd}$$

Figure 4.5: A transition system for the Emperor domain

Domain $D_{4.4}$**:**

$$\mathit{True} \Rightarrow \neg \mathit{Heads} \vee \neg \mathit{Tails} \tag{4.18}$$

$$\textbf{impossible } \mathit{Toss} \textbf{ if } \mathit{Heads} \vee \mathit{Tails} \tag{4.19}$$

$$\mathit{Toss} \textbf{ causes } \mathit{Heads} \vee \mathit{Tails} \tag{4.20}$$

$$\textbf{impossible } \mathit{PickUp} \textbf{ if } \neg \mathit{Heads} \wedge \neg \mathit{Tails} \tag{4.21}$$

$$\mathit{PickUp} \textbf{ causes } \neg \mathit{Heads} \wedge \neg \mathit{Tails} \tag{4.22}$$

The proposition (4.18) expresses, as a ramification constraint, the fact the the coin cannot lie both heads and tails. According to (4.19), it is impossible to toss a coin if it is lying either heads or tails; the idea is that the coin must be in hand. According to (4.20), tossing the coin causes it to lie either heads or tails. The disjunctive effect is intended to express the nondeterminism of coin tossing (but see below). According to (4.21), it is impossible to pick up a coin that is already in hand. Finally, (4.22) describes the explicit effect of picking up the coin.

The domain $D_{4.4}$ specifies the transition system pictured in Figure 4.1. The action *Toss*, as we previously remarked, is nondeterministic. Notice, however, that the nondeterminism of *Toss* arises here for a rather curious reason. Consider, for example, the transition

$$\{\neg H, \neg T\} \xrightarrow{Toss} \{H, \neg T\}.$$

The correctness of this transition can be checked as follows. By inertia, we have $\neg T$ and by the explicit effect of *Toss* we have $H \vee T$. (Also, by (4.18t) we have $\neg H \vee \neg T$.) Since these premises entail exactly the literals H and $\neg T$, the transition is correct. The transition

$$\{\neg H, \neg T\} \xrightarrow{Toss} \{\neg H, T\}$$

can be checked by similar reasoning. We see, therefore, that the nondeterminism of *Toss* arises from the combination of its disjunctive effect $H \vee T$ and inertia.

What is curious about this is that when the state of the coin changes from neither H nor T being true before the toss to, for example, H being true afterwards, it is clear that the reason for $\neg T$ being true after the toss is something other than inertia. It is, indeed, true that $\neg T$ is true both before and after the toss. But, intuitively, the reason that $\neg T$ is true after the toss is not inertia, but rather that H is true, from which, by (4.18), $\neg T$ follows.

We would like to specify that the fluent $\neg T$ is not preserved by inertia, but in the language $\mathcal{A}_{\rm CL}$ there is no way to do so. Moreover, if $\mathcal{A}_{\rm CL}$ were modified to make this possible, it would then be impossible to express the nondeterminism of coin tossing by the disjunctive effect $H \vee T$. Apparently, $\mathcal{A}_{\rm CL}$ would have to be modified again to provide some other means for expressing nondeterminism.[2] In

[2]In fact, even without altering inertia, it can be seen that the use of disjunctive effects alone is an inadequate means for expressing nondeterminism. Imagine, for example, that we view tossing the coin as an action that includes picking up the coin as a part, so that the coin can be tossed in each of its three possible states. (Let us now view the state of the coin in which it is lying neither heads nor tails as one in which the coin is balanced on its edge.) It is easy to see that, except in the state in which the coin is balanced on its edge, the disjunctive effect $H \vee T$ does not yield nondeterminism. Instead, in each of the states in which H or T is already true, tossing the coin leaves the state unchanged.

Similarly, if we were to attempt to model coin tossing in a language with only a single fluent, say, *Heads*, it is clear that writing

$$Toss \ \textbf{causes} \ Heads \vee \neg Heads$$

(essentially, *Toss* **causes** *True*) would not yield the desired results.

In natural language, we can indeed express the nondeterminism of coin tossing by expressions such as "Toss causes heads or tails" and even "Toss causes heads or not heads." But this works, we suggest, only because of certain "conversational implicatures" [Grice, 1989] that these expressions have in addition to their logical content; for example, in the latter case, the implicatures are that "Toss possibly causes heads," and "Toss possibly causes not heads." That these are indeed fact, extensions of both kinds, as well as others, are possible. However, we will not pursue this course here. Instead, in the next chapter, we will define an alternative formalism that is mathematically simpler and more expressive than the language $\mathcal{A}_{\mathrm{CL}}$. In this formalism, it will be possible to address the issues of inertia and nondeterminism, among others, more directly. $\diamond$

implicatures and not part of the logical content is suggested by the fact that they are lost when the phrase "heads or not heads" is replaced by other logically equivalent expressions (e.g., by *True* or by "if heads then heads"). The key to representing nondeterminism is to explicitly represent the implicatures of such natural language expressions. We will propose one way of doing this when we again consider the formalization of coin tossing in Chapter 7 (Example 7.2).

Chapter 5

The Language of Causal Theories

In the previous chapter, a domain description was taken to specify a transition system, and, thereby, a set of causally possible world histories (represented by the set of paths through the transition system). The possibility of representing the causally possible world histories in this way rests on three simplifying assumptions. The first assumption is that changes occur only when actions are performed. This assumption enables us to represent a world history by a sequence of alternating states and actions, e.g.,

$$s_1 \xrightarrow{A_1} s_2 \xrightarrow{A_2} s_3 \xrightarrow{A_3} s_4 \xrightarrow{A_4} s_5 .$$

(If one of the actions is *Wait*—an action with the empty explicit effect—then this is not a simplifying assumption.) The second assumption is that actions are not performed concurrently. The third assumption is that for each s_{i+1} in such a sequence, it is unnecessary to look beyond the facts represented in a single transition

$$s_i \xrightarrow{A_i} s_{i+1}$$

in order to find sufficient conditions for the facts in s_{i+1} to be caused. In $\mathcal{A}_{\rm CL}$, effect propositions refer to conditions in s_i and A_i, while static causal laws refer to conditions in s_{i+1}. In the remainder of this dissertation, we will abandon these three assumptions and will adopt a more general framework according to which a set of propositions in an action description language is understood to directly specify the set of causally possible world histories.[1]

We will define the language of causal theories as an extension of propositional logic. (Lifschitz [1997] has shown how to reformulate the central definitions for the case of predicate logic.) We will delay until Chapter 7 the discussion of how action domains are formalized in the language of causal theories. In this chapter and the next, we will be concerned solely with the formalism itself, the motivating ideas behind it, and some of its mathematical properties.

5.1 Informal Motivation

The world is governed by causal laws. The true causal laws determine which world histories are causally possible. Intuitively, a causally possible world history is one that conforms to the true causal laws, i.e., one in which every fact that is caused obtains. We will assume the *principle of universal causation*, according to which every fact that obtains is caused.[2] Accordingly, we can say that a causally possible world history is one in which exactly the facts that obtain are caused.

Now suppose that D is a complete description of the conditions under which facts are caused. (Intuitively, this means that whenever a fact is caused in a world history, D says that this is so.) In this case, we can say that a causally possible world history is one in which the facts that obtain are exactly those that are caused according to D. This is the key to understanding the formal definitions that follow.

Notice that we make two assumptions: (1) the principle of universal causation

[1]It is possible to generalize the definition of a transition system so that the transition function, instead of mapping a single action and a state to a set of states, maps a set of actions and a state to a set of states. Relative to this more general notion of a transition system, only the last of the above-mentioned assumptions remains as a simplification.

[2]This philosophical commitment is rewarded by mathematical simplicity in the main definition of causal theories. Moreover, as we will see in Chapter 7, in applications it is easily relaxed.

and (2) the completeness of D.

On the other hand, notice that we make no assumption about *where* in a causally possible world history the sufficient conditions for facts being caused are to be found. We assume only that they can be found somewhere within the world history itself. Thus, we allow, for example, the possibility that future facts may be sufficient conditions for facts in the past being caused. We also allow the possibility that a fact may be a sufficient condition (or part of one) for itself being caused. This last mentioned possibility turns out to play an important role in in formalizing action domains as causal theories. In effect, it provides a means of exempting facts from the principle of universal causation.

5.2 Syntax

We begin with a standard language of propositional logic, whose signature is given by a nonempty set of atoms. (In application to formalizing action domains (Chapter 7), the atoms will be taken to represent propositions about the values of fluents and the occurrences of actions at specific times.) By a *literal* we mean either A or $\neg A$, where A is an atom. We use the expressions *True* and *False* to stand for $(A \vee \neg A)$ and $(A \wedge \neg A)$, respectively, for some atom A.

By a *causal law* we mean an expression of the form

$$\phi \Rightarrow \psi \tag{5.1}$$

where ϕ and ψ are formulas of the underlying propositonal language. By a *causal theory* we mean a set of causal laws.

By the *antecedent* and *consequent* of (5.1), we mean the formulas ϕ and ψ, respectively. Note that (5.1) is not the material conditional, $\phi \supset \psi$. The intended reading of (5.1) is the following:

(i) *Necessarily, if ϕ then the fact that ψ is caused.*

Often, but not always, a stronger reading is also appropriate, namely:

(ii) *The fact that* ϕ causes *the fact that* ψ.

The term "causal law" is suggested by reading (ii).

Recalling our earlier discussion of sentences of forms (i) and (ii) in Section 1.2, notice that while sentence (ii) asserts the existence of a causal relationship between the facts associated with ϕ and ψ, sentence (i) does not. Intuitively, however, this difference in meaning is irrelevant for determining which facts are caused in a world history and (so also) which world histories are causally possible. Since the semantics of causal theories is limited to determining the causally possible world histories (we will not define truth conditions for causal laws, or the conditions under which causal laws are entailed), it supports readings (i) and (ii) equally well.[3]

It would be possible to abandon reading (ii) entirely in favor of the weaker reading (i)—but not *vice versa*, since sometimes we will write causal laws for which only reading (i) is intended. However, the advantage of maintaining reading (ii) is that it is often what we most naturally wish to say. In such cases, given the limited scope of the semantics of causal theories, there is no reason why we should not regard ourselves as having said it.[4]

5.3 Semantics

We identify an interpretation I for a propositional language with the set of literals L such that $I \models L$. Here, as throughout this dissertation, we will use the symbol L to stand exclusively for literals.

[3]In Appendix A, we will define an encoding of causal laws in a modal language for which truth conditions and entailment are defined. In this framework, only reading (i) (for the encodings) is supported by the formal semantics; reading (ii) will be abandoned.

[4]Our two readings for causal laws have the same sort of justification as our readings of effect propositions and static causal laws in $\mathcal{A}_{\rm CL}$. In $\mathcal{A}_{\rm CL}$ the readings are justified by the intuitive appropriateness of the transition systems that sets of such propositions are taken to specify. Readings (i) and (ii) for causal laws are justified by the intuitive appropriateness of the set of "possible world histories" that causal theories are taken to specify.

Let D be a causal theory, and I be an interpretation. We define

$$D^I = \{\psi : \text{ for some } \phi,\ \phi \Rightarrow \psi \in D \text{ and } I \models \phi\}.$$

That is, D^I is the set of consequents of all causal laws in D whose antecedents are true in I. Intuitively, whether we adopt reading (i) or (ii), D^I entails exactly the formulas that are caused to be true in I according to D. (This assumes, as is our intuition, that the set of formulas caused to be true in I is closed under entailment.)

We are now ready to state the main semantic definition.

Main Definition. Let D be a causal theory, and I be an interpretation. We say that I is *causally explained* according to D if I is the unique model of D^I.[5]

Intuitively, when D describes an action domain, the causally explained interpretations according to D correspond to the causally possible world histories.

Proposition 5.1 *Let D be a causal theory, and I be an interpretation. The following propositions are equivalent.*

(i) I is causally explained according to D.

(ii) For every formula ϕ, $I \models \phi$ if and only if $D^I \models \phi$.

(iii) For every literal L, $I \models L$ if and only if $D^I \models L$.

Proof. To show that (i) implies (ii), suppose (i). Then I is the unique model of D^I. Let ϕ be an arbitrary formula. There are two cases: (i) $I \models \phi$, in which case for every I' such that $I' \models D^I$, $I' \models \phi$, so $D^I \models \phi$, and (ii) $I \not\models \phi$, in which case there is an I' such that $I' \models D^I$ and $I' \not\models \phi$, so $D^I \not\models \phi$. We conclude that $I \models \phi$ if and only if $D^I \models \phi$. It is obvious that (ii) implies (iii). To show that (iii) implies

[5]Heinrich Herre and Gerd Wegner [1996] have independently defined the notions of a *generalized logic program* and a *strongly supported model*. The class of generalized logic programs turns out to be essentially equivalent to the class of causal theories, and the notion of a strongly supported model turns out to be equivalent to the notion of a causally explained interpretation. These equivalences were discovered at the Dagstuhl Seminar on Disjunctive Logic Programming and Databases: Nonmonotonic Aspects, 1996.

(i), suppose (iii). We will first show that D^I has no other model than I. Suppose that there is an interpretation I' such that $I \neq I'$ and $I' \models D^I$. Let L be a literal such that $I \models L$, but $I' \not\models L$. It follows that $D^I \not\models L$, which contradicts (iii). We conclude that D^I has no other model than I. That D^I has a model is clear from (iii), since otherwise D^I would entail every literal, including literals that are not true in I. We conclude that I is the unique model of D^I, from which (i) follows by definition. $\square$

In view of the equivalence of (i) and (ii) in Proposition 5.1, we can now say that I is causally explained according to D if and only if exactly the formulas that are true in I are caused to be true in I according to D.

Corollary 5.1 *Let D be a causal theory. An interpretation I is causally explained according to D if and only if*

$$I = \{L : D^I \models L\}. \tag{5.2}$$

Proof. Notice that (5.2) is simply (iii) of Proposition 5.1, written in a different form. $\square$

Recall that Definition 4 of Chapter 3 contained the following fixpoint equation

$$S' = \{L : (S \cap S') \cup E \cup D^{S'} \models L\}. \tag{5.3}$$

In essence, (5.2) can be obtained from (5.3) by dropping the premises $(S \cap S')$ and E, that is, the premises for inertia and the explicit effect, respectively. We will see in Chapter 7 that in the framework of causal theories, these premises need not be built in, since both inertia and the explicit effects of actions can be expressed by causal laws.

Given a causal theory D and a formula ϕ, we say that ϕ is a *consequence* of D if ϕ is true in every interpretation that is causally explained according to

D. Intuitively, when D describes an action domain, the consequences of D are the formulas that are true in all causally possible world histories.

In classical logic, adding a new axiom to a theory can never increase its set of models or diminish its set of consequences. The set of models is monotonically non-increasing, and the set of consequences is monotonically non-decreasing. In the language of causal theories, by contrast, adding a new causal law can increase the set of interpretations that are causally explained, and thereby diminish the set of consequences. Thus, the consequence relation for the language of causal theories is nonmonotonic. This is illustrated in the following section.

5.4 Examples

Example 5.1 Let $D_{5.1}$ be the causal theory (in the language with only the atom a) consisting of the causal law

$$a \Rightarrow a\,. \tag{5.4}$$

Take $I = \{a\}$. Notice that $D^I_{5.1} = \{a\}$. Since I is the unique model of $D^I_{5.1}$, I is causally explained according to $D_{5.1}$. No other interpretation is causally explained. Therefore, a is a consequence of $D_{5.1}$. $\Diamond$

Example 5.2 Now, let $D_{5.2}$ be the causal theory obtained by adding to $D_{5.1}$ the causal law

$$\neg a \Rightarrow \neg a\,. \tag{5.5}$$

One easily checks that both $\{a\}$ and $\{\neg a\}$ are causally explained according to $D_{5.2}$. Thus, a is not a consequence of $D_{5.2}$. This shows that the consequence relation for causal theories is nonmonotonic. $\Diamond$

Example 5.3 Let $D_{5.3}$ be the causal theory (in the language with the atoms a and b) consisting of the causal laws

$$\mathit{True} \Rightarrow a \supset b$$

$$b \Rightarrow a.$$

Take $I = \{a, b\}$. Notice that $D^I_{5.3} = \{a \supset b, a\}$. Since I is the unique model of $D^I_{5.3}$, I is causally explained according to $D_{5.3}$. No other interpretation is causally explained. Therefore, $a \wedge b$ is a consequence of $D_{5.3}$. $\diamond$

Example 5.4 Let $D_{5.4}$ be the causal theory (in the language with the atoms a and b) consisting of the causal laws

$$a \wedge b \Rightarrow b$$

$$\neg a \wedge \neg b \Rightarrow \neg b$$

$$a \Rightarrow a$$

$$\neg a \Rightarrow \neg a.$$

The reader may verify that exactly the following interpretations are causally explained according to $D_{5.4}$.

$$\{a, b\} \quad \{\neg a, \neg b\}$$

Thus, the formula $(a \equiv b)$ is a consequence of $D_{5.4}$. $\diamond$

With the exception of $D_{5.3}$, in each of the preceding causal theories the antecedent of every causal law is a conjunction of literals, and the consequent of every causal law is a literal. Causal theories of this form correspond syntactically to the class of basic logic programs [Lifschitz, 1996]. We will investigate this subclass of causal theories in next chapter.

5.5 Definitional Extension

In this section, we examine the role of explicit definitions in the language of causal theories.

5.5.1 Explicit Definitions

Let F be an atom. By an *explicit definition of* F, we mean a causal law of the form

$$True \Rightarrow F \equiv \psi$$

where ψ is a formula that does not contain F.

Let D be a causal theory with the signature (set of atoms) A. A causal theory D' is said to be a *definitional extension* of D if the signature of D' is $A \cup \{F\}$, for some atom F that is not in A, and

$$D' = D \cup \{True \Rightarrow F \equiv \psi\}$$

where $True \Rightarrow F \equiv \psi$ is an explicit definition of F.

Let D and D' be causal theories. We say that D' is a *conservative extension* of D if the signature of the language of D is a subset of the signature of the language of D', every causal law in D belongs to D', and for all formulas ϕ in the language of D, ϕ is a consequence of D' if and only if ϕ is a consequence of D.

Proposition 5.2 *Let D and D' be causal theories. If D' is a definitional extension of D then D' is a conservative extension of D.*

5.5.2 Replacement

The following proposition states a simple replacement property which causal theories inherit from classical propositional logic.

Proposition 5.3 *Let D be a causal theory that contains the causal law*

$$True \Rightarrow \phi \equiv \psi. \tag{5.6}$$

Let D' be a causal theory that is obtained by replacing zero or more occurrences of ψ by ϕ in D, excluding occurrences in (5.6). An interpretation I is causally explained according to D if and only if I is causally explained according to D'.

Example 5.5 Recall the causal theory $D_{5.4}$ above. By Proposition 5.2, we know that the following causal theory $D_{5.5}$ (in a language with the atoms a, b, and c) is a conservative extension of $D_{5.4}$.

$$a \wedge b \Rightarrow b$$
$$\neg a \wedge \neg b \Rightarrow \neg b$$
$$a \Rightarrow a$$
$$\neg a \Rightarrow \neg a$$
$$\mathit{True} \Rightarrow c \equiv \neg b$$

The reader may verify that exactly the following interpretations are causally explained according to $D_{5.5}$.

$$\{a, b, \neg c\}$$
$$\{\neg a, \neg b, c\}$$

Notice that the formula $(a \equiv b)$, which, as we saw above, is a consequence of $D_{5.4}$, is also a consequence of $D_{5.5}$.

By Proposition 5.3, we know that the causally explained interpretations, and thus also the consequences, of $D_{5.5}$ are unchanged by the replacements in the second causal law below.

$$a \wedge b \Rightarrow b$$
$$\neg a \wedge c \Rightarrow c$$
$$a \Rightarrow a$$
$$\neg a \Rightarrow \neg a$$
$$\mathit{True} \Rightarrow c \equiv \neg b$$

$\diamond$

According to Proposition 5.2, we can extend the underlying language of a causal theory D by a new atom F and add an explicit definition for F to D

without changing the consequences of D in the original language. According to Proposition 5.3, we can also freely replace the *definiens* of F by F anywhere in D (except in the explicit definition itself) without affecting the causally explained interpretations. Together, these propositions allow us to introduce abbreviations for complex formulas, *via* explicit definitions, and to use them to simplify causal theories. The importance of conservative extension and replacement theorems for formalizing action domains was recognized in [Kartha and Lifschitz, 1994].

5.5.3 Proofs of Propositions 5.2 and 5.3

Lemma 5.1 *Let F be an atom, A be a set of atoms, and D and D' be causal theories in languages with the signatures A and $A \cup \{F\}$, respectively. Furthermore, suppose that*

$$D' = D \cup \{\mathit{True} \Rightarrow F \equiv \psi\}$$

where $\mathit{True} \Rightarrow F \equiv \psi$ is an explicit definition of F. Let I be an interpretation of the language of D, and let I' be defined as follows:

$$I' = I \cup \{F\},\ \text{if } I \models \psi$$
$$I' = I \cup \{\neg F\},\ \text{if } I \not\models \psi.$$

If I is causally explained according to D, then I' is causally explained according to D'.

Proof. Notice that for every interpretation I of the language of D, $D'^{I'} = D^I \cup \{F \equiv \psi\}$. Suppose that I is causally explained according to D. Then I is the unique model of D^I. By the definition of I', it follows that I' is the unique model of $D^I \cup \{F \equiv \psi\}$, and thus also the unique model of $D'^{I'}$. Therefore, I' is causally explained according to D'. □

Lemma 5.2 *Let F, A, D and D' be as specified in Lemma 5.1. Let I' be an interpretation of the language of D', and let I be defined as follows:*

$$I = I' \setminus \{F, \neg F\}.$$

If I' is causally explained according to D' then I is causally explained according to D.

Proof. Notice that for every interpretation I' of the language of D', $D'^{I'} = D^I \cup \{F \equiv \psi\}$. Suppose that I' is causally explained according to D'. Then I' is the unique model of $D'^{I'}$, and thus also of $D^I \cup \{F \equiv \psi\}$. It follows that I is the unique model of D^I. Therefore, I is causally explained according to D. $\Box$

Proof (of Proposition 5.2). Let D' be a definitional extension of D. Suppose that the signature of the language of D is A, the signature of D' is $A \cup \{F\}$, and $D' = D \cup \{\mathit{True} \Rightarrow F \equiv \psi\}$. It is clear that the signature of the language of D is a subset of the signature of the language of D' and that every causal law in D belongs to D'. Thus, it remains only to show that for all formulas ϕ in the language of D, ϕ is a consequence of D' if and only if ϕ is a consequence of D. For the left-to-right direction, suppose that ϕ is not a consequence of D. Then there is a interpretation I that is causally explained according to D such that $I \not\models \phi$. It follows that the interpretation I', as defined in Lemma 5.1, is causally explained according to D'. Since I' is identical to I, except for assigning a value to F, it follows that $I' \not\models \phi$. Therefore, ϕ is not a consequence of D'. For the right-to-left direction, suppose that ϕ is not a consequence of D'. Then there is a interpretation I' that is causally explained according to D' such that $I' \not\models \phi$. It follows that the interpretation I as defined in Lemma 5.2, is causally explained according to D. Since I' is identical to I, except for assigning a value to F, it follows that $I \not\models \phi$. Therefore, ϕ is not a consequence of D. $\Box$

Proof (of Proposition 5.3). To begin, let $D_0 = D \setminus \{\mathit{True} \Rightarrow \phi \equiv \psi\}$, and let $D'_0 = D' \setminus \{\mathit{True} \Rightarrow \phi \equiv \psi\}$. So D'_0 is obtained from D_0 by replacing zero or more occurrences of ψ by ϕ. Let I be an interpretation. Clearly, $\phi \equiv \psi$ is contained in both D^I and D'^I. So if $I \not\models \phi \equiv \psi$ then I is not causally explained according to either D or D', and we are done. So suppose $I \models \phi \equiv \psi$. We will show that D^I and D'^I have the same models. Note that $D^I = (D_0 \cup \{\mathit{True} \Rightarrow \phi \equiv \psi\})^I = D_0^I \cup \{\phi \equiv \psi\}$, while $D'^I = (D'_0 \cup \{\mathit{True} \Rightarrow \phi \equiv \psi\})^I = D_0'^I \cup \{\phi \equiv \psi\}$. Since $I \models \phi \equiv \psi$, D_0^I and $D_0'^I$ are the same except that $D_0'^I$ contains ϕ in place of ψ in zero or more of its occurrences in D_0^I. It follows by the replacement of equivalents in propositional logic that D^I and D'^I have the same models. Therefore, I is the unique model of D^I if and only if I is the unique model of D'^I. Hence, I is causally explained according to D if and only if I is causally explained according to D'. $\Box$

5.6 An Embedding in Default Logic

In this section we describe an embedding, suggested by Hudson Turner (personal communication), of causal theories into default logic [Reiter, 1980]. The reader may skip this section (and the related Section 6.3) without loss of continuity.

In defining the syntax and semantics of default logic, we will be concerned only with propositional default theories. The definitions given here follow the style of definition in [Gelfond *et al.*, 1991a].

A *default* is an expression of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma} \tag{5.7}$$

where α, $\beta_1, \ldots, \beta_m$ $(m \geq 0)$, and γ are formulas of propositional logic. The components of a default are named as follows: α is called the *prerequisite*, $\beta_1, \ldots, \beta_m$ are called the *justifications*, and γ is called the *consequent*. If $m = 0$, (5.7) is an inference rule. If α is *True*, it may be dropped. If both $m = 0$ and α is *True*, (5.7) will be identified with γ. A *default theory* is a set of defaults.

Given a set X of formulas, by $Cn(X)$ we mean the set of formulas ϕ such that $X \models \phi$. $Cn(X)$ is the smallest set that contains X and is closed under propositional logic.

Given a set R of inference rules, by $Cn^*(R)$ we mean the smallest set X of formulas such that

(i) $X = Cn(X)$, and

(ii) for every $\frac{\alpha}{\gamma} \in R$, if $\alpha \in X$ then $\gamma \in X$.

$Cn^*(R)$ is the smallest set that is closed under propositional logic and the rules in R.

Let T be a default theory and E be a set of formulas. The reduct of T with respect to E (in symbols, T^E) is defined as

$$T^E = \left\{ \frac{\alpha}{\gamma} : \frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma} \in T \text{ and for all } i \ (1 \leq i \leq m),\ \neg\beta_i \notin E \right\}.$$

Notice that T^E is a set of inference rules. The set E is called an *extension* for T if $E = Cn^*(T^E)$.

We are now ready to define an embedding of causal theories into default logic. Let D be a causal theory. We define the corresponding default theory $T(D)$ as follows:

$$T(D) = \left\{ \frac{:\phi}{\psi} : \phi \Rightarrow \psi \in D \right\}.$$

Proposition 5.4 *Given a causal theory D, an interpretation I is causally explained according to D if and only if $Cn(I)$ is an extension for $T(D)$.*

The following lemma is used in the proof of Proposition 5.4.

Lemma 5.3 *For every causal theory D and interpretation I, $D^I = T(D)^{Cn(I)}$.*

Proof. For the left-to-right-direction, suppose that $\psi \in D^I$. Then for some $\phi \Rightarrow \psi \in D$, $I \models \phi$. We know that $\frac{:\phi}{\psi} \in T(D)$. Since $I \models \phi$, we also know that $\neg\phi \notin Cn(I)$. Therefore, $\frac{True}{\psi}$ (equivalently, ψ) is in $T(D)^{Cn(I)}$. For the right-to-left-direction, suppose that ψ (equivalently, $\frac{True}{\psi}$) is in $T(D)^{Cn(I)}$. Then for some

$\frac{:\phi}{\psi} \in T(D)$, $\neg\phi \notin Cn(I)$. We know that $\phi \Rightarrow \psi \in D$. Since $\neg\phi \notin Cn(I)$, $I \models \phi$. Therefore, $\psi \in D^I$. $\square$

Proof (of Proposition 5.4). Let I be an interpretation. I is causally explained according to D iff I is the unique model of D^I iff (by Lemma 5.3) I is the unique model of $T(D)^{Cn(I)}$ iff $Cn(I) = Cn(T(D)^{Cn(I)})$ iff $Cn(I) = Cn^*(T(D)^{Cn(I)})$ iff $Cn(I)$ is an extension for $T(D)$. $\square$

Let X be a set of formulas. We say that X is *consistent* if there is no formula ϕ such that X contains both ϕ and $\neg\phi$. We say that X is is *complete* if for every formula ϕ, X contains either ϕ or $\neg\phi$.

Corollary 5.2 *The interpretations that are causally explained according to a causal theory D correspond one-to-one to the complete, consistent extensions for $T(D)$.*

Proof. An extension E for $T(D)$ is complete and consistent if and only if for some interpretation I, $E = Cn(I)$. Thus, the corollary follows by Proposition 5.4. $\square$

In the next chapter, we will see that in the special case in which for every $\phi \Rightarrow \psi$ in D, ψ is a literal and ϕ is a conjunction of literals, there is a similar embedding into extended programs [Gelfond and Lifschitz, 1990].

Chapter 6

Objective Logic Programs

In this chapter we investigate the class of causal theories in which the antecedent of every causal law is a conjunction of literals and the consequent of every causal law is a literal. Syntactically, such literal-oriented causal theories correspond to the class of *basic logic programs* [Lifschitz, 1996]. Semantically, however, they are different. Informally speaking, basic programs represent possible *belief states* which are partial (in that they do not necessarily assign a value to every atom), while causal theories represent possible *world histories* which are total. In this sense, basic programs are "subjective" and causal theories are "objective." In light of this, we refer to the literal-oriented subclass of causal theories as "objective programs."

Objective programs turn out to have close semantic connections to the classical semantics for positive programs and the completion semantics for normal programs. Indeed, the semantics of objective programs is the natural generalization of the completion semantics [Clark, 1978] to the class of programs that allow negation to occur in the heads as well as the bodies of rules. In Chapter 9 this connection to the completion semantics will be exploited to provide a automated approach to query answering and planning with respect to causal theories that belong to the class of objective programs.

6.1 The Language of Objective Programs

In this section we define the syntax and semantics of objective programs.

6.1.1 Syntax

As in the case of general causal theories, we begin with a standard language of propositional logic, whose signature is given by a nonempty set of atoms.

An *objective program* is a set of rules of the form

$$L_0 \leftarrow L_1, \ldots, L_n \tag{6.1}$$

where $n \geq 0$ and for all i, $0 \leq i \leq n$, L_i is a literal. By the *head* of the rule (6.1), we mean the literal L_0. By the *body*, we mean the set of literals $\{L_1, \ldots, L_n\}$.

An objective program is simply a causal theory written in a conventional logic programming notation. Accordingly, we will also use the term "objective program" to refer to a set of causal laws of the form

$$L_1 \wedge \ldots \wedge L_n \Rightarrow L_0$$

where $n \geq 0$ and for all i, $0 \leq i \leq n$, L_i is a literal.

We will have reason to discuss the two subclasses of the class of objective programs, namely, the class of positive programs and the class of normal programs. The following definitions are standard in the logic programming literature.

By a *positive program* we mean a set of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_n$$

where $n \geq 0$ and for all i, $0 \leq i \leq n$, A_i is an atom.

By a *normal program* we mean a set of rules of the form

$$A_0 \leftarrow L_1, \ldots, L_n$$

where A_0 is an atom, and for all i, $1 \leq i \leq n$, L_i is a literal.

We will also have reason to discuss the class of extended programs [Gelfond and Lifschitz, 1990]. An *extended program* is a set of rules of the form

$$L_0 \leftarrow L_1, \ldots, L_k, not\ L_{k+1}, \ldots, not\ L_n. \tag{6.2}$$

Syntactically, an objective program is an extended program that does not contain *not*, the symbol for default negation (also called negation as failure).

6.1.2 Semantics

Objective programs are a special case of causal theories. Therefore, their semantics is already known. Nevertheless, it will be useful to also define the semantics of objective programs independently, since their restricted syntax makes possible certain simplifications.

As before, we identify an interpretation I with the set of literals L such that $I \models L$. We continue to use the symbol L to stand exclusively for literals.

Let Π be an objective program, and I be an interpretation. We define the set of literals *supported* in I by Π, in symbols Π^I, as

$$\Pi^I = \{L : \text{ for some } B,\ L \leftarrow B \in \Pi \text{ and } B \subseteq I\}.$$

Let I be an interpretation. We say that I is a *model* of Π if

$$\Pi^I \subseteq I.$$

We say that I is *supported* by Π if

$$I \subseteq \Pi^I.$$

Finally, we say that I is a *supported model* of Π if

$$I = \Pi^I.$$

We say that a formula ϕ is a *consequence* of Π if ϕ is contained in every supported model of Π.[1]

Recall the definition

$$D^I = \{\psi : \text{ for some } \phi,\ \phi \Rightarrow \psi \in D \text{ and } I \models \phi\}$$

from Section 5.3. Since B is a set of literals, I is a model of B if and only if $B \subseteq I$. Thus, the definition of Π^I is a specialization of the definition of D^I to the case of objective programs.

Recall that an interpretation I is said to be causally explained according to a causal theory D just in case I is the unique model of D^I. Since Π^I is a set of literals, I is the unique model of Π^I just in case $I = \Pi^I$. Thus, the definition of a supported model is specialization of the definition of a causally explained interpretation to the case of objective programs.

Finally, the definition of consequence for objective programs is an obvious specialization of the notion of consequence for causal theories generally.[2]

Example 6.1 The following program $\Pi_{6.1}$ is a notational variant of the causal theory of Example 5.4.

$$\begin{aligned} b &\leftarrow a, b \\ \neg b &\leftarrow \neg a, \neg b \\ a &\leftarrow a \\ \neg a &\leftarrow \neg a \end{aligned}$$

Let $I = \{a, b\}$. Notice that $\Pi^I_{6.1} = \{a, b\}$. Since $I = \Pi^I_{6.1}$, I is a supported model of $\Pi_{6.1}$. Now let $I' = \{\neg a, \neg b\}$. Notice that $\Pi^{I'}_{6.1} = \{\neg a, \neg b\}$. Since $I' = \Pi^{I'}_{6.1}$, I' is

[1]The notion of a supported model has been previously defined for the class of normal logic programs by Apt, Blair, and Walker [1988]. According to their definition, a normal program Π is supported by an interpretation I if for every atom A in I there exists a rule $A \leftarrow L_1, \ldots, L_n$ in Π such that $I \models L_1 \wedge \ldots \wedge L_n$. For the class of objective programs, it is natural to modify this definition to say that for every literal L in I there exists a rule $L \leftarrow L_1, \ldots, L_n$ in Π such that $I \models L_1 \wedge \ldots \wedge L_n$. This is equivalent to the definition given above.

[2]Normally, the consequence relation for a logic program is defined to hold only between a program and a literal, rather than, as here, between a program and a (propostional) formula.

also a supported model of $\Pi_{6.1}$. There are no other supported models. Therefore, $(a \equiv b)$ is a consequence of $\Pi_{6.1}$. $\diamond$

6.2 Literal Completion

The original semantics for logic programs with negation was the completion semantics for normal programs [Clark, 1978]. In this section, we show that the supported model semantics for objective programs corresponds to a straightforward generalization of completion semantics for the case of programs in which negation is allowed to occur in the heads of rules.

We are interested only in the propositional case, where the completion semantics for normal programs may be defined as follows. Let Π be a normal program. We say that Π is *completable* if for every atom A there are finitely many rules in Π with the head A. Let Π be a completable normal program. By the *completion* of Π (in symbols, $comp(\Pi)$) we mean the set of equivalences

$$A \equiv B_1 \vee B_2 \vee \ldots \vee B_k \tag{6.3}$$

for each atom A, where

$$\begin{aligned} A &\leftarrow B_1 \\ A &\leftarrow B_2 \\ &\ldots \\ A &\leftarrow B_k \end{aligned}$$

are the rules in Π with the head A. If there are no such rules $(k = 0)$, (6.3) stands for $A \equiv$ *False*.

According to the completion semantics [Clark, 1978], the models of a normal program Π are the models of $comp(\Pi)$.

The completion procedure for normal programs can be extended to the class of objective programs as follows. We first extend the notion of a completable program to the class of objective programs as follows. Let Π be an objective program. We say that Π is *completable* if for every literal L there are finitely many rules in Π with the head L. Let Π be a completable objective program. By the *literal completion* of Π (in symbols, $lcomp(\Pi)$) we mean the set of formulas

$$L \equiv B_1 \vee B_2 \vee \ldots \vee B_k \tag{6.4}$$

for each literal L, where

$$\begin{aligned} L &\leftarrow B_1 \\ L &\leftarrow B_2 \\ &\ldots \\ L &\leftarrow B_k \end{aligned}$$

are the rules in Π with the head L. If there are no such rules ($k = 0$), (6.4) stands for $L \equiv \mathit{False}$.

According to the following proposition, the supported models of a completable objective program are precisely the models of its literal completion.

Proposition 6.1 *Let Π be a completable objective program. An interpretation I is a supported model of Π if and only if I is a model of $lcomp(\Pi)$.*

Proof. For the left-to-right direction, let I be a supported model of Π. Suppose that $L \equiv B_1 \vee \ldots \vee B_n$ is an element of the literal completion of Π. We will show that $I \models L \equiv B_1 \vee \ldots \vee B_n$. Since $\Pi^I \subseteq I$, we know that for every rule $L \leftarrow B_i$ ($1 \leq i \leq n$) in Π, if $I \models B_i$ then $I \models L$. Therefore, $I \models B_1 \vee \ldots \vee B_n \supset L$. Since $I \subseteq \Pi^I$, if $L \in I$ then for some rule $L \leftarrow B_i$ ($1 \leq i \leq n$) in Π, $I \models B_i$. Therefore, $I \models L \supset B_1 \vee \ldots \vee B_n$.

For the right-to-left direction, let I be a model of the literal completion of Π. Suppose that L is an arbitrary literal in Π^I. Then for some $L \leftarrow B \in \Pi$, $I \models B$. Consider the equivalence $L \equiv B_1 \vee \ldots \vee B_n$ in the literal completion of Π.

We know that $I \models L \equiv B_1 \vee \ldots \vee B_n$, and that for some i $(1 \leq i \leq n)$, $B = B_i$. Therefore, $I \models$L. So $\Pi^I \subseteq I$. Now let L be an arbitrary literal in I. Again, consider the equivalence $L \equiv B_1 \vee \ldots \vee B_n$ in the literal completion of Π. We know that $I \models L \equiv B_1 \vee \ldots \vee B_n$. Therefore, for some i $(1 \leq i \leq n)$, $I \models B_i$. We also know that $L \leftarrow B_i$ is in Π. Therefore, $L \in \Pi^I$. So $I \subseteq \Pi^I$. $\Box$

Example 6.2 Consider the following program $\Pi_{6.2}$.

$$b \leftarrow a, b$$

$$\neg b \leftarrow \neg a, \neg b$$

$$\neg a \leftarrow b$$

$$\neg a \leftarrow \neg a$$

The literal completion of $\Pi_{6.2}$ is

$$b \equiv a \wedge b$$

$$\neg b \equiv \neg a \wedge \neg b$$

$$a \equiv \mathit{False}$$

$$\neg a \equiv \neg a \vee b.$$

The unique of model of this propositional theory is $\{\neg a, \neg b\}$, which is also the unique supported model of $\Pi_{6.2}$. $\Diamond$

According to Proposition 6.1, questions about the nonmonotonic consequence relation for completable objective programs can be transformed into questions about the monotonic consequence relation for propositional logic. In Chapter 9, we will exploit this fact in an approach to automated query answering and planning.

Proposition 6.2 *Let* Π *be a completable normal program and* Π' *be the objective program obtained by adding to* Π*, for every atom* A*, the rule*

$$\neg A \leftarrow \neg A.$$

An interpretation I is a supported model of Π' if and only if I is a model of comp(Π).

Proof. Notice that the literal completion of Π' is simply the completion of Π, plus, for each atom A, the tautology

$$\neg A \equiv \neg A.$$

This means that the completion of Π and the literal completion of Π' have the same models. Thus, the proposition follows by Proposition 6.1. $\square$

Example 6.3 As an illustration of Proposition 6.2, consider the normal program $\Pi_{6.3}$,

$$a \leftarrow b$$

and the corresponding objective program $\Pi'_{6.3}$,

$$\begin{aligned} a &\leftarrow b \\ \neg a &\leftarrow \neg a \\ \neg b &\leftarrow \neg b. \end{aligned}$$

The unique supported model of $\Pi'_{6.3}$ is $\{\neg a, \neg b\}$, which is also the unique model of the completion of $\Pi_{6.3}$,

$$\begin{aligned} a &\equiv b \\ b &\equiv \mathit{False}. \end{aligned}$$

$\diamond$

Proposition 6.2 suggests that the completion semantics for normal programs can be viewed as consisting of two parts. First, it requires supported models. Second, it assumes automatic support for all negative literals in an interpretation. Since negation is not allowed in the heads of normal program rules, a normal program is incapable of supporting negative literals explicitly. Therefore, the assumption of negative support in the case of normal programs is inevitable. For programs that allow negation in the heads of rules, on the other hand, the assumption of negative support is both unnecessary and inappropriate. Accordingly, a natural extension of the completion semantics for this larger class of programs is one that retains the requirement of supportedness but drops the assumption of negative support. This is precisely the semantics of objective programs.

6.3 Other Connections to Logic Programming

In this section, we investigate the relationship between the semantics of objective programs and other proposed semantics for logic programs. The reader may skip this section without loss of continuity.

6.3.1 Classical Semantics

Historically, the first logic programs were positive programs. They did not allow either classical or default negation but consisted simply of Horn clauses from resolution theorem proving. (Logic programming began with the realization that certain linear resolution procedures behaved with respect to such clauses essentially as a program interpreter [Kowalski, 1974].) As such, the first semantics for logic programs was the classical semantics in which $\leftarrow$ was understood as the material conditional. The models of the program were the models of the clausal theory. This is a special case of the definition of a model for an objective program given in Section 5.3.

The following proposition shows a connection between the classical semantics and the semantics of objective programs. This connection holds for arbitrary objective programs, not only for positive programs.

Proposition 6.3 *Let Π be an objective program and Π' be the program obtained by adding to Π, for every atom A, the rules*

$$A \leftarrow A$$

$$\neg A \leftarrow \neg A.$$

An interpretation I is a supported model of Π' if and only if I is a model of Π.

Proof. By Proposition 6.1, it suffices to show that I is a model of the literal completion of Π' if and only if I is a model of Π. Let

$$L \equiv L \vee B_1 \vee \ldots \vee B_n \tag{6.5}$$

be an arbitrary formula in the literal completion of Π'. In light of the rules in $\Pi' \setminus \Pi$, we know that the literal L which appears on the lefthand side of (6.5) also, as shown, appears as a disjunct on the right. (In this case, we have written L as the first disjunct on the right.) Now, (6.5) is equivalent to the conjunction of

$$L \supset L \vee B_1 \vee \ldots \vee B_n$$

$$L \vee B_1 \vee \ldots \vee B_n \supset L.$$

The first conditional is a tautology, and the second is equivalent to the conjunction of the following material conditionals,

$$B_1 \supset L$$

$$B_2 \supset L$$

$$\ldots$$

$$B_n \supset L$$

which correspond precisely to the rules in Π with the head L. Since this holds for every equivalence in Π', the models of the literal completion of Π' are precisely the classical models of Π. □

Example 6.4 As an illustration of Proposition (6.3), consider the program $\Pi_{6.4}$,

$$a \leftarrow b$$

and the corresponding objective program $\Pi'_{6.4}$,

$$\begin{array}{l} a \leftarrow b \\ a \leftarrow a \\ \neg a \leftarrow \neg a \\ b \leftarrow b \\ \neg b \leftarrow \neg b. \end{array}$$

$\Pi'_{6.4}$ has three supported models,

$$\{a, b\}, \ \{a, \neg b\}, \ \{\neg a, \neg b\},$$

which are also the models of $\Pi_{6.4}$ according to the classical semantics of positive programs. $\diamond$

6.3.2 Stable Model Semantics

Let Π be a positive program. The models of Π are given by the classical semantics defined in Section 6.3.1. We say that a model I of Π is *minimal* if there is no model I' of Π such that the set of atoms in I' is a proper subset of the set of atoms in I.[3] It is well-known [van Emden and Kowalski, 1976] that every positive program Π has a unique minimal model. Let us designate the minimal model of Π by $\alpha(\Pi)$.

Now let Π instead be a normal program, and I be an interpretation. By Π_I we designate the program that is obtained from Π by deleting

[3]In the original definition of the stable model semantics for normal programs [Gelfond and Lifschitz, 1988], an interpretation I was represented, not by a set of literals, but by a set of atoms. Every atom in such a set M was taken to be true in I, and every atom not M was taken to be false in I. Given this representation, a minimal model can be defined as a model which has no subset that is also a model. Since we represent an interpretation by a set of literals, we require a different definition that distinguishes between an interpretation and the set of atoms in it.

(i) each rule that contains a negative literal $\neg A$ in its body with $A \in I$, and

(ii) all negative literals in the bodies of the remaining rules.

Notice that Π_I is a positive program. An interpretation I is said to be a *stable model* of Π if $I = \alpha(\Pi_I)$ [Gelfond and Lifschitz, 1988].

By the *positive atom dependency graph* for a normal program we mean the directed graph which has atoms as nodes, and which has an edge from each atom that appears in the head of a rule to each atom that appears positively in the body of the rule. A normal program Π is called *positive-order-consistent* [Fages, 1994] if there are no infinite paths in the positive atom dependency graph for Π.

Fages has shown that the completion semantics and stable model semantics coincide for positive-order-consistent normal programs. We, therefore, have the following proposition.

Proposition 6.4 *Let* Π *be a completable, positive-order-consistent, normal program. Let* Π' *be the objective program obtained by adding to* Π*, for every atom* A*, the rule*

$$\neg A \leftarrow \neg A.$$

An interpretation I *is a supported model of* Π' *if and only if* I *is a stable model of* Π.

Example 6.5 As an illustration of Proposition 6.4, consider the normal program $\Pi_{6.5}$,

$$a \leftarrow \neg b$$

$$b \leftarrow \neg a$$

$$c \leftarrow a$$

$$c \leftarrow b$$

and the corresponding objective program $\Pi'_{6.5}$,

$$\begin{aligned} a &\leftarrow \neg b \\ b &\leftarrow \neg a \\ c &\leftarrow a \\ c &\leftarrow b \\ \neg a &\leftarrow \neg a \\ \neg b &\leftarrow \neg b \\ \neg c &\leftarrow \neg c. \end{aligned}$$

The stable models of $\Pi_{6.5}$ are $\{a, \neg b, c\}$ and $\{\neg a, b, c\}$. These are also the supported models of $\Pi'_{6.5}$. $\diamond$

Example 6.6 As motivation for the restriction to positive-order-consistent normal programs consider the program $\Pi_{6.6}$,

$$a \leftarrow a$$

and the corresponding objective program $\Pi'_{6.6}$,

$$\begin{aligned} a &\leftarrow a \\ \neg a &\leftarrow \neg a. \end{aligned}$$

The unique stable model of $\Pi_{6.6}$ is $\{a\}$, but $\Pi'_{6.6}$ has two supported models, $\{a\}$ and $\{\neg a\}$. Note that $\Pi_{6.6}$ is not positive-order-consistent. $\diamond$

Example 6.7 Similarly, the program $\Pi_{6.7}$,

$$a0 \leftarrow a1,\ a1 \leftarrow a2, \ldots$$

has a unique stable model in which for all i ($i > 0$), a_i is false. However, the corresponding objective program, which includes in addition the rules

$$\neg a_0 \leftarrow \neg a_0, \; \neg a1 \leftarrow \neg a1, \ldots$$

has two supported models, one in which for all i ($i > 0$), a_i is false, and the other in which for all i ($i > 0$), a_i is true. $\diamond$

6.3.3 An Embedding in Extended Programs

The "answer set" semantics [Gelfond and Lifschitz, 1990] for extended programs can be defined by an embedding into default logic as follows. Given an extended program Π, we define $D(\Pi)$ to be the set of default rules

$$\frac{L_1 \wedge \ldots \wedge L_k : \overline{L_{k+1}}, \ldots, \overline{L_n}}{L_0}$$

(where $\overline{L}$ is the complement of L) such that

$$L_0 \leftarrow L_1, \ldots, L_k, not\ L_{k+1}, \ldots, not\ L_n$$

is a rule in Π.

The following proposition has been shown by Gelfond and Lifschitz.

Proposition 6.5 [Gelfond and Lifschitz, 1990]. *For any extended program* Π, *if* S *is an answer set for* Π *then* $Cn(S)$ *is an extension for* $D(\Pi)$, *and for every extension* E *for* $D(\Pi)$ *there is exactly one answer set* S *for* Π *such that* $Cn(S) = E$.

Corollary 6.1 *Let* Π *be an extended program, and* S *be a set of literals.* S *is an answer set for* Π *if and only if* $Cn(S)$ *is an extension for* $D(\Pi)$.

The preceding proposition shows that $D(\Pi)$ is a faithful embedding of extended programs into default logic.

Since objective programs are a special case of causal theories, we know by Proposition 5.4 that objective programs can also be embedded in default logic. Let Π be an objective program. We define an embedding $T'(\Pi)$ of Π into default logic as

$$T'(\Pi) = \left\{ \frac{: L_1, \ldots, L_n}{L} : L \leftarrow L_1, \ldots, L_n \in \Pi \right\}.$$

We have the following proposition.

Proposition 6.6 *Let* Π *be an objective program. An interpretation* I *is a supported model of* Π *if and only if* $Cn(I)$ *is an extension for* $T'(\Pi)$*.*

Proof. Recall the embedding $T(D)$ of a causal theory D in default logic of Section 5.6. Since an objective program is a notational variant of a causal theory, we can take T also to be defined for objective programs as

$$T(\Pi) = \left\{ \frac{: L_1 \wedge \ldots \wedge L_n}{L} : L \leftarrow L_1, \ldots, L_n \in \Pi \right\}.$$

Let I be an interpretation. By Proposition 5.4, we know that I is a supported model of Π if and only if $Cn(I)$ is an extension for $T(\Pi)$. However, noticing that $Cn(I)$ is the set of formulas true in I, it is easy to see that for any finite set B of literals, $\neg \bigwedge B \notin Cn(I)$ if and only if for every literal $L \in B$, $\neg L \notin Cn(I)$. It follows that $T(\Pi)^{Cn(I)} = T'(\Pi)^{Cn(I)}$, and thus that $Cn(I)$ is an extension for $T(\Pi)$ if and only if $Cn(I)$ is an extension for $T'(\Pi)$. We conclude that I is a supported model of Π if and only if $Cn(I)$ is an extension for $T'(\Pi)$. $\Box$

Using the embeddings of extended and objective programs into default logic, we can show that there is also the following embedding of objective programs into extended programs. Given an objective program Π, we define a corresponding extended program $R(\Pi)$ as follows:

$$R(\Pi) = \{L \leftarrow \mathit{not}\, \overline{L_1}, \ldots, \mathit{not}\, \overline{L_n} : L \leftarrow L_1, \ldots, L_n \in \Pi\}.$$

Proposition 6.7 *Let Π be an objective program. For any interpretation I, I is a supported model of Π if and only if I is an answer set for $R(\Pi)$.*

Proof. Let Π be an objective program. Notice that $T'(\Pi) = D(R(\Pi))$. It follows, by Proposition 6.6, that I is a supported model of Π if and only if $Cn(I)$ is an extension for $D(R(\Pi))$. By Corollary 6.1, $Cn(I)$ is an extension for $D(R(\Pi))$ if and only if I is an answer set for $R(\Pi)$. Therefore, I is a supported model of Π if and only if I is an answer set for $R(\Pi)$. □

6.4 Some Standard Ways of Lending Support

The supported model semantics imposes a rather strong completeness requirement on objective programs. If an objective program is to have any supported models at all, then it must include, for every atom A, rules that support either A or $\neg A$. (A similar fact holds for causal theories generally.) The burden of writing programs that satisfy this completeness requirement can be lessened by adopting certain standard ways of augmenting a program. We have employed three such ways in the examples of this chapter.

- Inertial support

$$\begin{aligned} b &\leftarrow a, b \\ \neg b &\leftarrow \neg a, \neg b \end{aligned}$$

- Negative support

$$\neg a \leftarrow \neg a$$

- Classical support

$$\begin{aligned} a &\leftarrow a \\ \neg a &\leftarrow \neg a \end{aligned}$$

Inertial support was used in program $\Pi_{6.1}$. Negative support was used in program $\Pi'_{6.3}$. Classical support was used in program $\Pi'_{6.4}$. The name "inertial support" is motivated by an application in the next chapter.

Chapter 7

Formalizing Action Domains as Causal Theories

In this chapter we describe a general approach to formalizing action domains as causal theories. In doing so, we define a uniform action description language $\mathcal{L}_{\mathrm{CL}}$. Essentially, $\mathcal{L}_{\mathrm{CL}}$ is the language of causal theories, restricted to a particular kind of signature. In calling the language "uniform" we draw attention to the fact that—unlike, for example, the language $\mathcal{A}_{\mathrm{CL}}$—the language $\mathcal{L}_{\mathrm{CL}}$ contains only one kind of proposition. Both static and dynamic causal laws can be expressed by propositions of this one kind.

7.1 The Language $\mathcal{L}_{\mathrm{CL}}$

The signature for a specific $\mathcal{L}_{\mathrm{CL}}$ language is specified by a triple $\langle \mathbf{A}, \mathbf{F}, \mathbf{T} \rangle$, where $\mathbf{A}$ is a set of action names, $\mathbf{F}$ is a nonempty set of fluent names, and $\mathbf{T}$ is a nonempty set of time names, corresponding to a subset of the integers. We view time as continuous but refer to only a discrete subset of times by name. The atoms of the language are expressions of the forms a_t and f_t, where a, f, and t are action, fluent, and time names, respectively. Intuitively, a_t is true if and only if the action named by a begins to occur immediately after the time named by t (normally, we will express this by saying that a occurs at t), and f_t is true if and only if the fluent named by f holds at the time named by t.[1] A formula of the language is a propositional combination of expressions of these two forms.

An $\mathcal{L}_{\mathrm{CL}}$ *domain description* is a causal theory—i.e., a set of causal laws—in an $\mathcal{L}_{\mathrm{CL}}$ language.

Note that there are no restrictions on the times that may be referenced in the antecedent or consequent of a causal law. In particular, the times referenced in the antecedent are not required to precede (or not follow) those referenced in the consequent. Thus, static causal laws—and even laws that intuitively describe causation that runs backwards in time—are allowed.

7.2 The Suitcase Domain

As an initial illustration, we will formalize a domain from [Lin, 1995] in which there is a suitcase with two latches, each of which may be in either of two positions, up or down. The suitcase is spring-loaded so that whenever both latches are in the up position the suitcase is caused to be open. We will model the opening of the suitcase (as Lin does as well) as a static effect; that is, we will not model a state of the domain in which both latches are up but the suitcase is not (yet) open.

To formalize the Suitcase domain in the language $\mathcal{L}_{\mathrm{CL}}$, we first choose a set of actions and elementary fluents, and a set of names to designate them. One possible choice is the following.

$\mathit{Toggle}(L1)$: the action of toggling Latch 1

$\mathit{Toggle}(L2)$: the action of toggling Latch 2

Close : the action of closing the Suitcase

[1]Normally, we will drop the phrase "named by" in contexts such as this. So, for example, given an action name a, a fluent name f, and a time name t, we will allow ourselves to write "the action a," "the fluent f," and "the time t." Our intention is to always make it clear by the accompanying words whether we are referring to a name or to what it names.

$Up(L1)$: the fluent that Latch 1 is up

$Up(L2)$: the fluent that Latch 2 is up

$Open$: the fluent that the Suitcase is open

The first three symbols designate possible actions in the domain. The remaining three symbols designate the states of objects in the domain, specifically, the states of the latches and the suitcase.

Next, we choose a set of time names. In the Suitcase domain, we identify time with the natural numbers. Other possible choices would be the integers, or a finite sequence of the natural numbers or integers. There may be many reasons for choosing to represent time as finite; the domain itself may exist for only a finite time, the domain description may correctly describe the domain over only a finite time, or we may simply be interested in what happens in the domain only over a finite time.

We will normally specify the signature for an $\mathcal{L}_{\text{CL}}$ language by a BNF-style grammar. For example, the signature for the Suitcase domain is specified as follows, where nonterminal symbols are written in lowercase.

$$latch ::= L1 \mid L2$$

$$action ::= Toggle(latch) \mid Close$$

$$fluent ::= Up(latch) \mid Open$$

$$time ::= 0 \mid 1 \mid \cdots$$

The clauses for *action*, *fluent*, and *time* specify the signature $\langle action, fluent, time \rangle$ for a specific $\mathcal{L}_{\text{CL}}$ language. The type *latch* is an auxiliary type which simplifies the specification of the types *action* and *fluent*.

Given our choice of language, the Suitcase domain can be formalized (in part, as we will see) by writing schemas as follows. Below and throughout this disseration, t is used as a meta-variable of type *time*. Here we also use l as a meta-variable of type *latch*.

$$Toggle(l)_t \wedge Up(l)_t \Rightarrow \neg Up(l)_{t+1} \tag{7.1}$$

$$Toggle(l)_t \wedge \neg Up(l)_t \Rightarrow Up(l)_{t+1} \tag{7.2}$$

$$Close_t \Rightarrow \neg Open_{t+1} \tag{7.3}$$

$$Up(L1)_t \wedge Up(L2)_t \Rightarrow Open_t. \tag{7.4}$$

According to schemas (7.1) and (7.2), at every time t, toggling one of the latches at t causes it to be in the opposite state at time $t+1$. According to schema (7.3), at every time t, closing the suitcase at t causes it not to be open at time $t+1$. According to schema (7.4), both latches being up at a time t causes the suitcase to be open also at t.

Schemas (7.1)–(7.3) are dynamic causal laws. Intuitively, they are similar in meaning to the propositions

$$Toggle(l) \textbf{ causes } Up(l) \textbf{ if } \neg Up(l)$$

$$Toggle(l) \textbf{ causes } \neg Up(l) \textbf{ if } Up(l)$$

$$Close \textbf{ causes } \neg Open$$

of the language $\mathcal{A}_{\mathrm{CL}}$. Schema (7.4) is a static causal law. Intuitively, it is similar in meaning to the proposition

$$Up(L1) \wedge Up(L2) \Rightarrow Open$$

of the language $\mathcal{A}_{\mathrm{CL}}$. In fact, the static and dynamic laws shown above are only two of many possible kinds. We will see other kinds of causal laws (with other patterns of temporal reference) in the next section and in later examples.

7.3 Inertial Fluents and Exogenous Facts

The causal theory (7.1)–(7.4) is incomplete, because it does not state sufficient conditions for certain kinds of facts being caused—specifically, facts preserved by inertia, facts about the initial situation, and facts about which actions occur (and when). In this section we describe some standard ways of augmenting an $\mathcal{L}_{\rm CL}$ domain description in order to fill this gap.

Explaining Action Occurrences

Normally, in formalizing an action domain we do not describe the causes of actions. This is not because we believe that the agent's actions are not caused, or that they are "self-caused," or that the agent has free will. (We may or may not believe such things; it does not matter.) Rather, the reason that we do not describe the causes of actions is that they are irrelevant to the purposes of deliberation and planning. For these purposes, we must be able to answer "what if" questions, such as the one below, without regard to what the agent is or is not destined to do.

> *What if the agent were to simultaneously perform the actions Toggle*$(L1)$ *and Toggle*$(L2)$*?*

Therefore, even if we did specify the causes of actions, we would somehow have to ignore them for the purposes of deliberation and planning. For these purposes, the agent's abilities are relevant, but his destiny is not.

Nevertheless, we know that a causal theory must specify conditions that are sufficient for every fact in a causally explained interpretation to be caused, including facts about the occurrences (and non-occurrences) of actions. We can reconcile this observation with the point made in the preceding paragraph by representing that facts about action occurrences may be *exogenous* to the causal theory. We do this by writing the following schemas,

$$a_t \Rightarrow a_t \tag{7.5}$$

$$\neg a_t \Rightarrow \neg a_t \tag{7.6}$$

where a and t are meta-variables of type *action* and *time*, respectively. According to schema (7.5), the occurrence of an action a at a time t is caused whenever a

occurs at t. According to schema (7.6), the non-occurrence of an action a at a time t is caused whenever a does not occur at t. Since this is so, no other cause for the occurrence or non-occurrence of a at t is required. Intuitively, the effect of schemas (7.5) and (7.6) is to exempt facts about action occurrences from the principle of universal causation.

Explaining Facts about the Initial Situation

When, as in the Suitcase domain, we identify time with the natural numbers (rather than, for example, the integers), we have to be concerned with how facts about the initial state of a world (i.e., at time 0) are to be explained. In the Suitcase domain, we can think of time 0 as the moment at which the suitcase came into existence, or as simply an arbitrary moment during the "life" of the suitcase. Either way, whatever in the real world causes the latches to be either up or down at time 0, and whatever it is that causes the suitcase to be either open or closed at time 0 (except in the case that both latches are initially up) lies outside of time as it is represented in our theory. Therefore, except in the one case mentioned, we cannot hope to describe the real causes of these facts. Instead, we represent that facts about time 0 may be exogenous to our theory by writing the following schemas,

$$f_0 \Rightarrow f_0 \tag{7.7}$$

$$\neg f_0 \Rightarrow \neg f_0 \tag{7.8}$$

where f is a meta-variable of type *fluent*. According to schema (7.7), a fluent f is caused to hold at time 0 whenever it does hold at time 0. According to schema (7.8), a fluent f is caused to not hold at time 0 whenever it does not hold at time 0. Since this is so, no other explanation for f's holding or not holding at time 0 is required. Intuitively, the effect of including schemas (7.7) and (7.8) is to exempt facts about the initial situation from the principle of universal causation.

Explaining Facts by Inertia

In some instances, when a fluent remains true from one time to the next, its truth at the second time can be explained by inertia. If this is true for all pairs of successive times, we say that the fluent is "inertial." In Chapters 3 and 4, we tacitly assumed that every fluent literal designated an inertial fluent. (Recall that in defining $Res(E,S)$, for an explicit effect E and state S, we denoted the literals that were preserved by inertia in a candidate next state S' by $S \cap S'$.) We will call this the *standard inertia assumption*.

The standard inertia assumption is appropriate for the Suitcase domain and many others. However, it is not always appropriate. Depending on the language, there may be fluent literals that do not designate inertial fluents, and there may be inertial fluents that are not designated by fluent literals. We will see examples of both kinds later in this chapter.

By a *fluent formula* we mean a propositional combination of fluent names. Given a fluent formula σ and a time name t, we write σ_t to stand for the formula obtained from σ by simultaneously replacing each occurrence of each fluent name f by the atom f_t. As an example, the expression $(\neg H \wedge \neg T)_0$, for fluent names H and T, stands for the formula $(\neg H_0 \wedge \neg T_0)$.

Using this convention, the schematic form of the inertia law is

$$\sigma_t \wedge \sigma_{t+1} \Rightarrow \sigma_{t+1} \tag{7.9}$$

where σ stands for a fluent formula, and t is a meta-variable of type *time*.

Normally, it will be convenient to specify the inertial fluent formulas for a domain by adding an extra clause to the specification of the signature. For example, in the case of the Suitcase domain, we would specify the standard inertia assumption by adding the clause

$$\textit{inertial-formula} ::= [\neg]\textit{fluent}.$$

• $\neg Toggle(L1)_0$	• $\neg Toggle(L1)_1$	• $\neg Toggle(L1)_2$	$\cdots$
• $Toggle(L2)_0$	• $\neg Toggle(L2)_1$	• $\neg Toggle(L2)_2$	$\cdots$
• $\neg Close_0$	• $\neg Close_1$	• $\neg Close_2$	$\cdots$
• $Up(L1)_0$	• $Up(L1)_1$	• $Up(L1)_2$	$\cdots$
• $\neg Up(L2)_0$	$Up(L2)_1$	• $Up(L2)_2$	$\cdots$
• $\neg Open_0$	$Open_1$	• $Open_2$	$\cdots$

Figure 7.1: A possible world history in the Suitcase domain

According to this specification, the following fluent formulas designate inertial fluents: $Up(L1)$, $\neg Up(L1)$, $Up(L2)$, $\neg Up(L2)$, $Open$, and $\neg Open$. Intuitively, the effect of including schema (7.9) is to exempt the persisting values of inertial fluents from the principle of universal causation.

7.4 The Suitcase Domain (continued)

The complete description of the Suitcase domain is expressed by the schemas (7.1)–(7.9). The schemas (7.1)–(7.4) are domain dependent. The schemas (7.5)–(7.9) represent standard ways of augmenting an $\mathcal{L}_{\text{CL}}$ domain description. We will call them the *standard schemas*. Notice that the set of causal laws represented by the standard schemas is determined by the signature of the domain—specifically, by the action, fluent, and time names—and by the specification of the inertial fluents. In the terminology of Section 6.4, the schemas (7.5)–(7.6) and (7.7)–(7.8) provide classical support to a causal theory, and the schema (7.9) provides inertial support. The frame problem is solved by schema (7.9).

Given a causal theory D, we identify the causally possible world histories according to D with the interpretations that are causally explained according to D. As an example, let D be the domain description for the Suitcase domain, and let I be the interpretation displayed in Figure 7.1. Notice that I specifies, for every action a and time t, whether or not a occurs at t, and, for every elementary fluent f

and time t, whether or not f holds at t. (In this instance, the ellipses are intended to mean that after time 2 no action occurs and no fluent changes its value.) It is not difficult to see that I is causally explained according to D. The bullets indicate the literals at times 0, 1, and 2 that appear in D^I due to the standard schemas (7.5)–(7.9). The two literals that are not marked by bullets appear in D^I due to the schemas (7.2) and (7.4). The remaining literals (represented by the ellipses) appear in D^I because of schemas (7.5)–(7.6) and (7.9). (The atoms $Open_t$, for all $t > 1$, also appear in D^I due to schema (7.4).) Since D^I contains no other formulas, we conclude that $I = D^I$. Thus, I is the unique model of D^I. Therefore, I is causally explained according to D.

The following formula is a consequence of D.[2]

$$Up(L1)_0 \wedge Up(L2)_0 \wedge Close_0 \supset Toggle(L1)_0 \vee Toggle(L2)_0 \tag{7.10}$$

To see this, notice that (7.10) is entailed by the formulas

$$\begin{aligned}
&\neg Open_1 \equiv Close_0 \vee (\neg Open_0 \wedge \neg Open_1)\\
&Open_1 \equiv (Up(L1)_1 \wedge Up(L2)_1) \vee (Open_0 \wedge Open_1)\\
&\neg Up(L1)_1 \equiv (Toggle(L1)_0 \wedge Up(L1)_0) \vee (\neg Up(L1)_0 \wedge \neg Up(L1)_1)\\
&\neg Up(L2)_1 \equiv (Toggle(L2)_0 \wedge Up(L2)_0) \vee (\neg Up(L2)_0 \wedge \neg Up(L2)_1)
\end{aligned}$$

which belong to the literal completion of D.

In general, whenever both latches are up, it is impossible to perform *only* the action of closing the suitcase; one must also toggle one of the latches. This may seem unintuitive. However, recall that we have chosen to model the suitcase being open as a static effect of the latches being up, so there is no time in any causally possible world history at which both latches are up and the suitcase is closed.

The reader may check that the interpretation displayed in Figure 7.2 is also causally explained according to D. Notice that in this interpretation at time 0

[2]We assume the following order of precedence for the connectives: $\neg$ binds most tightly, next $\wedge$ and $\vee$, and finally $\supset$ and $\equiv$.

• $\mathit{Toggle}(L1)_0$	• $\neg \mathit{Toggle}(L1)_1$	• $\neg \mathit{Toggle}(L1)_2$	$\cdots$
• $\neg \mathit{Toggle}(L2)_0$	• $\neg \mathit{Toggle}(L2)_1$	• $\neg \mathit{Toggle}(L2)_2$	$\cdots$
• Close_0	• $\neg \mathit{Close}_1$	• $\neg \mathit{Close}_2$	$\cdots$
• $\mathit{Up}(L1)_0$	$\neg \mathit{Up}(L1)_1$	• $\neg \mathit{Up}(L1)_2$	$\cdots$
• $\mathit{Up}(L2)_0$	• $\mathit{Up}(L2)_1$	• $\mathit{Up}(L2)_2$	$\cdots$
• Open_0	$\neg \mathit{Open}_1$	• $\neg \mathit{Open}_2$	$\cdots$

Figure 7.2: Another possible world history in the Suitcase domain

two actions—*Toggle(L1)* and *Close*—are performed concurrently, and the suitcase is successfully closed.

7.5 The Expressive Capacity of $\mathcal{L}_{\mathrm{CL}}$

In the course of formalizing the Suitcase domain, we have seen how static causal laws, the explicit effects of actions, and fluent preconditions are represented in the language $\mathcal{L}_{\mathrm{CL}}$. Ramification and qualification constraints, as discussed in Section 3.3, are expressed by schemas of the forms

$$\mathit{True} \Rightarrow \sigma_t$$

$$\neg\sigma_t \Rightarrow \mathit{False}$$

respectively, where σ (the constraint) is a fluent formula.

In this section we further illustrate the expressive potential of the language $\mathcal{L}_{\mathrm{CL}}$ by means of a number of small examples.

7.5.1 Concurrent Actions

We have already observed in relation to the Suitcase domain the possibility of performing actions concurrently. The need for concurrency is illustrated by the following example.

Example 7.1 In this domain, from [Gelfond *et al.*, 1991b], there is a bowl of soup. We suppose that the agent, using his two hands, can raise or lower each side of the bowl independently of the other. However, unless he performs the corresponding actions concurrently, he will spill the soup.[3]

The signature is specified by the following grammar.

$$\begin{aligned}
&\mathit{side} ::= \mathit{Left} \mid \mathit{Right}\\
&\mathit{action} ::= \mathit{Raise}(\mathit{side}) \mid \mathit{Lower}(\mathit{side})\\
&\mathit{fluent} ::= \mathit{Up}(\mathit{side}) \mid \mathit{Down}(\mathit{side}) \mid \mathit{Spilled}\\
&\mathit{time} ::= 0 \mid \cdots \mid 5
\end{aligned}$$

We adopt the standard inertia assumption.

$$\mathit{inertial\text{-}formula} ::= [\neg]\mathit{fluent}$$

The causal theory $D_{7.1}$ for the Soup domain is represented by the standard schemas (7.5)–(7.9), plus the schemas (7.11)–(7.14) below. (Here s is a meta-variable of type *side*.)

$$\mathit{True} \Rightarrow \mathit{Down}(s)_t \equiv \neg \mathit{Up}(s)_t \tag{7.11}$$

$$\mathit{Raise}(s)_t \Rightarrow \mathit{Up}(s)_{t+1} \tag{7.12}$$

$$\mathit{Lower}(s)_t \Rightarrow \mathit{Down}(s)_{t+1} \tag{7.13}$$

$$\mathit{Up}(\mathit{Left})_t \not\equiv \mathit{Up}(\mathit{Right})_t \Rightarrow \mathit{Spilled}_t \tag{7.14}$$

Schema (7.11) represents, for each side s and time t, an explicit definition of $\mathit{Down}(s)_t$; in effect, it defines *Down* in terms of *Up*. Schemas (7.12) and (7.13) describe the explicit effects of raising and lowering each side of the bowl.[4] Schema (7.14) says that bowl's being tilted causes the soup to be spilled.

[3] As is noted in [Gelfond *et al.*, 1991b], this is essentially the example from [Pednault, 1987] in which two agents lift opposite sides of a table.

[4] When t is 5, $t+1$ is not an expression of type *time*. Consequently, there are no corresponding instances of (7.12) and (7.13) in $D_{7.1}$. In general, schemas represent all and only those of their instances whose atoms belong to the signature with respect to which they are defined.

The formulas

$$\neg Up(Left)_0 \wedge \neg Up(Right)_0 \wedge Raise(Left)_0 \wedge \neg Raise(Right)_0 \supset Spilled_1$$

$$\neg Spilled_0 \wedge Raise(Left)_0 \wedge Raise(Right)_0 \supset \neg Spilled_1$$

are consequences of $D_{7.1}$.[5] $\diamond$

In [Gelfond *et al.*, 1991b], [Baral and Gelfond, 1993], and [Thielscher, 1995b], the Soup domain is described essentially as follows: (i) the action of raising either side of the bowl causes the soup to be spilled, but (ii) assuming the soup is not already spilled, raising both sides of the bowl concurrently causes it not to be spilled. The formal renderings of statements (i) and (ii) are made consistent by a mechanism that cancels the normal "inheritance" of effects. In this case, since the effects of individually raising each side of the bowl are inconsistent with the effect of concurrently raising both sides, the effects of the individual subactions are not inherited.

In a framework in which it is impossible to write causal laws that describe the indirect effects of actions, some kind of inheritance and cancellation mechanism is necessary in order to efficiently specify the effects of concurrent actions. However, in a framework in which such laws can be written, this is not the case. The reason is that we can adopt the following formalization strategy. First, we describe the explicit effects of individual actions at a sufficiently basic level so that they can be said to hold without exception. Secondly, we describe the potential ramifications of these effects on other fluents by means of additional causal laws.

[5]These claims are machine checked in Section 9.4. For this purpose, we first eliminate $Down(s)_t$, for all s of type *side* and t of type *time*, from the signature and the corresponding definitions from the domain description. (See Section 5.5.) Notice that when $\neg Up(s)_t$ is substituted for $Down(s)_t$ in the causal laws represented by the standard schemas, each of the new causal laws is redundant. It is also necessary to replace (7.14) by the two causal laws

$$Up(Left)_t \wedge \neg Up(Right)_t \Rightarrow Spilled_t$$
$$\neg Up(Left)_t \wedge Up(Right)_t \Rightarrow Spilled_t$$

in order to bring the domain description into the class of objective programs. These transformations have no effect on the consequences of $D_{7.1}$ in the reduced language.

The strategy just described was used in our formalization of the Soup domain, where we explicitly described the effects of raising and lowering each side of the bowl, not on the fluent *Spilled*, but on the fluents $Up(s)$ and $Down(s)$, where s is *Left* or *Right*. The causal connections between facts about the values of these fluents and facts about *Spilled* are reflected in the additional causal laws represented by schema (7.14).[6]

It is possible in $\mathcal{L}_{\mathrm{CL}}$ to explicitly describe the effects of performing actions concurrently. For example, if soup-raising were a contest, we might write

$$Raise(Right)_t \wedge Raise(Left)_t \Rightarrow Win_{t+1} .$$

However, in $\mathcal{L}_{\mathrm{CL}}$ if the explicitly described effect of performing a set of actions concurrently is incompatible with the effect of its individual subactions, the result is that the actions simply cannot be performed concurrently. No cancellation mechanism is built into the semantics.

Notice that in our formalization of the Soup domain it is impossible, except at the last time 5, to concurrently raise and lower the same side of the bowl. This is because, according to (7.12) and (7.13), the actions have inconsistent effects.[7] However, even when two actions have consistent effects and each is individually performable—such as $Raise(Left)$ and $Raise(Right)$—it may be impossible for an agent to perform both actions concurrently. (Imagine, for example, that the agent is a robot with only one arm.) When this is so, the domain description should explicitly rule out this possibility.

We can rule out the possibility of the agent concurrently performing any pair of distinct actions (even at the last time, if there is one) by writing the schema

$$a_t \wedge a'_t \Rightarrow False \quad \text{where } a \neq a' \tag{7.15}$$

[6]A similar approach to formalizing the Soup domain is taken in [Turner, 1996].

[7]It is necessary to exempt time 5 from the preceding remark, because when t is 5, (7.12) and (7.13) are not in the language of the domain.

where t is a meta-variable of type *time*, and a and a' are meta-variables of type *action*. Adding Schema 7.15 to $D_{7.1}$ would lead to the consequence that the agent could neither raise nor lower an unspilled bowl of soup without spilling it.

7.5.2 Nondeterministic Actions

The semantics of causal theories rests on the principle of universal causation, according to which every fact is caused. Intuitively, in the case of a nondeterministic action, there is no cause for one of its possible effects rather than another. We have already seen, however—in schemas (7.5)–(7.9)—that there are ways of effectively exempting facts from the principle of universal causation. We can use laws of a similar form to describe nondeterministic actions. This is illustrated by the following coin tossing example.

Example 7.2 When a coin is tossed, whether it will land heads or tails is determined (if at all) by an untold number of conditions—such as the exact force, spin, and direction of the toss, and the velocities of specific molecules in the surrounding air—of which common sense knows little or nothing. In view of this complexity, whether or not coin tossing is truly nondeterministic, we may choose to model it as if it were.[8] This can be done as follows.

The signature is specified by the following grammar.

$$\begin{aligned} \textit{action} &::= \textit{Toss} \\ \textit{fluent} &::= \textit{Heads} \\ \textit{time} &::= 0 \mid 1 \mid \cdots \end{aligned}$$

We adopt the standard inertia assumption.

$$\textit{inertial-formula} ::= [\neg]\textit{fluent}$$

[8]See [Lewis, 1986b] (Postcript B) for an argument against the presumption that coin tossing is known to be deterministic.

The causal theory $D_{7.2}$ for the Coin Tossing domain is represented by the standard schemas (7.5)–(7.9), plus the schemas (7.16)–(7.17) below.

$$\mathit{Toss}_t \wedge \mathit{Heads}_{t+1} \Rightarrow \mathit{Heads}_{t+1} \tag{7.16}$$

$$\mathit{Toss}_t \wedge \neg\mathit{Heads}_{t+1} \Rightarrow \neg\mathit{Heads}_{t+1} \tag{7.17}$$

Intuitively, schema (7.16) says that tossing the coin possibly causes it to land heads, and schema (7.17) says that tossing the coin possibly causes it to land tails.[9] Intuitively, according to schemas (7.16) and (7.17), for every time t, Toss_t renders Heads_{t+1} exogenous.

According to $D_{7.2}$, the set of causally explained interpretations in which the coin is tossed at every time has the cardinality of the continuum. $\diamond$

In general, in order to express that at every time t, the action a possibly causes the formula σ to hold at $t+1$, we write the schema

$$a_t \wedge \sigma_{t+1} \Rightarrow \sigma_{t+1}.$$

In the previous example, the two possible effects of *Toss*—namely, *Heads* and $\neg$*Heads*—are inconsistent, so on each occasion *Toss* can bring about at most one of them. The following example, which is credited to Ray Reiter in [Kartha and Lifschitz, 1994] and [Shanahan, 1997], is different in this respect.

Example 7.3 Consider the action of dropping a block onto the surface of a table that is painted black and white. The block may land entirely within a black area, entirely within a white area, or on both a black and white area.

The signature is specified by the following grammar.

$$\begin{aligned} \mathit{action} &::= \mathit{Drop} \\ \mathit{fluent} &::= \mathit{Black} \mid \mathit{White} \\ \mathit{time} &::= 0 \mid 1 \mid \cdots \end{aligned}$$

[9] Schemas (7.16) and (7.17), can be viewed as making explicit the conversational implicatures of the natural language statement: "Toss causes heads or not heads." (See the long footnote associated with Example 4.4.)

We adopt the standard inertia assumption.

$$\textit{inertial-formula} ::= [\neg]\textit{fluent}$$

The domain description $D_{7.3}$ for this domain is represented by the standard schemas (7.5)–(7.9), plus the schemas (7.18)–(7.20) below.

$$Drop_t \Rightarrow Black_{t+1} \vee White_{t+1} \tag{7.18}$$

$$Drop_t \wedge Black_{t+1} \Rightarrow Black_{t+1} \tag{7.19}$$

$$Drop_t \wedge White_{t+1} \Rightarrow White_{t+1} \tag{7.20}$$

According to the schema (7.18), the action of dropping the block causes it to land so that it is either on a black or a white area. According to schemas (7.19) and (7.20), dropping the block possibly causes it to land on a black area and possibly causes it to land on a white area.[10]

According to $D_{7.3}$, the set of causally explained interpretations in which the block is dropped at every time has the cardinality of the continuum. Whenever *Drop* occurs, it has three possible effects: *Black*, *White*, and $Black \wedge White$. ◇

7.5.3 Actions with Delayed Effects

In each of the examples that we have considered so far, we have specified that the effects of an action are realized at the next time after the the action occurs. This is not required. We may also describe actions with delayed effects. Moreover, we may do so without describing the mechanism by which the delay is caused. This is illustrated by the following example.

Example 7.4 Consider a time bomb that, after being armed, ticks down from 3 to 0 and explodes at 0. In the following domain description, we do not represent the process of ticking down.

[10]Schemas (7.19) and (7.20) can be viewed as making explicit the conversational implicatures of the natural language statement: "The action of dropping the block causes it to land either on a black or a white area (or both)."

The signature is specified by the following grammar.

$$\begin{aligned} \mathit{action} &::= \mathit{Arm} \\ \mathit{fluent} &::= \mathit{Exploded} \\ \mathit{time} &::= 0 \mid 1 \mid \cdots \end{aligned}$$

We adopt the standard inertia assumption.

$$\mathit{inertial\text{-}formula} ::= [\neg]\mathit{fluent}$$

The causal theory $D_{7.4}$ is represented by the standard schemas (7.5)–(7.9), plus the schemas (7.21)–(7.22) below.

$$\mathit{Arm}_t \Rightarrow \mathit{Exploded}_{t+3} \tag{7.21}$$

$$\mathit{Arm}_t \wedge \mathit{Exploded}_t \Rightarrow \mathit{False} \tag{7.22}$$

According to (7.21), arming the bomb has the delayed effect after 3 time units of causing the bomb to have exploded. Schema (7.22) expresses an action precondition for arming the bomb.

The formula

$$\mathit{Arm}_0 \supset \mathit{Exploded}_3$$

is an obvious consequence of $D_{7.4}$. $\diamond$

A causal law such as (7.22), describing a delayed effect, is appropriate only if it is impossible for events that could cancel the effect to intervene during the delay. In the previous example, if we wished to allow the possibility that the bomb might be disarmed after it is armed but before it has exploded, it would be necessary to model the mechanism by which the delayed effect otherwise comes to pass. This can be done by using the general approach illustrated in the next section.[11]

[11]Mendez, Lobo, Llopis, and Baral [1996] have defined an extension of the language $\mathcal{A}$ of Gelfond and Lifschitz [1992] which allows one to refer to facts about past states of the world in the preconditions of effect propositions. Since there are no restrictions on the time references that may appear in the antecedents and consequents of causal laws, references of a similar kind are possible in the language of causal theories.

7.5.4 Things that Change by Themselves

In each of the examples that we have considered so far, changes have occurred only when actions are performed. In the following example, by contrast, changes may occur even at times when no action is performed.

Example 7.5 Imagine a row of five dominoes, numbered 1–5 and arranged in order from left to right. A sufficient condition for a domino being (caused to be) down is that the domino immediately to its left has just fallen down. For simplicity, we assume that the dominoes can fall only from left to right. An agent can tip any domino.

The signature is specified by the following grammar.

$$\begin{aligned}
&\mathit{domino} ::= 1 \mid 2 \mid 3 \mid 4 \mid 5\\
&\mathit{action} ::= \mathit{Tip}(\mathit{domino})\\
&\mathit{fluent} ::= \mathit{Down}(\mathit{domino}) \mid \mathit{Up}(\mathit{domino})\\
&\mathit{time} ::= 0 \mid \cdots \mid 5
\end{aligned}$$

We adopt the standard inertia assumption.

$$\mathit{inertial\text{-}formula} ::= [\neg]\mathit{fluent}$$

We can express that a domino d has fallen in the time interval between t and $t+1$ by writing the conjunction

$$\mathit{Up}(d)_t \wedge \mathit{Down}(d)_{t+1}.$$

The causal theory $D_{7.5}$ for the Domino domain is represented by the standard schemas (7.5)–(7.9), plus the schemas (7.23)–(7.25) below. Here d and d' are metavariables of type *domino*. (The expression $d+1$ stands for the name of the successor of the number named by d.)

$$True \Rightarrow Down(d)_t \equiv \neg Up(d)_t \tag{7.23}$$

$$Tip(d)_t \Rightarrow Down(d)_{t+1} \tag{7.24}$$

$$Up(d)_t \wedge Down(d)_{t+1} \Rightarrow Down(d')_{t+2} \quad \text{where } d' = d{+}1 \tag{7.25}$$

Schema (7.23) represents, for every domino d and time t, an explicit definition of $Down(d)_t$; in effect, it defines *Down* in terms of *Up*. Schema (7.24) describes the explicit effect of tipping a domino. Intuitively, according to schema (7.25), a causally sufficient condition for a domino being down is that the domino to its left has just fallen down.[12]

The formulas

$$[\bigwedge_{n=1..5} Up(n)_0] \wedge Tip(1)_0 \supset [\bigwedge_{n=1..5} Down(n)_5]$$
$$Up(1)_0 \wedge [\bigwedge_{t=1..5} \neg Tip(1)_t] \supset Up(1)_5$$

are consequences of $D_{7.5}$.[13] $\Diamond$

Notice that according to schema (7.25) it is not the state of a domino—the fact that it is down—that causes its successor domino to subsequently be down. Rather, it is the domino's *change* of state—the fact that it fell down—that is the cause. Consider the following alternative to (7.25).

$$Down(d)_t \Rightarrow Down(d')_{t+1} \quad \text{where } d' = d{+}1 \tag{7.26}$$

According to (7.26), the fact that a domino is down causes its successor to subsequently be down. Intuitively, this is not what we wish to say; the domino is down

[12]Schema (7.25) represents a set of dynamic causal laws that do not mention the occurrences of actions. Laws of this general kind have appeared previously in [Geffner, 1990] and [Thielscher, 1995b].

[13]These claims are machine checked in Section 9.4. For this purpose, we first eliminate $Down(d)_t$, for all d of type *domino* and t of type *time*, from the signature and the corresponding definitions from the domain description. (See Section 5.5.) Notice that when $\neg Up(d)_t$ is substituted for $Down(d)_t$ in the causal laws represented by the standard schemas, each of the new causal laws is redundant. These transformations have no effect on the consequences of $D_{7.5}$ in the reduced language.

either because its predecessor has just fallen down or because it was already down and has remained so by inertia. To see this more clearly, imagine the following scenario: all of the dominoes are initially up, we tip domino 1 and perform no other actions. Then, whether we include (7.25) or (7.26), it will follow that all of the dominoes will be down at time 5 (cf. the first of the above-mentioned consequences). However, if we were to replace (7.25) by (7.26), it would be impossible (even if such an action were added to the domain description) to stand the dominoes back up in reverse order. This is not what we intend.[14]

7.5.5 Non-literal Consequents

In the preceding examples, we have seen causal laws with non-literal consequents used in explicit definitions—(7.11) and (7.23)—and in connection with the specification of nondeterminism—(7.18). The following example is of a different kind. The domain is due to Marc Denecker (personal communication).

Example 7.6 Imagine that there are two gears, each powered by a separate motor. There are switches that toggle the motors on and off, and there is a button that moves the gears so as to connect or disconnect them from one another. The motors turn the gears in opposite (i.e., compatible) directions. A gear is caused to turn if either its motor is on or it is connected to a gear that is turning.

In our formalization of the Gears domain, the signature is specified by the following grammar.

$$\begin{aligned}
&\mathit{index} \ ::= \ 1 \mid 2 \\
&\mathit{switch} \ ::= \ S(\mathit{index}) \\
&\mathit{gear} \ ::= \ G(\mathit{index}) \\
&\mathit{action} \ ::= \ \mathit{Toggle}(\mathit{switch}) \mid \mathit{Push}
\end{aligned}$$

[14]See [Van Belleghem *et al.*, 1996] and [Thielscher, 1996] for related discussions. We present an alternative formalization of the Domino domain as the final example of Section 9.4. In this formalization, we extend the language $\mathcal{L}_{\mathrm{CL}}$ to allow explicit reference to events.

$$\mathit{fluent} ::= \mathit{MotorOn}(\mathit{gear}) \mid \mathit{Connected} \mid \mathit{Turning}(\mathit{gear})$$

$$\mathit{time} ::= 0 \mid 1 \mid \cdots$$

We adopt the following nonstandard inertia assumption.

$$\mathit{inertial\text{-}formula} ::= [\neg]\mathit{MotorOn}(\mathit{gear}) \mid [\neg]\mathit{Connected} \mid \neg\mathit{Turning}(\mathit{gear})$$

Notice that $\mathit{Turning}(G(1))$ and $\mathit{Turning}(G(2))$ are not declared to be inertial. The reason, as we shall suppose, is that in the absence of a cause for turning, friction will cause a gear not to turn. In the terminology of Lifschitz and Rabinov [1989], $\mathit{Turning}(G(1))$ and $\mathit{Turning}(G(2))$ are "momentary fluents." They tend to revert to being false.

The causal theory $D_{7.6}$ for the Gears domain is represented by the standard schemas (7.5)–(7.9), plus the schemas (7.27)–(7.33) below. (Here i is a meta-variable of type *index*.)

$$\mathit{Toggle}(S(i))_t \wedge \mathit{MotorOn}(G(i))_t \Rightarrow \neg\mathit{MotorOn}(G(i))_{t+1} \tag{7.27}$$

$$\mathit{Toggle}(S(i))_t \wedge \neg\mathit{MotorOn}(G(i))_t \Rightarrow \mathit{MotorOn}(G(i))_{t+1} \tag{7.28}$$

$$\mathit{Push}_t \wedge \mathit{Connected}_t \Rightarrow \neg\mathit{Connected}_{t+1} \tag{7.29}$$

$$\mathit{Push}_t \wedge \neg\mathit{Connected}_t \Rightarrow \mathit{Connected}_{t+1} \tag{7.30}$$

$$\mathit{MotorOn}(G(i))_t \Rightarrow \mathit{Turning}(G(i))_t \tag{7.31}$$

$$\mathit{Connected}_t \Rightarrow \mathit{Turning}(G(1))_t \equiv \mathit{Turning}(G(2))_t \tag{7.32}$$

$$\neg\mathit{Turning}(G(i))_t \Rightarrow \neg\mathit{Turning}(G(i))_t \tag{7.33}$$

Schemas (7.27)–(7.30) describe the explicit effects of toggling the switches and pushing the button. Schema (7.31) says that a gear's motor being on causes it to turn. Schema (7.32) says that the gears being connected causes them to turn (and not turn) together. Schema (7.33) expresses the momentary nature of turning, that is, the natural tendency of the gears not to turn.

The formulas

$$[\bigvee_{1=1..2} \mathit{MotorOn}(G(i))_0] \wedge [\bigwedge_{1=1..2} \neg \mathit{Toggle}(S(i))_0] \wedge \neg \mathit{Connected}_0 \wedge \mathit{Push}_0$$
$$\supset \bigwedge_{1=1..2} \mathit{Turning}(G(i))_1$$
$$[\bigvee_{1=1..2} \neg \mathit{MotorOn}(G(i))_0] \wedge [\bigwedge_{1=1..2} \neg \mathit{Toggle}(S(i))_0] \wedge \mathit{Connected}_0 \wedge \mathit{Push}_0$$
$$\supset \bigvee_{1=1..2} \neg \mathit{Turning}(G(i))_1$$

are consequences of $D_{7.6}$. $\diamond$

According to the "physics" of the Gears domain, as we imagine it, the natural tendency of the gears is to remain from one time to the next—connected or disconnected—as they are. Behavior in accordance with this tendency is explained by the inertia laws for *Connected* and $\neg$*Connected*, and so need not be otherwise explained. What remain to be explained are only the deviations from this natural tendency, i.e., changes in the state of connectedness of the gears.

The momentary character of turning is a fact of a similar kind. According to the "physics" of the Gears domain, as we imagine it, the natural tendency of the gears from one time to the next is to stop turning. Behavior in accordance with this tendency is explained by the "momentary laws" (represented by (7.33)) for $\neg \mathit{Turning}(G(1))$ and $\neg \mathit{Turning}(G(2))$, and so need not be otherwise explained. What remain to be explained are only the deviations from this natural tendency, i.e., the gears turning.[15]

Causal statements with non-literal consequents, such as (7.32), have been a point of some confusion in the AI literature. Often they are disallowed in formal languages for describing actions, and when they are allowed, they tend to be viewed

[15] In addition to inertial and momentary fluents, we should also mention as a third class of fluents, the class of *exogenous* fluents. Just as the inertia laws provide standard explanations for persistence and the momentary laws provide standard explanations for falsity, so the exogenous fluent laws—which are analogous to (7.5)–(7.6)—provide standard explanations for both truth and falsity, thus leaving nothing to explain. The three classes of fluents correspond to the three kinds of standard support listed in Section 6.4.

as a means of expressing nondeterminism. This view is encouraged by the fact that in natural language we tend to describe nondeterministic actions by statements in which "or" is used after "causes," as in: *Tossing a coin causes it to land heads or tails.* It is further encouraged by the fact that in some cases non-literal consequents indeed do give rise to nondeterminism, although this often comes about only because certain noninertial fluents are mistakenly specified as inertial, as illustrated in Example 4.4. In the language of causal theories, the connection between non-literal consequents and nondeterminism is not particularly close. Notice, for example, that (7.32) is not used to express nondeterminism.

7.6 Language Dependence and Inertia

In each of the preceding examples, every inertial fluent has been designated by a fluent literal. In all examples but the last, every fluent literal has also been inertial. In the next example, we show that it is sometimes necessary—depending on the fluent language—to designate inertial fluents by complex formulas.

Example 7.7 Let us reformalize the Coin Tossing domain so that, instead of assuming that the coin is always lying heads or tails, we admit also the third possibility that it might be balanced on its edge. We will continue to assume that tossing the coin causes it land either heads or tails, but we will now allow the coin to be stood on its edge by an explicit action of the agent.

One possible language is given by the following grammar.

$$\begin{aligned} &\mathit{action} ::= \mathit{Toss} \mid \mathit{Stand\_on\_Edge} \\ &\mathit{fluent} ::= \mathit{Heads} \mid \mathit{Tails} \\ &\mathit{time} ::= 0 \mid 1 \mid \cdots \end{aligned}$$

In this language we have fluent names that correspond to two of the three possible states of the coin. We do not have a fluent name which corresponds to the state in which the coin is standing on its edge. Rather, the coin's being in this state is represented by the fluent formula $\neg \mathit{Heads} \wedge \neg \mathit{Tails}$. Since the coin can persist in any of its three states by inertia, we specify the following inertial fluents.

$$\mathit{inertial\text{-}formula} ::= \mathit{Heads} \mid \mathit{Tails} \mid \neg \mathit{Heads} \wedge \neg \mathit{Tails}$$

Notice that the complements of the above-mentioned formulas—$\neg \mathit{Heads}$, $\neg \mathit{Tails}$, and $\neg(\neg \mathit{Heads} \wedge \neg \mathit{Tails})$—do not designate inertial fluents. This is because none of these formulas corresponds to the coin's being in one of its three possible states. (Intuitively, it is these states that can persist by inertia.) Instead, each of them corresponds to the coin's being in either of two states. For example, $\neg \mathit{Tails}$ is true when the coin is either lying heads or on its edge. To say that $\neg \mathit{Tails}$ is inertial would mean that its truth is explained by inertia even when, for example, the state of the coin changes from standing on its edge to lying heads. Intuitively, this is incorrect (cf. our discussion of coin tossing in Section 4.3).

The causal theory $D_{7.7}$ is represented by the standard schemas (7.5)–(7.9), plus the schemas (7.34)–(7.38) below.

$$\mathit{True} \Rightarrow \neg \mathit{Heads}_t \vee \neg \mathit{Tails}_t \tag{7.34}$$

$$\mathit{Stand\_on\_Edge}_t \Rightarrow \neg \mathit{Heads}_{t+1} \wedge \neg \mathit{Tails}_{t+1} \tag{7.35}$$

$$\mathit{Toss}_t \Rightarrow \mathit{Heads}_{t+1} \vee \mathit{Tails}_{t+1} \tag{7.36}$$

$$\mathit{Toss}_t \wedge \mathit{Heads}_{t+1} \Rightarrow \mathit{Heads}_{t+1} \tag{7.37}$$

$$\mathit{Toss}_t \wedge \mathit{Tails}_{t+1} \Rightarrow \mathit{Tails}_{t+1} \tag{7.38}$$

According to schema (7.34), the coin is always caused to exist in one of its three states. Schema (7.35) describes the effect of standing the coin on its edge. Schemas (7.36)–(7.38) describe the possible effects of tossing the coin. According to schema (7.36), whenever the coin is tossed, it is caused to land either heads or tails. According to schemas (7.37) and (7.38), the coin is possibly caused to land heads and possibly caused to land tails. $\diamond$

Example 7.7 shows that it is not necessary to choose a fluent language in which every inertial fluent is named by a fluent literal. Often, however, it is natural to do so. If we wish, we can introduce a new fluent *Edge* as an abbreviation for the fluent $\neg \mathit{Heads} \wedge \neg \mathit{Tails}$ as follows.

Example 7.8 The signature is specified by the following grammar.

$$\begin{aligned} &\mathit{action} ::= \mathit{Toss} \mid \mathit{Stand\_on\_Edge} \\ &\mathit{fluent} ::= \mathit{Heads} \mid \mathit{Tails} \mid \mathit{Edge} \\ &\mathit{time} ::= 0 \mid 1 \mid \cdots \end{aligned}$$

We now specify the inertial fluents as follows.

$$\mathit{inertial\text{-}formula} ::= \mathit{Heads} \mid \mathit{Tails} \mid \mathit{Edge}$$

The causal theory $D_{7.8}$ for our new version of the Coin Tossing domain is represented by the standard schemas (7.5)–(7.9), plus the schemas (7.39)–(7.44) below.

$$\mathit{True} \Rightarrow \mathit{Edge}_t \equiv \neg \mathit{Heads}_t \wedge \neg \mathit{Tails}_t \tag{7.39}$$

$$\mathit{True} \Rightarrow \neg \mathit{Heads}_t \vee \neg \mathit{Tails}_t \tag{7.40}$$

$$\mathit{Stand\_on\_Edge}_t \Rightarrow \mathit{Edge}_{t+1} \tag{7.41}$$

$$\mathit{Toss}_t \Rightarrow \mathit{Heads}_{t+1} \vee \mathit{Tails}_{t+1} \tag{7.42}$$

$$\mathit{Toss}_t \wedge \mathit{Heads}_{t+1} \Rightarrow \mathit{Heads}_{t+1} \tag{7.43}$$

$$\mathit{Toss}_t \wedge \mathit{Tails}_{t+1} \Rightarrow \mathit{Tails}_{t+1} \tag{7.44}$$

The new domain description is obtained from the previous one by adding the explicit definitions represented by (7.39) and the following new instances of schemas (7.7) and (7.8)

$$\mathit{Edge}_0 \Rightarrow \mathit{Edge}_0$$

$$\neg \mathit{Edge}_0 \Rightarrow \neg \mathit{Edge}_0$$

and by substituting $Edge_t$ (for all times t) for its definiens in the causal laws represented by (7.35) and the inertia schemas. It can be shown, using the results of Section 5.5, that $D_{7.8}$ and $D_{7.7}$ have the same consequences in the language of $D_{7.7}$.

$\diamond$

What is inertia? Is it merely a communication or representation convention? Or is there some real or imagined physical reality that underlies the claim that a fluent is inertial? In formalizing domains as causal theories, we take the latter point of view. Our aim is to correctly represent the causes of facts (or the conditions under which facts are caused) according to the "physics" of the domain, as we imagine it. It is common in action description formalisms to build in the assumption that every fluent literal designates an inertial fluent, and it is nearly universally assumed that every inertial fluent is designated by a fluent literal. Neither assumption is built into the language of causal theories. As a consequence, the language of causal theories provides a degree of language independence that is uncommon in action description languages.[16]

[16]Typically, the phenonomenon of language dependence in action description formalisms has gone unremarked upon in the AI literature. A notable exception is [Winslett, 1988].

Chapter 8

Two Action Query Languages

In this chapter, we define two action query languages for use with the language $\mathcal{L}_{\text{CL}}$. The first and simplest of these can be used to express facts and queries about the actual world. The second action query language includes modal operators for historical necessity and possibility. In this language, we are able to pose queries that concern not only the actual world, but other causally possible worlds as well.

With respect to the first (non-modal) query language, we lay the theoretical foundations for an approach to automated query answering and planning that is based on satisfiability checking in propositional logic. This approach is described and illustrated in Chapter 9.

8.1 Action Query Languages for $\mathcal{L}_{\text{CL}}$

In Chapter 4, we described the distinction, due to Lifschitz [1995], between an action description language which is used to specify a transition system and an action query language which is used to describe properties of paths (representing causally possible world histories) in a transition system. Recall that the central semantic definition associated with an action query language, according to Lifschitz, is the definition of the consequence relation that holds between a set Γ of axioms and a query Q

relative to a transition system T; in symbols, $\Gamma \vdash_T Q$.

Although we view the language $\mathcal{L}_{\rm CL}$ as an action description language, unlike the action description languages described by Liftschitz, an $\mathcal{L}_{\rm CL}$ domain description does not specify a transition system. Instead, it specifies a set of interpretations. These interpretations, like the paths of a transition system, represent causally possible world histories. Thus, despite the different ways in which causally possible world histories are represented, domain descriptions in both kinds of action description languages can be said to specify the same kinds of informal objects. The difference in representation, however, gives rise to a corresponding difference in how the consequence relation for an action query language is specified in the two frameworks; in particular, the transition system T in $\Gamma \vdash_T Q$ is replaced in an $\mathcal{L}_{\rm CL}$ (world histories) framework by a set S of interpretations. In defining the semantics of each of the action description languages defined below, we will write $\Gamma \vdash_S Q$ to say that a query Q is a consequence of a set Γ of axioms, relative to a set S of interpretations.

8.2 The Query Language $\mathcal{L}_{\rm A}$: Actuality

In this section, we define an action query language $\mathcal{L}_{\rm A}$ that can be used to express facts and queries about the actual world. We also lay the foundations for an approach to automated query answering and planning (in the combined language $\mathcal{L}_{\rm CL} + \mathcal{L}_{\rm A}$) for subclasses of $\mathcal{L}_{\rm CL}$ domain descriptions.

The signature for a specific $\mathcal{L}_{\rm A}$ language is specified, as it is for an $\mathcal{L}_{\rm CL}$ language, by a triple $\langle \mathbf{A}, \mathbf{F}, \mathbf{T} \rangle$, where $\mathbf{A}$ is a set of action names, $\mathbf{F}$ is a nonempty set of fluent names, and $\mathbf{T}$ is a nonempty set of time names. The set of atoms is the set of expressions of the forms a_t and f_t, where a, f, and t are action, fluent, and time names, respectively. By a *formula* of $\mathcal{L}_{\rm A}$ we mean a propositional combination of atoms. By an *axiom* or a *query* of $\mathcal{L}_{\rm A}$ we simply mean a formula.

Let Γ be a set of axioms, S be a set of interpretations of $\mathcal{L}_{\rm A}$, and Q be a query. We say that Q is a *consequence of* Γ *in* S—in symbols, $\Gamma \vdash_S Q$—if Q is true in every interpretation in S that is a model of Γ.

Given specific $\mathcal{L}_{\mathrm{CL}}$ and $\mathcal{L}_{\mathrm{A}}$ languages with the same signature, the consequence relation for the combined language $\mathcal{L}_{\mathrm{CL}} + \mathcal{L}_{\mathrm{A}}$ is defined as follows. Let D be a domain description in $\mathcal{L}_{\mathrm{CL}}$, and let Γ and Q be a set of axioms and a query of $\mathcal{L}_{\mathrm{A}}$, respectively. We say that Q is a *consequence of* Γ *according to* D (in symbols, $\Gamma \vdash_D Q$) if $\Gamma \vdash_S Q$, where S is the set of causally explained interpretations according to D.

Notice that if Γ is finite, then $\Gamma \vdash_D Q$ if and only if $(\bigwedge \Gamma \supset Q)$ is a consequence of D.

8.2.1 Query Answering

The following proposition justifies an approach to automated query answering in the combined language $\mathcal{L}_{\mathrm{CL}} + \mathcal{L}_{\mathrm{A}}$ with respect to finite $\mathcal{L}_{\mathrm{CL}}$ objective programs.

Proposition 8.1 *Let D be a finite $\mathcal{L}_{CL}$ objective program. Let Γ be a set of axioms and Q be a query in the underlying propositional language of D. Then $\Gamma \vdash_D Q$ if and only if $lcomp(D) \cup \Gamma \models Q$.*

Proof. Since D is finite, it is completable. Therefore, by Proposition 6.1, the causally explained interpretations according to D are the models of $lcomp(D)$. It follows that $\Gamma \vdash_D Q$ if and only if Q is true in every model of $lcomp(D) \cup \Gamma$, and thus if and only if $lcomp(D) \cup \Gamma \models Q$. $\square$

The following corollary suggests an approach to query answering that is based on satisfiability checking in propositional logic.

Corollary 8.1 *Under the assumptions of Proposition 8.1, $\Gamma \vdash_D Q$ if and only if $lcomp(D) \cup \Gamma \cup \{\neg Q\}$ is unsatisfiable.*

8.2.2 Satisfiability Planning

In this section, we lay the theoretical foundations for satisfiability planning (in the style of Kautz and Selman [1992,1996]) with respect to the class of finite $\mathcal{L}_{\rm CL}$ objective programs in which all actions are "deterministic."

By a *complete initial state description*, we mean a set Γ_0 of formulas—each a propositional combination of atoms of the form f_0, where f is a fluent name—such that for every atom of the form f_0, either $\Gamma_0 \vdash_D f_0$ or $\Gamma_0 \vdash_D \neg f_0$, but not both.

By a *time-specific goal*, we mean a propositional combination of atoms of the form f_t, where f is a fluent name. We do not require all of the atoms in a time-specific goal to refer to the same time.

By a *plan* we mean a consistent set of literals of the forms a_t and $\neg a_t$, where a and t are action and time names, respectively.

The notion of a plan, as just defined, differs from the common notion (according to which a plan is a sequence of actions) in two respects.[1] First, a plan, as defined here, may include actions performed concurrently; zero or more actions may be performed at a time. Second, a plan may incompletely specify which actions occur and do not occur at a time. Thus, for some action a and time t (in a plan), the plan may include neither a_t nor $\neg a_t$.

Let D be a finite $\mathcal{L}_{\rm CL}$ objective program, Γ_0 be a complete initial state description, P be a plan, and G be a time-specific goal. We say that P is *executable* if there is a causally explained interpretation according to D that satisfies $\Gamma_0 \cup P$.[2]

[1]In [McCarthy and Hayes, 1969] and [Manna and Waldinger, 1987], a plan is not necessarily a sequence of actions, but may have a more complex structure. Constructing such plans is in some respects a more general and more difficult problem than the one we address here.

[2]This is a very weak notion of executability. In particular, even if P completely specifies which actions occur and do not occur at all times, it may be inadequate in the presence of nondeterministic actions. Informally, this can be seen by considering the plan that calls for one to first toss a coin and then truly report that it has landed heads. Since (regardless of the initial state) there is a causally possible world history that conforms to the execution of this plan, the plan is executable in the sense defined. Intuitively, however, this does not guarantee that the plan can be executed, since if the coin should land tails instead of heads, the second action would be impossible.

We say that P is *effective* if

$$\Gamma_0 \cup P \vdash_D G. \tag{8.1}$$

By Corollary 8.1, we know under the conditions stated above that the executability and effectiveness of P can be verified by satisfiability checking. The planning problem, however, is not the problem of verifying that a plan is executable and effective, but rather the problem of *finding* a plan that can be so verified. The realization of Kautz and Selman [1992] was that for some syntactically defined classes of theories the problem of finding such a plan also can be solved by satisfiability checking. They describe a class with this property in the language of propositional logic. In the remainder of this section, we do the same in the language of causal theories.

Let D be an objective program. The *atom dependency graph* of D is the directed graph which has atoms as nodes, and which, for each rule $L_0 \leftarrow L_1, \ldots, L_n$ in D, has an edge from the atom in L_0 to each of the atoms in $L_1, \ldots, L_n$.

An atom dependency graph defines an ordering between nodes as follows. We say that a node m *is less than* a node n (in symbols, $m < n$) if there is a path containing at least one edge from n to m in the graph. So the edges in the graph point downward in the ordering.

We say that an $\mathcal{L}_{\text{CL}}$ domain description D is *simple* if it satisfies the following conditions.

1. In the signature of D, time is defined as the natural numbers or an initial segment of the natural numbers, i.e., for some n,

$$\mathit{time} \;:=\; 0 \;\mid\; 1 \;\mid\; \cdots \;\mid\; n.$$

2. D is an objective program, and $D = D_x \cup D_i \cup D'$, where D_x is the set of rules represented by the schemas (7.5)–(7.8), D_i is the set of rules represented by the inertial schema (7.9), and every rule in D' has the form

$$[\neg]f_t \;\leftarrow\; [\neg]x^1_{t_1}, \ldots, [\neg]x^n_{t_n}$$

where f is a fluent name and, for all i $(1 \leq i \leq n)$, x^i is either an action or fluent name, and t is greater than or equal to t_i. (Intuitively, the last condition rules out backward causation.)

3. The ordering relation defined by the atom dependency graph of D' is well-founded.

The domain description for the Suitcase domain is simple. The domain description for the Coin Tossing domain in Section 7.5.2 violates condition (3), and thus is not simple.

By an *action history* we mean a set Γ_a of literals such that for every action name a and time t, Γ_a contains exactly one of the literals a_t and $\neg a_t$, and contains no other literals besides these. An action history specifies, for every time t, exactly which actions occur at t.

Intuitively, every action in a simple $\mathcal{L}_{\text{CL}}$ domain description is deterministic. This is made precise in the following proposition.

Proposition 8.2 *Let D be a simple $\mathcal{L}_{CL}$ domain description. Let Γ_0 be a complete initial state description and Γ_a be an action history. There is at most one causally explained interpretation according to D that satisfies $\Gamma_0 \cup \Gamma_a$.*

Proof. Since D is a simple $\mathcal{L}_{\text{CL}}$ domain description, we know that $D = D_x \cup D_i \cup D'$, where D_x, D_i, and D' are as defined in (2) above. Let H be the atom dependency graph of D'. We proceed by the method of contradiction. Suppose that there are two causally explained interpretations of D that satisfy $\Gamma_0 \cup \Gamma_a$. Call them I and I'. We know that I and I' assign different values to one or more nodes in H. Select A to be such a node whose time subscript t is minimal among all such nodes and such that A is minimal in the ordering relation defined by H. (We know that this selection is possible because D is simple.) Without loss of generality, let us suppose that $I \models A$ and $I' \models \neg A$. By the definition of a supported model for objective programs, there must be rules, $A \leftarrow B$ and $\neg A \leftarrow B'$, in D such that $B \subseteq I$

but $B \not\subseteq I'$, and $B' \not\subseteq I$ but $B' \subseteq I'$. Since I and I' both satisfy Γ_0, we know that they agree on the values of all atoms of the form f_0, where f is a fluent name. (Recall that Γ_0 is complete with respect to such atoms.) Similarly, since I and I' both satisfy Γ_a, we know that they agree on the values of all atoms of the form a_t, where a and t are action and time names, respectively. It follows that A must be an atom of the form f_{t+1}, for some fluent name f. Thus, the rules $A \leftarrow B$ and $\neg A \leftarrow B'$ in fact belong to $D_i \cup D'$. By our choice of A, we know that neither rule can belong to D' (because otherwise I and I' assign different values to some atom in B or some atom in B', but all such atoms are less than A in the ordering). Nor, for a similar reason, can both belong to D_i (because otherwise I and I' assign different values to f_t). Thus, there is no such node A to which I and I' assign different values. Consequently, there is at most one causally explained interpretation of D that satisfies $\Gamma_0 \cup \Gamma_a$. □

Corollary 8.2 *Let D be a simple $\mathcal{L}_{CL}$ domain description, Γ_0 be a complete initial state description, and G be a time-specific goal. Suppose that I is a causally explained interpretation according to D that satisfies $\Gamma_0 \cup G$. Let Γ_a be the action history contained in I. Then*

$$\Gamma_0 \cup \Gamma_a \vdash_D G. \tag{8.2}$$

Proof. By Proposition 8.2, we know that I is the only causally explained interpretation according to D that satisfies $\Gamma_0 \cup \Gamma_a$. By hypothesis, I also satisfies G. Therefore, every causally explained interpretation according to D that satisfies $\Gamma_0 \cup \Gamma_a$ satisfies G, that is, $\Gamma_0 \cup \Gamma_a \vdash_D G$. □

8.3 The Query Language $\mathcal{L}_{\text{H}}$: Historical Necessity

In this section we define an action query language $\mathcal{L}_{\text{H}}$ that can be used for expressing facts about the actual world and time-dependent modal queries about both the actual world and other causally possible worlds.

According to common sense, there is an asymmetry between the past and present on the one hand and the future on the other. Whereas the past and present cannot now be other than they are, the future in at least some respects can be.[3] This asymmetry cannot be explained in terms of epistemic possibility, since we may be equally ignorant of the past, present, and future. Instead, it is explained in terms of the time-dependent modalities of historical necessity and possibility.

Intuitively, a proposition becomes *historically necessary* when the present and the past determine that it is true, and it remains *historically possible* so long as the present and past do not determine that it is false.

The concepts of historical necessity and possibility have been formalized in the framework of modal-temporal logic by [Montague, 1968], [Chellas, 1971], and [Kamp, 1979]. In these logics, formulas are understood to designate time-dependent propositions and are evaluated with respect to a world and a time. In particular, the accessibility relation for the historical necessity operator is defined as a function of time. Since we are working in a framework in which propositions are not time-dependent (the atoms of an action query language contain their own fixed time references), we will find it convenient to proceed in a different manner. Instead of introducing a single time-dependent necessity operator $\Box$, we will introduce a distinct operator $\boxed{n}$, for each time n.

The signature for an $\mathcal{L}_{\mathrm{H}}$ language, as for an $\mathcal{L}_{\mathrm{CL}}$ language, is specified by a triple $\langle \mathbf{A}, \mathbf{F}, \mathbf{T} \rangle$, where $\mathbf{A}$ is a set of action names, $\mathbf{F}$ is a nonempty set of fluent

[3]In speaking in this manner, we assume that even in a nondeterministic world there is a unique actual future and that every proposition about it is presently true or presently false, even if it is undetermined (by past and present facts) which it is.

The point at issue is the one discussed by Aristotle in a famous passage from *De Interpretatione* (Chapter 9). Assuming that it is not presently determined whether or not there will be a sea battle tomorrow, Aristotle asks whether it is true now that there will be a sea battle or true now that there will not be. Although the passage is not entirely clear, he seems to deny that either is the case. We look at this differently. The openness of the future (as opposed to the present and past) lies not, we maintain, in there being propositions about the future that are neither presently true nor presently false, but in there being propositions about the future that are neither presently determined to be true nor presently determined to be false.

names, and **T** is a nonempty set of time names. The set of atoms is the set of expressions of the forms a_t and f_t, where a, f, and t are action, fluent, and time names, respectively. The set of *formulas* of $\mathcal{L}_{\mathrm{H}}$ is the smallest set that contains the atoms and is closed under the following rules: if ϕ is a formula, then $\neg\phi$ is a formula; if ϕ and ψ are formulas, then $\phi \wedge \psi$ is a formula; and if ϕ is a formula, then $\boxed{\scriptstyle n}\,\phi$ is a formula, for every time name n. We define $\diamond_{n}\phi$ as an abbreviation for $\neg\boxed{\scriptstyle n}\neg\phi$. We assume that the other standard propositional connectives ($\vee$, $\supset$, and $\equiv$) are defined as abbreviations in the usual way.

Let $\mathcal{L}$ be the sublanguage of $\mathcal{L}_{\mathrm{H}}$ that consists of all formulas that do not contain a modal operator. Since $\mathcal{L}$ is a language of propositional logic, the notion of an interepretation of $\mathcal{L}$ is defined. By an *axiom* we mean a formula of $\mathcal{L}$. By a *query* we mean a formula of $\mathcal{L}_{\mathrm{H}}$.

We will identify an interpretation I with the set of literals L such that $I \models L$. Let I be an interpretation of $\mathcal{L}$, and m be a time. By $I|m$ we mean the set of all literals in I of either the form $[\neg]f_k$, where f is a fluent name and $k \leq m$, or of the form $[\neg]a_k$, where a is an action name and $k < m$.

A structure is a pair (I, S), where I is an interpretation of $\mathcal{L}$ and S is a set of such interpretations with $I \in S$. We define the conditions under which a structure (I, S) satisfies a formula ϕ (in symbols, $(I, S) \models \phi$) as follows.

$$(I,S) \models \phi \text{ iff } \phi \in I, \text{ if } \phi \text{ is an atom}$$

$$(I,S) \models \neg\phi \text{ iff } (I,S) \not\models \phi$$

$$(I,S) \models \phi \wedge \psi \text{ iff } (I,S) \models \phi \text{ and } (I,S) \models \psi$$

$$(I,S) \models \boxed{\scriptstyle n}\,\phi \text{ iff } (I',S) \models \phi, \text{ for all } I' \in S \text{ s.t. } I'|m = I|m.$$

Intuitively, according to the last clause, $\boxed{\scriptstyle n}\,\phi$ is true in a world I if and only if ϕ is true in all worlds that coincide with I at all times up to and including time n. Notice that these worlds are not required to coincide with I on facts about the actions that occur, as we say, at time n, but only at earlier times.[4]

[4]In this regard, recall that an atom a_t, where a is an action name and t is a time name, is

Let ϕ be a formula and Γ be a set of formulas. We say that (I, S) is a *model* of ϕ if $(I, S) \models \phi$. We say that (I, S) is a *model* of Γ if (I, S) is a model of every formula in Γ. We say that ϕ is a *consequence* of Γ (in symbols, $\Gamma \models \phi$) if every model of Γ is a model of ϕ.

It is not difficult to see that for all formulas ϕ and all times r and n, if $r \leq n$ then $\boxed{\mathrm{r}}\phi \models \boxed{\mathrm{n}}\phi$ and $\Diamond_{\mathrm{n}}\phi \models \Diamond_{\mathrm{r}}\phi$. This reflects the commonsense belief that with the passage of time more tends to become necessary and less possible.

Let Γ be a set of axioms, S be a set of interpretations of $\mathcal{L}$, and Q be a query. We say that Q is a *consequence of* Γ *in* S (in symbols, $\Gamma \vdash_S Q$) if every model (I, S) of Γ is a model of Q.

Given specific $\mathcal{L}_{\mathrm{CL}}$ and $\mathcal{L}_{\mathrm{H}}$ languages with the same signature, the consequence relation for the combined language $\mathcal{L}_{\mathrm{CL}} + \mathcal{L}_{\mathrm{H}}$ is defined as follows. Let D be a domain description in $\mathcal{L}_{\mathrm{CL}}$, and let Γ and Q be a set of axioms and a query of $\mathcal{L}_{\mathrm{H}}$, respectively. We say that Q is a *consequence of* Γ *according to* D (in symbols, $\Gamma \vdash_D Q$) if $\Gamma \vdash_S Q$, where S is the set of causally explained interpretations according to D.

The modalities of historical necessity and possibility are dependent upon the past and present facts of the actual world. As an illustration, let D be the description of the Domino domain from Example 7.5, modified so that the only possible action is to tip domino 1, i.e., $\mathit{Tip}(1)$. Let

$$\Gamma = \{ \bigwedge_{i=1..5} \mathit{Up}(i)_0, \bigwedge_{n=0..5} \neg \mathit{Tip}(1)_n \}.$$

Intuitively, in Γ we make the following assertions about the actual world: (i) initially all five dominos are up, and (ii) domino 1 is not tipped at any time. Each of the following is true.

$$\Gamma \vdash_D \bigwedge_{n=0..5} \mathit{Up}(3)_n$$

$$\Gamma \vdash_D \bigwedge_{n=0..2} \Diamond_{\mathrm{n}} \neg \mathit{Up}(3)_5$$

understood to say that the action a occurs immediately after t.

$$\Gamma \vdash_D \bigwedge_{n=3..5} \boxed{\scriptstyle n}\, Up(3)_5$$

These results can be explained as follows. In the actual world, domino 3 remained standing at all times. It was possible before time 3 to cause domino 3 to be down by time 5 (by tipping domino 1 at time 0, 1, or 2). However, by time 3 this was no longer possible.

The previous example shows that the modalities of historical necessity and possibility are useful in formulating queries about action domains even in the absence of nondeterministic actions. Intuitively, the reason for this is that the choices of which (if any) action to perform lead to alternative historically possible worlds, regardless of whether or not the actions themselves do so.

As a second example, we will consider a simple extension to the coin tossing domain of Example 7.2. In this domain, there are two sources of nondeterminism: (i) the decision whether or not to toss the coin, and (ii) the inherent nondeterminism of coin tossing itself.

Example 8.1 Let us formalize a coin tossing game in which one wins by tossing heads twice in a row.

The signature is specified by the following grammar.

$$\begin{aligned} &\mathit{action} ::= \mathit{Toss} \\ &\mathit{fluent} ::= \mathit{Heads} \mid \mathit{Win} \\ &\mathit{time} ::= 0 \mid 1 \mid \cdots \end{aligned}$$

We will adopt the standard inertia assumption.

$$\mathit{inertial\text{-}formula} ::= [\neg]\mathit{fluent}$$

The causal theory for the coin tossing domain $D_{8.1}$ is represented by the standard schemas (7.5)–(7.9), plus the schemas (8.3)–(8.5) below.

$$\mathit{Toss}_t \wedge \mathit{Heads}_{t+1} \Rightarrow \mathit{Heads}_{t+1} \tag{8.3}$$

$$\mathit{Toss}_t \wedge \neg\mathit{Heads}_{t+1} \Rightarrow \neg\mathit{Heads}_{t+1} \tag{8.4}$$

$$\mathit{Toss}_t \wedge \mathit{Heads}_{t+1} \wedge \mathit{Toss}_{t+1} \wedge \mathit{Heads}_{t+2} \Rightarrow \mathit{Win}_{t+2} \tag{8.5}$$

According to schema (8.5), tossing heads twice in a row causes one to win the game.[5]

Now let us suppose that Γ is the following set of formulas.

$$\{\neg \mathit{Win}_0, \mathit{Toss}_0, \mathit{Toss}_1, \neg\mathit{Heads}_1, \mathit{Heads}_2\}$$

Intuitively, according to Γ, among the facts of the actual world are these: we had not already won at time 0, we did toss the coin at times 0 and 1, the first toss came up tails, and the second came up heads. Each of the following is true.

$$\Gamma \vdash_D \overset{0}{\Diamond} \mathit{Win}_2$$

$$\Gamma \vdash_D \neg \overset{1}{\Diamond} \mathit{Win}_2$$

These results can be explained as follows. At time 0, it was possible to throw heads twice in a row. In fact, however, we were unlucky. The first toss did not come up heads, and so already by time 1 we had no chance of winning by time 2. $\Diamond$

[5]Actually, this may be an example of a determination relation that is not, strictly speaking, causal, although it is "causal-like." (See [Kim, 1974] for a discussion of non-causal determination relations.) As an aside, notice that it would be impossible to express the rule for winning in such a straightforward way in a transition system-based language such as $\mathcal{A}_{\mathrm{CL}}$.

Chapter 9

Query Answering and Planning

In this chapter, we describe and illustrate an approach to automated query answering with respect to finite $\mathcal{L}_{\text{CL}}$ objective programs and to planning with respect to simple, finite $\mathcal{L}_{\text{CL}}$ objective programs. The approach is based on satisfiability checking in propositional logic.

9.1 Automated Query Answering

A satisfiability algorithm searches for an interpretation that satisfies a given set Γ of (propositional) formulas. Some satisfiability algorithms search the space of possible interpretations exhaustively and nonredundantly, and in this sense are "systematic." For such algorithms, failing to find a satisfying interpretation shows the unsatisfiability of Γ. For query answering on the basis of Corollary 8.1, we require a systematic satisfiability algorithm.

Currently, one of the best implementations of a systematic satisfiability algorithm is the program called *ntab* (previously known as *tableau*) by Crawford and Auton [1993]. It is an efficient implementation of the Davis-Putnam procedure [Davis and Putnam, 1960] and incorporates some surprisingly effective heuristics. The program *ntab* reads a file of formulas in clausal form (literals are represented

```
:- declare_types
        type(action,[load,wait,shoot]),
        type(fluent,[loaded,alive]),
        type(time,[0..3]),
        type(atom,[o(action,time),h(fluent,time)]).

:- declare_variables
        var(A,action), var(T,time), var(F,fluent).

h(loaded,T+1) <- o(load,T).
-h(alive,T+1) <- o(shoot,T), h(loaded,T).
-h(loaded,T+1) <- o(shoot,T).

o(A,T) <- o(A,T).
-o(A,T) <- -o(A,T).
h(F,0) <- h(F,0).
-h(F,0) <- -h(F,0).

h(F,T+1) <- h(F,T), h(F,T+1).
-h(F,T+1) <- -h(F,T), -h(F,T+1).
```

Figure 9.1: An input file for the Yale Shooting domain

by postive and negative integers) and determines whether or not the given set of clauses is satisfiable. If the set of clauses is satisfiable, the program responds by writing "SAT" and (optionally) a satisfying interpretation. If the set of clauses is unsatisfiable, the program responds by writing "UNSAT."

In Appendix B, we list a Prolog program, called *satp*, which uses *ntab* to do query answering with respect to finite $\mathcal{L}_{\rm CL}$ objective programs. This program takes an input file representing an $\mathcal{L}_{\rm CL}$ domain description, such as that displayed in Figure 9.1. (This is the same program that we previewed in Chapter 1.) The directive `declare_types`, which appears first in Figure 9.1, is used to define a specific $\mathcal{L}_{\rm CL}$ language in a syntactic variant of the grammars illustrated in the previous chapter. The terminal symbols now are not capitalized, but instead are distinguished from the non-terminal symbols only by their failing to appear as the first argument to the `type/2` functor. The directive `declare_variables` is used to declare any meta-variables that we may wish to use in describing the causal theory. Following these two directives, the file contains rules and schemas written in the syntax of objective programs. (Meta-variables are represented as Prolog variables, and thus are written in uppercase here, in contrast to the previous chapter.) An atom such as $Load_2$ is here written as `o(load,2)` and read as: *the action Load occurs at time 2*. An atom such as $Alive_2$ is here written as `h(alive,2)` and read as: *the fluent Alive holds at time 2*. The last two rule schemas say that every fluent literal is inertial.

Notice that there are only finitely many time names in the signature specified in Figure 9.1. This is essential, since otherwise the set of atoms and, therefore, also the literal completion of the causal theory would be infinite.

The three main user-level procedures provided in *satp* are `load_file/1`, `query/0`, and `plan/0`. The procedure `load_file/1` reads in a file such as the one displayed in Figure 9.1 and (using the language specification and the meta-variable declarations) generates the ground causal theory D. It then also generates the clausal form of $lcomp(D)$ and stores the resulting clauses in the Prolog database.

After an input file has been loaded, queries are posed by calling the procedure `query/0`. This procedure reads from the terminal a set Γ of axioms and a query Q and converts $\Gamma \cup \{\neg Q\}$ to clausal form. The clauses of

$$lcomp(D) \cup \Gamma \cup \{\neg Q\} \tag{9.1}$$

are then written to a file, and *ntab* is called with this file as input. After the call to *ntab* returns, the procedure `query/0` reads the output file generated by *ntab*, decodes it (replacing integers by symbolic literals), and reports the results. The procedure `query/0` answers "yes"—meaning $\Gamma \vdash_D Q$—if *ntab* answers "UNSAT," and answers "no"—meaning $\Gamma \not\vdash_D Q$—if *ntab* answers "SAT." These answers are justified by Corollary 8.1. In the case of a "no" answer, the intepretation found by *ntab* to satisfy (9.1) is displayed as a counterexample.

Example calls to the procedures `load_file/1` and `query/0` are shown in

Figure 9.2.[1] The first query in Figure 9.2 poses the question: Will the turkey be dead in the state that results from executing the actions load, wait, and shoot (in that order), given that the gun is initially unloaded and the turkey is alive? As noted in Section 1.3, we do not assume that load, wait, and shoot are the only actions performed at times 0, 1, and 2. Therefore, this is not precisely the temporal projection problem considered by Hanks and McDermott [1987]. The second query asks whether it follows from the fact that the action of shooting the gun is performed at times 0, 1, and 2 that the turkey will be dead at time 3. The answer of course is "no," since, as the displayed interpretation shows, we have said nothing to guarantee that the gun is ever loaded. Additional examples of query answering are included in Section 9.4.

9.2 Automated Planning

In this section we describe and illustrate the procedure `plan/0`, which implements our approach to satisfiability planning (Kautz and Selman [1992, ,1996]) with respect to simple, finite $\mathcal{L}_{\text{CL}}$ objective programs.

Let D be such a program. Assume that the input file for D (for example, the file displayed in Figure 9.1) has already been loaded using `load_file/1`. The procedure `plan/0` reads from the terminal a complete initial state description Γ_0 and a time-specific goal G, and converts $\Gamma_0 \cup \{G\}$ to clausal form. The clauses of

$$\mathit{lcomp}(D) \cup \Gamma_0 \cup \{G\} \tag{9.2}$$

are then written to a file, and *ntab* is called with this file as input. If *ntab* answers "UNSAT," then by Corollary 8.1 we know that $\Gamma_0 \vdash_D \neg G$, which means that it is impossible to achieve the time-specific goal G, given Γ_0. In this case, `plan/0`

[1]The fact that there are fewer clauses than rules in this example is due to the elimination of tautologies. Also, in case the reader should wish to check the counts reported by `load_file/1`, we note that, as a convenience, the atoms `true` and `false` and the rules `true <-` and `-false <-`, are automatically added by `load_file/1` to every input domain description.

```
| ?- load_file(yale).
% 22 atoms, 51 rules, 26 clauses loaded.
yes
| ?- query.
enter facts (then ctrl-d)
|: h(alive,0).
|: -h(loaded,0).
|: o(load,0).
|: o(wait,1).
|: o(shoot,2).
|:
enter query
|: -h(alive,3).

yes
| ?- query.
enter facts (then ctrl-d)
|: o(shoot,0).
|: o(shoot,1).
|: o(shoot,2).
|:
enter query
|: -h(alive,3).

0. -loaded  alive
Action(s): shoot

1. -loaded  alive
Action(s): shoot

2. -loaded  alive
Action(s): shoot

3. -loaded  alive
Action(s):

no
```

Figure 9.2: Query answering

answers "no." If *ntab* answers "SAT," then the interpretation I that was found to satisfy (9.2) is displayed. The action history Γ_a contained in I can be read off from this display. If Γ_0 is, indeed, a complete initial state description, then we know by the fact that I satisfies (9.2) that Γ_a is executable. If, moreover, D is simple, then by Corollary 8.2 we also know that Γ_a is effective.

Two example calls to `plan/0` are displayed in Figure 9.3. In the first of these, we assume that initially the turkey is alive and the gun is unloaded. We pose the problem of finding a plan to achieve the goal of the turkey being dead and the gun being loaded at time 3. A plan is found which calls for performing the actions load, shoot, and load (and no other actions) at the times shown. In the second call to `plan/0` in Figure 9.3, we seek to find a plan to bring the turkey back to life. Unfortunately, there is no way to achieve this goal.

In some cases we may not be sure whether an $\mathcal{L}_{\rm CL}$ domain description is simple, or we may even know that it is not simple, and yet we may hope to find an executable and effective plan for achieving G (given a complete initial state description Γ_0) by the method just described. In such cases, however, even if an interpretation I is found that satisfies $\Gamma_0 \cup \{G\}$, there is no guarantee that it contains an effective plan. Accordingly, if *ntab* finds an interpretation I that satisfies (9.2), then, after displaying it, `plan/0` asks the user whether an attempt should be made to verify that the action history Γ_a contained in I is effective. If the user requests that this be done, *ntab* is then called again to check the satisfibability of

$$lcomp(D) \cup \Gamma_0 \cup \Gamma_a \cup \{\neg G\}. \tag{9.3}$$

If *ntab* returns "UNSAT," then by Corollary 8.1 we know that

$$\Gamma_0 \cup \Gamma_a \vdash_D G$$

which verifies that Γ_a is effective. On the other hand, if *ntab* returns "SAT," then by Corollary 8.1 we know that

$$\Gamma_0 \cup \Gamma_a \not\vdash_D G$$

```
| ?- plan.
enter facts (then ctrl-d)
|: h(alive,0).
|: -h(loaded,0).
|:
enter goal
|: h(loaded,3) & -h(alive,3).

0. -loaded  alive
Action(s): load

1.  loaded  alive
Action(s): shoot

2. -loaded -alive
Action(s): load

3.  loaded -alive
Action(s):

Verify plan?  y
plan verified.

yes
| ?- plan.
enter facts (then ctrl-d)
|: -h(alive,0).
|: -h(loaded,0).
|:
enter goal
|: h(alive,3).

no
```

Figure 9.3: Planning to achieve a time-dependent goal

which shows that Γ_a is not effective. In the latter case, an interpretation satisfying (9.3) is displayed as a counterexample, and `plan/0` gives up.[2]

Since the $\mathcal{L}_{\text{CL}}$ domain description for the Yale shooting domain is simple and the facts given in Figure 9.3 comprise a complete initial state description, the plan verification step in this instance, while reassuring, is unnecessary.

The procedure `plan/0` can be used to pose goals that are not time specific by using a meta-variable for time in place of a specific time in the goal. On backtracking, the program instantiates the meta-variable to successive times. If the time names are declared in the natural order—from those designating smaller numbers to larger—the result is to search for the earliest time at which the goal can be achieved by a process of iterative deepening [Korf, 1985]. This is illustrated in Figure 9.4. The goal is first instantiated to times 0 and 1, before it is finally solved at time 2.

Although we display an entire action history when a plan is found, normally only a part of the action history is necessary for achieving a goal. For instance, in the last example above, only the actions performed at times 0 and 1 are necessary; the load action at time 2 is not. In this case and many others, it is natural to take the plan to be the part of the action history that precedes the time at which the goal is first achieved. However, in some cases—particularly in domains in which there are delayed effects or in which things change by themselves—this part of an action history may be too much. (Consider, for instance, an action that initiates a process of change that inevitably causes a goal to be achieved, regardless of the actions that may be performed after it.) Dealing appropriately with this issue is a topic for future research.

As an illustration of the possibility that a candidate plan may not turn out to

[2]It should be noted that even if a plan is shown to be effective and executable in the weak sense in which we have defined the term, in the event of nondeterminism there is no guarantee that the plan can be carried out. Consider again, for example, the plan that consists in performing precisely the following actions: first toss a coin and then truly report that it has landed heads. Assuming that the goal is for the coin to be lying heads, this an effective plan. However, as previously observed, there is no guarantee that the plan can be carried out, since it is (historically) possible (at time 0) that tossing the coin will result in its landing tails.

```
| ?- plan.
enter facts (then ctrl-d)
|: h(alive,0).
|: -h(loaded,0).
|:
enter goal
|: -h(alive,T).

0. -loaded  alive
Action(s): load

1.  loaded  alive
Action(s): shoot

2. -loaded -alive
Action(s): load

3.  loaded -alive
Action(s):

Verify plan?  y
plan verified.

yes
```

Figure 9.4: Planning to achieve a time-independent goal

```
:- declare_types
        type(action,[toss]),
        type(fluent,[heads]),
        type(time,[0..2]),
        type(atom,[o(action,time),h(fluent,time)]).

:- declare_variables
        var(A,action), var(F,fluent), var(T,time).

h(heads,T+1) <- o(toss,T), h(heads,T+1).
-h(heads,T+1) <- o(toss,T), -h(heads,T+1).

o(A,T) <- o(A,T).
-o(A,T) <- -o(A,T).
h(F,0) <- h(F,0).
-h(F,0) <- -h(F,0).

h(F,T+1) <- h(F,T), h(F,T+1).
-h(F,T+1) <- -h(F,T), -h(F,T+1).
```

Figure 9.5: An input file for the Coin Tossing domain

be effective, consider the $\mathcal{L}_{\mathrm{CL}}$ domain description for nondeterministic coin tossing displayed in Figure 9.5.

The first two rule schemas represent the part of the causal theory that corresponds to the set D' in the definition of a simple $\mathcal{L}_{\mathrm{CL}}$ domain description. Notice that the ordering relation defined by the atom dependency graph for this set is not well-founded. Consequently, this is not a simple $\mathcal{L}_{\mathrm{CL}}$ domain description.

In the session displayed in Figure 9.6, we assert that the coin is not lying heads at time 0 and pose the problem of finding a plan to cause the coin to be lying heads at time 2. A candidate plan is found, namely, the plan of tossing the coin at time 0 and performing no other actions. However, the attempt to verify the plan fails, because tossing the coin at time 0 could just as well result in its landing tails, as the final displayed interpretation shows.

Additional examples of planning—including an example of planning in a

```
| ?- load_file(toss).
% 8 atoms, 18 rules, 6 clauses loaded.
yes
| ?- plan.
enter facts (then ctrl-d)
|: -h(heads,0).
|:
enter goal
|: h(heads,2).

0. -heads
Action(s): toss

1.  heads
Action(s):

2.  heads
Action(s):

Verify plan?  y
verification failed.

0. -heads
Action(s): toss

1. -heads
Action(s):

2. -heads
Action(s):

no
```

Figure 9.6: An unverifiable plan

dynamic domain—are included in Section 9.4.

9.3 Final Remarks on Query Anwering and Planning

Kautz and Selman [1996] have experimentally investigated the application of systematic and unsystematic satisfiability algorithms in planning. Specifically, they have compared the systematic program *tableau* [Crawford and Auton, 1993] and an unsystematic (stochastic) program called *Walksat* [Selman *et al.*, 1994]. They report solving much larger planning problems using *Walksat* than *tableau*. Indeed, at the present time, satisfiability planning using *Walksat* may be the most efficient approach to planning that is known.

Since *Walksat* is unsystematic, it cannot be used for query answering or plan verification, where testing for unsatisfiability is required. It can, however, be used for planning with respect to simple $\mathcal{L}_{\mathrm{CL}}$ domain descriptions, since in this case the plan verification step is unnecessary.[3]

Finally, we wish to emphasize that the restriction in this section to $\mathcal{L}_{\mathrm{CL}}$ domain descriptions that belong to the class of objective programs is a very serious limitation. It is not difficult to find domains that can be properly formalized only by writing causal laws that have non-literal consequents.[4]

There is, however, some reason to hope that the restriction to literal consequents can be relaxed. Lifschitz [1997] has shown how to reformulate the syntax and semantics of causal theories in classical logic. In the propositional case, this reformulation introduces second-order quantifiers. However, it is always possible to eliminate these quantifiers, and doing so again yields a standard propositional

[3]We should remark in this context that our central concern in this dissertation is not with the efficiency of planning, but rather with finding a natural and expressive language for formalizing domains of action and change. The appeal of satisfiability planning methods—as opposed to methods, such the one described in [McAllester and Rosenblitt, 1991], which presupposes STRIPS-like [Fikes and Nilsson, 1971] add and delete lists—is that satisfiability planning methods can be used with any formalism that can be translated into classical propositional logic.

[4]Attempting to formalize such domains by writing causal laws with only literal consequents can lead to unintended causal loops. Compare the Seesaw domain of Example 3.9.

theory. This means that query answering and planning can still, in principle, be carried out by satisfiability checking, even for causal theories that contain laws with non-literal consequents. It remains to be seen, however, whether the process of eliminating the second-order quantifiers can be done with acceptable efficiency, and how their elimination impacts the size of the resulting theory. These are topics for future research.

9.4 Examples

In this section, we present several additional examples showing the behavior of the program *satp*. We will not comment on the individual examples, but rather in each case will simply list the program file and a session in which the program is loaded and used in query answering and planning. In all cases, the run times are no more than a few seconds on a Sun SPARCstation 5.

Meta-variables appearing in the formulas given as "facts" to `query/0` or `plan/0` are implicitly universally quantified, as they are in rule schemas. Meta-variables appearing in a "query" or a "goal" are implicitly existentially quantified. As in rule schemas, it is possible to restrict the instantiations of facts, queries, and goals by using `where` clauses.

In the final two examples of this section—the Airport domain and the second Domino domain—we extend the $\mathcal{L}_{\mathrm{CL}}$ language to include atoms of two new forms. In the Airport domain, we include atoms that describe features of the domain that do not vary with time, and in the second Domino domain, we include atoms of the form `o(event,time)`. Events, like actions, are properly said to occur, and facts about them may be causes of change. Unlike actions, however, events are typically endogenous to our causal theories. In the second Domino domain, we describe the conditions under which falling events occur. In the absence of such conditions, we specify that falling events do not occur by writing causal laws similar to those used to specify momentary fluents.[5]

```
% File: suitcase (see Chapter 7)

:- declare_types
        type(latch,[l1,l2]),
        type(action,[toggle(latch),close]),
        type(fluent,[up(latch),open]),
        type(time,[0..5]),
        type(atom,[o(action,time),h(fluent,time)]).

:- declare_variables
        var(L,latch),
        var(A,action),
        var(F,fluent),
        var(T,time).

-h(up(L),T+1) <- o(toggle(L),T), h(up(L),T).
h(up(L),T+1) <- o(toggle(L),T), -h(up(L),T).
-h(open,T+1) <- o(close,T).
h(open,T) <- h(up(l1),T), h(up(l2),T).

o(A,T) <- o(A,T).
-o(A,T) <- -o(A,T).
h(F,0) <- h(F,0).
-h(F,0) <- -h(F,0).

h(F,T+1) <- h(F,T), h(F,T+1).
-h(F,T+1) <- -h(F,T), -h(F,T+1).
```

```
| ?- load_file(suitcase).
% loading file /v/hank/v18/mccain/d/pl/suitcase
% 38 atoms, 105 rules, 68 clauses loaded.
yes
```

[5]Since in the last two examples of this section we extend the language $\mathcal{L}_{\mathrm{CL}}$, the domain descriptions in these cases are not simple, and so the effectiveness of our plans is not guaranteed by Corollary 8.2. Also, in each of these examples and in the Stuffy Room domain which precedes them, we use the special atom `false`. For this reason also, these examples fall outside the class of simple $\mathcal{L}_{\mathrm{CL}}$ domain descriptions. Presumably, our definitions can be generalized to account for examples such as these, but this remains to be done.

```
| ?- query.
enter facts (then ctrl-d)
|: h(up(l1),0).
|: h(up(l2),0).
|: o(close,0).
|:
enter query
|: o(toggle(l1),0) | o(toggle(l2),0).    % '|' is 'or'

yes
| ?- plan.
enter facts (then ctrl-d)
|: -h(open,0).
|: -h(up(l1),0).
|: -h(up(l2),0).
|:
enter goal
|: h(open,5) & -h(up(l1),5) & -h(up(l2),5).

0. -up(l1) -up(l2) -open
Actions: toggle(l1) toggle(l2)

1.  up(l1)  up(l2)  open
Actions:

2.  up(l1)  up(l2)  open
Actions:

3.  up(l1)  up(l2)  open
Actions:

4.  up(l1)  up(l2)  open
Actions: toggle(l1) toggle(l2)

5. -up(l1) -up(l2)  open
Actions:

Verify plan?  y
plan verified.

yes
```

```
% File: soup (see Chapter 7)

:- declare_types
        type(side,[left,right]),
        type(action,[raise(side),lower(side)]),
        type(fluent,[up(side),spilled]),
        type(time,[0..5]),
        type(atom,[o(action,time),h(fluent,time)]).

:- declare_variables
        var([S,S1],side),
        var(A,action),
        var(F,fluent),
        var(T,time).

h(up(S),T+1) <- o(raise(S),T).
-h(up(S),T+1) <- o(lower(S),T).
h(spilled,T) <- h(up(S),T), -h(up(S1),T) where S \== S1.

o(A,T) <- o(A,T).
-o(A,T) <- -o(A,T).
h(F,0) <- h(F,0).
-h(F,0) <- -h(F,0).

h(F,T+1) <- h(F,T), h(F,T+1).
-h(F,T+1) <- -h(F,T), -h(F,T+1).


| ?- load_file(soup).
% loading file /v/hank/v18/mccain/d/pl/soup
% 44 atoms, 118 rules, 69 clauses loaded.
yes
| ?- query.
enter facts (then ctrl-d)
|: -h(up(S),0).
|: o(raise(left),0).
|: -o(raise(right),0).
|:
enter query
|: h(spilled,1).

yes
```

```
| ?- query.
enter facts (then ctrl-d)
|: -h(spilled,0).
|: o(raise(S),0).
|:
enter query
|: -h(spilled,1).

yes
```

```
% File: domino (see Chapter 7)

:- declare_types
        type(domino,[1,2,3,4,5]),
        type(action,[tip(domino)]),
        type(fluent,[up(domino)]),
        type(time,[0..5]),
        type(atom,[o(action,time),h(fluent,time)]).

:- declare_variables
        var([D,D1],domino),
        var(A,action),
        var(T,time),
        var(F,fluent).

-h(up(D),T+1) <- o(tip(D),T).
-h(up(D1),T+2) <- h(up(D),T), -h(up(D),T+1) where D1 is D+1.

o(A,T) <- o(A,T).
-o(A,T) <- -o(A,T).
h(F,0) <- h(F,0).
-h(F,0) <- -h(F,0).

h(F,T+1) <- h(F,T), h(F,T+1).
-h(F,T+1) <- -h(F,T), -h(F,T+1).
```

```
| ?- load_file(domino).
% loading file /v/hank/v18/mccain/d/pl/domino
% 62 atoms, 163 rules, 109 clauses loaded.
```

```
yes
| ?- query.
enter facts (then ctrl-d)
|: h(up(D),0).
|: o(tip(1),0).
|:
enter query
|: -h(up(1),5) & -h(up(2),5) &
   -h(up(3),5) & -h(up(4),5) & -h(up(5),5).

yes
| ?- query.
enter facts (then ctrl-d)
|: h(up(1),0).
|: -o(tip(1),T).
|:
enter query
|: h(up(1),5).

yes
```

```
% File: stuffy (derived from [Ginsberg and Smith,1988])

:- declare_types
        type(location,[d1,d2,floor]),
        type(duct,[d1,d2]),
        type(object,[o1,o2]),
        type(action,[move(object,location)]),
        type(fluent,[on(object,location),blocked(duct),stuffy]),
        type(time,[0..3]),
        type(atom,[o(action,time),h(fluent,time)]).

:- declare_variables
        var([L,L1],location),
        var(D,duct),
        var([O,O1],object),
        var([A,A1],action),
        var(T,time),
        var(F,fluent),
        var(G,inertial_fluent).
```

```
:- display_literals(positive).

h(on(O,L),T+1) <- o(move(O,L),T).
h(blocked(D),T) <- h(on(O,D),T).
-h(blocked(D),T) <- -h(on(o1,D),T), -h(on(o2,D),T).
h(stuffy,T) <- h(blocked(d1),T),  h(blocked(d2),T).
-h(stuffy,T) <- -h(blocked(D),T).

-h(on(O,L),T) <- h(on(O,L1),T) where L\== L1.

false <- h(on(O,D),T), h(on(O1,D),T) where O @< O1.
false <- o(A,T), o(A1,T) where A @< A1.

o(A,T) <- o(A,T).
-o(A,T) <- -o(A,T).
h(F,0) <- h(F,0).
-h(F,0) <- -h(F,0).

h(on(O,L),T+1) <- h(on(O,L),T), h(on(O,L),T+1).
```

```
| ?- load_file(stuffy).
% loading file /v/hank/v18/mccain/d/pl/stuffy
% 62 atoms, 256 rules, 235 clauses loaded.
yes
| ?- query.
enter facts (then ctrl-d)
|: h(on(o1,floor),0).
|: -o(move(o1,D),T).
|:
enter query
|: -h(stuffy,3).

yes
| ?- plan.
enter facts (then ctrl-d)
|: h(on(o1,d1),0).
|: h(on(o2,d2),0).
|:
enter goal
|: h(on(o1,d2),T) & h(stuffy,T).
```

```
0.  on(o1,d1)  on(o2,d2)  blocked(d1)  blocked(d2)  stuffy
Action(s): move(o2,floor)

1.  on(o1,d1)  on(o2,floor)  blocked(d1)
Action(s): move(o1,d2)

2.  on(o1,d2)  on(o2,floor)  blocked(d2)
Action(s): move(o2,d1)

3.  on(o2,d1)  on(o1,d2)  blocked(d1)  blocked(d2)  stuffy
Action(s):

Verify plan?  y
plan verified.

yes


% File: airport (derived from [McCarthy,1959])

:- declare_types
        type(location,[desk,garage,airport]),
        type(entity,[i,car]),
        type(action,[walk(location,location),
                     drive(location,location)]),
        type(fluent,[at(entity,location)]),
        type(time,[0..4]),
        type(atom,[walkable(location,location),
                   drivable(location,location),
                   o(action,time),h(fluent,time)]).

:- declare_variables
        var([A,A1],action),
        var([F],fluent),
        var([T,T1],time),
        var([E],entity),
        var([L,L1],location).

:- display_literals(positive).

walkable(desk,garage) <- true.
walkable(garage,desk) <- true.
-walkable(L,L1) <- -walkable(L,L1).
```

```
drivable(garage,airport) <- true.
drivable(airport,garage) <- true.
-drivable(L,L1) <- -drivable(L,L1).

h(at(i,L1),T+1) <- o(walk(L,L1),T).

false <- o(walk(L,L1),T), -h(at(i,L),T).
false <- o(walk(L,L1),T), -walkable(L,L1).

h(at(i,L1),T+1) <- o(drive(L,L1),T).
h(at(car,L1),T+1) <- o(drive(L,L1),T).

false <- o(drive(L,L1),T), -h(at(i,L),T).
false <- o(drive(L,L1),T), -h(at(car,L),T).
false <- o(drive(L,L1),T), -drivable(L,L1).

-h(at(E,L),T) <- h(at(E,L1),T) where L \== L1.

o(A,T) <- o(A,T).
-o(A,T) <- -o(A,T).
h(F,0) <- h(F,0).
-h(F,0) <- -h(F,0).

h(F,T+1) <- h(F,T), h(F,T+1).
-h(F,T+1) <- -h(F,T), -h(F,T+1).
```

```
| ?- load_file(airport).
% loading file /v/hank/v18/mccain/d/pl/airport
% 140 atoms, 657 rules, 461 clauses loaded.
yes
| ?- query.
enter facts (then ctrl-d)
|: h(at(i,desk),0).
|: h(at(car,airport),0).
|:
enter query
|: -h(at(i,airport),4).

yes
| ?- plan.
```

```
enter facts (then ctrl-d)
|: h(at(i,desk),0).
|: h(at(car,garage),0).
|:
enter goal
|: h(at(i,airport),T).

0.  at(i,desk)  at(car,garage)
Actions: walk(desk,garage)

1.  at(i,garage)  at(car,garage)
Actions: drive(garage,airport)

2.  at(i,airport)  at(car,airport)
Actions: drive(airport,garage)

3.  at(i,garage)  at(car,garage)
Actions: drive(garage,airport)

4.  at(i,airport)  at(car,airport)
Actions:

Verify plan?  y
plan verified.

yes
| ?- plan.
enter facts (then ctrl-d)
|: h(at(i,desk),0).
|: h(at(car,garage),0).
|:
enter goal
|: h(at(i,airport),T) & h(at(i,desk),T1) where T1 > T.

0.  at(i,desk)  at(car,garage)
Actions: walk(desk,garage)

1.  at(i,garage)  at(car,garage)
Actions: drive(garage,airport)

2.  at(i,airport)  at(car,airport)
Actions: drive(airport,garage)
```

```
3.  at(i,garage)  at(car,garage)
Actions: walk(garage,desk)

4.  at(i,desk)  at(car,garage)
Actions:

Verify plan?  y
plan verified.

yes
```

File: domino-events (see Chapter 7)

```
:- declare_types
        type(domino,[1,2,3,4,5]),
        type(action,[tip(domino)]),
        type(fluent,[up(domino)]),
        type(time,[0..5]),
        type(event,[fall(domino)]),
        type(atom,[o(action,time),o(event,time),h(fluent,time)]).

:- declare_variables
        var([D,D1],domino),
        var([A,A1],action),
        var(E,event),
        var(T,time),
        var(F,fluent).

o(fall(D),T) <- o(tip(D),T).
-h(up(D),T+1) <- o(fall(D),T).
o(fall(D1),T+1) <- o(fall(D),T), h(up(D1),T+1) where D1 is D+1.

false <- o(fall(D),T), -h(up(D),T).
false <- o(A,T), o(A1,T) where A \== A1.

-o(E,T) <- -o(E,T).

o(A,T) <- o(A,T).
-o(A,T) <- -o(A,T).
h(F,0) <- h(F,0).
```

```
-h(F,0) <- -h(F,0).

h(F,T+1) <- h(F,T), h(F,T+1).
-h(F,T+1) <- -h(F,T), -h(F,T+1).
```

```
| ?- load_file('domino-events').
% loading file /v/hank/v18/mccain/d/pl/domino-events
% 92 atoms, 377 rules, 327 clauses loaded.
yes
| ?- query.
enter facts (then ctrl-d)
|: h(up(D),0).
|: o(tip(1),0).
|:
enter query
|: -h(up(1),5) & -h(up(2),5) &
   -h(up(3),5) & -h(up(4),5) & -h(up(5),5).

yes
| ?- query.
enter facts (then ctrl-d)
|: h(up(1),0).
|: -o(tip(1),T).
|:
enter query
|: h(up(1),5).

yes
| ?- plan.
enter facts (then ctrl-d)
|: h(up(D),0).
|:
enter goal
|: -h(up(1),T) & -h(up(2),T) &
   -h(up(3),T) & -h(up(4),T) & -h(up(5),T).

0.  up(1)  up(2)  up(3)  up(4)  up(5)
Events: fall(2)
Actions: tip(2)

1.  up(1) -up(2)  up(3)  up(4)  up(5)
```

```
Events: fall(1) fall(3)
Actions: tip(1)

2. -up(1) -up(2) -up(3)  up(4)  up(5)
Events: fall(4) fall(5)
Actions: tip(5)

3. -up(1) -up(2) -up(3) -up(4) -up(5)
Actions:

4. -up(1) -up(2) -up(3) -up(4) -up(5)
Actions:

5. -up(1) -up(2) -up(3) -up(4) -up(5)
Actions:

Verify plan?  y
plan verified.

yes
```

Chapter 10

Conclusion

We have investigated the role of causal knowledge in commonsense reasoning about action and change. In this section, we summarize our main contributions and list a number of topics for future work.

10.1 Summary of Contributions

The main contributions are the following.

- We have defined two formalisms in which "causal laws" representing the conditions under which facts are caused can be expressed—the action description language $\mathcal{A}_{\text{CL}}$, and the language of causal theories and its specialization $\mathcal{L}_{\text{CL}}$. In Appendix A, we also define a modal formalism, called CEL, which generalizes the language of causal theories and indirectly gives truth conditions for causal laws.

- We have studied the relationship between the language of causal theories and default logic, and the relationship between objective programs (the logic program subclass of causal theories) and various semantics for logic programming.

- We have defined a translation from objective programs into classical propositional logic via a generalization of the completion procedure [Clark, 1978] for normal logic programs.

- We have described and illustrated a general approach to formalizing action domains as causal theories. We have shown how to formalize inertial, momentary, and exogenous fluents, and we have shown how to express ramification and qualification constraints, explicit definitions, concurrency, nondeterministic actions, actions with delayed effects, and dynamic domains in which things change by themselves.

- We have defined two action query languages, including one that contains operators for the natural modalities of historical necessity and possibility.

- On the basis of the above-mentioned generalization of Clark's completion procedure, we have described and implemented an approach to automated query answering and planning which is based on satisfiability checking [Kautz and Selman, 1992].

- Finally, we have attempted to illuminate several conceptual issues that arise in formalizing action domains: (i) the inappropriateness of using state constraints to infer the indirect effects of actions, (ii) the meaning of inertia, (iii) the issue of language dependence in action formalisms, and (iv) the significance of causal laws with non-literal consequents and their role in specifying nondeterministic actions.

10.2 Topics for Future Work

The following is a list of items for future work that would improve upon the results that we have described.

- The approach to query answering and planning described in Chapter 9 is limited to the class of objective programs. This limitation—specifically, the restriction to causal laws with literal consequents—is, as we have remarked, a serious limitation which must be overcome. We discussed one potential remedy in Section 9.3.

- In our version of satisfiability planning, we find complete action histories in which the goal is achieved. As discussed in Section 9.2, this leaves open the problem of finding a part of an action history that is truly essential to achieving the goal.

- When time is infinite, the literal completion of an objective program is also infinite. It would be useful to know syntactic conditions that would guarantee that specific classes of consequences (in the reduced language) are unaffected when time is changed from infinite to finite, or from a larger finite size to a smaller one.

- Satisfiability planning methods are not effective when there are nondeterministic actions or when there is incomplete knowledge of the initial state. It is important to investigate other planning methods for these cases.

In addition to the above topics, there are a variety of other issues of interest related to formalizing action domains about which we have said little or nothing. We will mention only a few of these.

- In the language $\mathcal{L}_{\rm CL}$, we describe when an action begins to occur and when its effects appear, but we do not describe the duration of the action itself. It would be interesting to investigate the usefulness of taking the atoms of causal theories to be expressions which reference two time points, rather than one. For example, we might write $Load_{t,t+5}$ to say that the action of loading the gun occurs in the interval from time t to $t+5$. Such expressions would be similar to the "temporal propositions" (e.g., $True(t, t+5, Load)$) that Shoham [1987,1990] has proposed as the participants of causal relations.

- The correspondence between the causally possible world histories and the causally explained interpretations of a causal theory rests on the assumption that the causal theory is complete with respect to the conditions under which facts are caused. Thus, no allowance is made in the semantics of causal theories for ignorance or uncertainty. It is not easy to see how the demand for completeness might be lessened while preserving, for example, our solution to the frame problem. Perhaps uncertainty could be represented by a set of complete causal theories. If so, how should such a set be represented?[1]

- We have not attempted to model the knowledge state of the agent. Nor have we addressed the issue of knowledge producing actions. To this end, one would hope to adapt the methods of [Moore, 1985a] and [Scherl and Levesque, 1993].

10.3 A Final Word

In each of the formalisms that we have defined, we have been guided by three concerns: conceptual plausibility, mathematical simplicity, and expressiveness. In an attempt to satisfy the first of these concerns, we have endeavored to explain the ideas behind each of our semantic definitions. It is primarily by means of such explanations, we believe, that mathematical formalisms gain content, primarily in relation to them that semantic definitions can be judged right or wrong, and primarily through the construction and criticism of such explanations that progess will

[1]In some cases, it is possible to model incomplete knowledge of the conditions under which facts are caused by nondeterminism. For instance, our various formalizations of coin tossing can be viewed in this light—not as models of nondeterminism but of ignorance. However, this strategy is not always possible. For instance, suppose that we know that a coin either has two heads or two tails, but we do not know which. In this case, modeling coin tossing as nondeterministic would exaggerate the extent of our ignorance. It would admit causally possible world histories that we know are not possible, namely, those in which the coin is tossed more than once and comes up sometimes heads and sometimes tails.

be made.

What is perhaps most responsible for the degree to which we have achieved the goals of mathematical simplicity and expressiveness is our decision not to introduce "foundationalist" assumptions into the semantics of our formalisms; for example, the assumption that every fact in a causally possible world history must be "causally grounded" in facts about the initial state of the world (if any), facts about the actions performed, and facts preserved by inertia. In each of our formalisms, we have required the facts in any causally possible world history (or possible next state) to be exactly those that are caused according to our theory. But we have not required the facts to be causally grounded in any smaller foundation than the set of all facts itself. (By contrast, the causal framework of [McCain and Turner, 1995]—described in Appendix A—presupposes foundationalism, requiring the facts in any possible next state to be grounded in inertia and the explicit effect.) Because we do not presuppose foundationalism, our formalisms are mathematically simpler than they otherwise would have been, and in important ways they are more expressive. Specifically, in each of our formalisms it is possible to say that a fact is a condition for itself being caused. As we have seen, this expressive possibility is the key to representing—in the language of causal theories—inertia, exogenous facts, momentary fluents, and nondeterminism.

Appendix A

Related Formalisms

In this appendix, we discuss two additional formalisms that are related to those defined in this dissertation. We compare the causal framework defined in Chapter 3 with the earlier framework of [McCain and Turner, 1995] in which static causal laws are represented by inference rules, and we present a nonmonotonic modal formalism that generalizes the language of causal theories.

A.1 Causal Laws and Inference Rules

In Chapter 3, we defined $Res_D(E, S)$, the set of states that can result after performing an action with the explicit effect E in the state S, given background knowledge in the form of a set D of static causal laws. The definition given there differs from an earlier definition given in [McCain and Turner, 1995]. In this section, we present the earlier definition and compare it with the new one.

A.1.1 An Earlier Causal Framework

Let X be a set of formulas of propositional logic and D be a set of static causal laws. We say that X is *closed under* D, if for every static causal law $\phi \Rightarrow \psi$ in D, if $\phi \in X$ then $\psi \in X$.

The derivability relation $\vdash$ in propositional logic is easily extended to take account of static causal laws. Given a set Γ of formulas, a set D of static causal laws, and a formula ϕ, we write

$$\Gamma \vdash_D \phi$$

to mean that ϕ is an element of the smallest set of formulas that contains Γ and is closed with respect to propositional logic and closed under D.

As in the case of Definition 4 of Chapter 3, the states are the interpretations that satisfy, for each static causal law $\phi \Rightarrow \psi$ in D, the corresponding material conditional $\phi \supset \psi$.

Definition 5 [McCain and Turner, 1995] For any set D of static causal laws, explicit effect E, and state S, $Res^5_D(E, S)$ is the set of states S' such that

$$S' = \{L : (S \cap S') \cup E \vdash_D L\}.$$

For each of the examples considered in Chapter 3, with the exception of Example 3.10, Definitions 4 and 5 yield identical results. As an illustration, consider again Example 3.6.

Example A.1

$$\begin{aligned} S &= \{\neg \mathit{Up1}, \mathit{Up2}, \neg \mathit{On}\} \\ E &= \{\mathit{Up1}\} \\ D &= \{(\mathit{Up1} \equiv \mathit{Up2}) \Rightarrow \mathit{On},\ \ \neg(\mathit{Up1} \equiv \mathit{Up2}) \Rightarrow \neg \mathit{On}\}. \end{aligned}$$

Previously, we observed that $Res^4_D(E, S)$ contained the single possible next state $S' = \{\mathit{Up1}, \mathit{Up2}, \mathit{On}\}$. Now, we find that $Res^5_D(E, S)$ is the same. $\Diamond$

Both Definitions 4 and 5 require exactly the literals that are true in any possible next state to be caused. However, they differ on how what is caused is to be determined. Consider, for example, the fluent *On*, which holds in S'. According to both definitions, *On*'s becoming true in S' is a ramification induced by the first of the two causal laws in D. However, the grounds for inferring that *On* is caused differ in the two cases. In the case of Definition 4, the ground is that $Up1 \equiv Up2$ is true in S'. In the case of Definition 5, the ground is that $Up1 \equiv Up2$ is derivable from $(S \cap S') \cup E$. Definition 5 is based on the foundationalist assumption that all literals in any possible next state are "causally grounded" in the subset of facts preserved by inertia and the explicit effect. Definition 4 makes no such assumption.

Definitions 4 and 5 agree in the preceding example and in many others, but they do not always agree, as the following examples show.

Example A.2 In this example, we illustrate the behaviors of the two definitions with respect to "reflexive" causal laws of the form $\phi \Rightarrow \phi$. Let

$$\begin{aligned} S &= \{\neg p, \neg q\} \\ E &= \{p\} \\ D &= \emptyset. \end{aligned}$$

According to both Definitions 4 and 5, the unique possible next state is $\{p, \neg q\}$. However, if we change the example by adding to D the static causal law $q \Rightarrow q$, the two definitions yield different results. Let

$$\begin{aligned} S &= \{\neg p, \neg q\} \\ E &= \{p\} \\ D' &= \{q \Rightarrow q\}. \end{aligned}$$

According to Definition 5, there is still the same unique possible next state,

$$Res^5_D(E, S) = \{\{p, \neg q\}\}.$$

However, according to Definition 4 there is now a second possibility,

$$Res^4_{D'}(E, S) = \{\{p, \neg q\}, \{p, q\}\}.$$

$\diamond$

Example A.3 The following example, which is due to Hudson Turner (personal communication), illustrates the fact that Definition 4 possesses a certain disjunction property that Definition 5 lacks.

$$S = \{\neg p, \neg q\}$$
$$E = \{q\}$$
$$D = \{\mathit{True} \Rightarrow (p \equiv q)\}.$$

According to both Definitions 4 and 5, the unique possible next state is $\{p, q\}$. However, if we modify the example, replacing D by D' below, we find different results. Let

$$S = \{\neg p, \neg q\}$$
$$E = \{q\}$$
$$D' = \{p \Rightarrow (p \equiv q), \neg p \Rightarrow (p \equiv q)\}.$$

According to Definition 4, there is the same unique possible next state,

$$\mathit{Res}^4_{D'}(E, S) = \{\{p, q\}\}.$$

So $\mathit{Res}^4_D(E,S) = \mathit{Res}^4_{D'}(E,S)$. According to definition 5, on the other hand, there is no possible next state,

$$\mathit{Res}^5_D(E, S) = \emptyset.$$

So $\mathit{Res}^5_D(E,S) \neq \mathit{Res}^5_{D'}(E,S)$. $\Diamond$

Intuitively, the behavior of Definition 4 in the preceding examples is compatible with reading $\phi \Rightarrow \psi$ as: *in every state in which ϕ is true, ψ is caused to be true*, or *ϕ's being true (in a state) causes ψ to be true (in the same state)*. The behavior of Definition 5, on the other hand, is intuitively incompatible with these readings. Instead, Definition 5 supports the reading: *necessarily, if ϕ is caused to be true (in a state) then ψ is caused to be true (in the same state).* In order to test these intuitions, the reader is encouraged to apply the different readings to the static causal laws in Examples A.2 and A.3 and to ask himself in each case what intuitively follows about the possible next states.

A.1.2 Formal Connections

In this section, we investigate the relationship between Definitions 4 and 5 more precisely.

The following proposition and corollary show that, according to Definition 5, the possible next states always differ minimally from the initial state among the states that satisfy the explicit effect. This property is not possessed by Definition 4, as is shown in Example A.2.

Proposition A.1 [McCain and Turner, 1995]. *Let D be a set of inference rules, and $B = \{\phi \supset \psi : \phi \Rightarrow \psi \in D\}$. For every state S and explicit effect E, $Res^5_D(E,S) \subseteq Res^3_B(E,S)$.*

Corollary A.1 *Let D and B be as in Proposition A.1. For every state S and explicit effect E, the states in $Res^5_D(E,S)$ differ minimally from S (as defined by set inclusion) among the states that satisfy E.*

Proof. By the Propositions A.1 and 3.1, we know that $Res^5_D(E,S) \subseteq Res^W_B(E,S)$. Since, by definition, every state in $Res^W_B(E,S)$ differs minimally from S among the states that satisfy E, the same is true of $Res^5_D(E,S)$. $\square$

The following proposition shows that the set of possible next states according to Definition 5 is always a subset of the set of possible next states according to Definition 4.

Proposition A.2 *For any state S, explict effect E, and set D of static causal laws, $Res^5_D(E,S) \subseteq Res^4_D(E,S)$.*

Proof. Suppose $S' \in Res^5_D(E,S)$. Then

$$S' = \{L : (S \cap S') \cup E \vdash_D L\}. \tag{A.1}$$

It follows that

$$D^{S'} = \{\psi : \text{ for some } \phi,\ \phi \Rightarrow \psi \in D \text{ and } (S \cap S') \cup E \vdash_D \phi\}.$$

So

$$S' = \{L : (S \cap S') \cup E \cup D^{S'} \models L\}.$$

Therefore, $S' \in Res^4_D(E,S)$. $\Box$

The containment relation described in Proposition A.2 does not, in general, hold in the opposite direction. It does so, however, in special cases. This is shown by the following proposition, where we direct our attention to a set of static causal laws only if (i) it belongs to the class of objective programs (so all of its laws have the form $B \Rightarrow L$, where B is a conjunction of literals and L is a literal), and (ii) its atom dependency graph contains no infinite paths. (The atom dependency graph of an objective program is defined in Section 8.2.2.) Notice that the explicit effect is required to be a set of literals.

Proposition A.3 *Let D be a set of static causal laws that belongs to the class of objective programs. If the atom dependency graph for D contains no infinite paths, then for every state S and set E of literals, $Res^4_D(E,S) = Res^5_D(E,S)$.*

Proof. By Proposition A.2, we know that $Res^5_D(E,S) \subseteq Res^4_D(E,S)$. To show inclusion in the opposite direction, let us suppose that $S' \in Res^4_D(E,S)$. Then

$$S' = \{L : (S \cap S') \cup E \cup D^{S'} \models L\}. \tag{A.2}$$

We wish to show that $S' \in Res^5_D(E,S)$, i.e., that

$$S' = \{L : (S \cap S') \cup E \vdash_D L\}.$$

By (A.2), we know that

$$\{L : (S \cap S') \cup E \vdash_D L\} \subseteq S'.$$

To prove that

$$S' \subseteq \{L : (S \cap S') \cup E \vdash_D L\}$$

we proceed by contradiction. Suppose there exists a literal $L_1 \in S'$ such that $(S \cap S') \cup E \not\vdash_D L_1$. By (A.2), we know that $(S \cap S') \cup E \cup D^{S'} \models L_1$. It follows that $L_1 \in D^{S'}$. Thus, for some conjunction of literals B_1 there exists a static causal law $B_1 \Rightarrow L_1 \in D$ such that $S' \models B_1$ and $(S \cap S') \cup E \not\vdash_D B_1$. It follows that for some literal conjunct L_2 in B_1, $L_2 \in S'$ but $(S \cap S') \cup E \not\vdash_D L_2$. The preceding argument can be repeated indefinitely to show the existence of similar literals L_3, L_4, and so on. This contradicts our assumption that there are no infinite paths in the atom dependency graph of D. We conclude that there is no literal $L \in S'$ such that $(S \cap S') \cup E \not\vdash_D L$. Thus, $S' \subseteq \{L : (S \cap S') \cup E \vdash_D L\}$. $\square$

In Example 3.10, we observed that the presence of loops in a set of static causal laws (for example, as occurs in $\neg Up(A) \Rightarrow Up(B)$ and $Up(B) \Rightarrow \neg Up(A)$) could lead, by Definition 4, to the possibility of spontaneous change. The following proposition shows that spontaneous change is impossible under the conditions described in Proposition A.3.

Proposition A.4 *Let D be a set of static causal laws that belongs to the class of objective programs. If the atom dependency graph for D contains no infinite paths, then for every state S, $Res^4_D(\emptyset, S) = \{S\}$.*

Proof. Let S be a state. It is clear that $S = \{L : (S \cap S) \cup \emptyset \cup D^S \models L\}$. So we know that $S \in Res^4_D(\emptyset, S)$. By Proposition A.3 and Corollary A.1, every state in $Res^4_D(\emptyset, S)$ differs minimally from S among the states that satisfy $\emptyset$. It follows that $Res^4_D(\emptyset, S)$ contains no state other than S. $\square$

A.2 A Modal Generalization of Causal Theories

In this section, we define a system of modal logic called CEL (for Causal Explanation Logic) and an embedding of causal theories into CEL.

In defining the semantics of causal theories in Chapter 5, we defined the notion of a causally explained interpretation, but we did not specify truth conditions for causal laws. This deficiency is remedied by the above-mentioned embedding. We will see that the truth conditions for causal laws given by the embedding conform to the weaker of the two readings for causal laws that we have used throughout this dissertation. The ideas in this chapter are the product of joint work with Hudson Turner. CEL is related to the work of Geffner [1990, 1992].

A.2.1 CEL: Causal Explanation Logic

CEL is obtained by augmenting S5 modal logic by a definition of the causally explained interpretations. We begin by briefly reviewing the syntax and semantics of S5 modal logic.

A propositional modal language is given by a set of atoms. The formulas of the language are inductively defined as follows:

- an atom is a formula,
- if ϕ and ψ are formulas, then $\phi \wedge \psi$ is a formula, and
- if ϕ is a formula, then both $\neg\phi$ and $\mathsf{C}\phi$ are formulas.

Here C is the modal necessity operator. The other standard propositional connectives ($\vee$, $\supset$, and $\equiv$) are introduced by abbreviations in the usual way.

An S5 structure is a pair (I, S), where I is an interpretation (of the set of atoms) and S is a set of interpretations such that $I \in S$. We continue to identify an interpretation I with the set of literals L such that $I \models L$. Truth in a structure is defined as follows. (Here ϕ and ψ are arbitrary formulas.)

$$
\begin{aligned}
&(I,S) \models A \ \text{ iff } \ A \in I, \quad \text{ if } A \text{ is an atom,}\\
&(I,S) \models \neg\phi \ \text{ iff } \ (I,S) \not\models \phi,\\
&(I,S) \models \phi \wedge \psi \ \text{ iff } \ (I,S) \models \phi \text{ and } (I,S) \models \psi,\\
&(I,S) \models \mathsf{C}\phi \ \text{ iff } \text{ for all } I' \in S,\ (I',S) \models \phi
\end{aligned}
$$

Given an S5 theory T, we write $(I,S) \models T$ to mean that $(I,S) \models \phi$, for every $\phi \in T$. In this case, we say that (I,S) is a *model* of T. We also say that (I,S) is an *I-model* of T, emphasizing the distinguished interpretation I.

The following definition was first formulated by Hudson Turner. An interpretation I is *causally explained* according to a CEL theory T if for every set S of interpretations

$$(I,S) \models T \text{ iff } S = \{I\}.$$

This means that $(I,\{I\})$ is the unique I-model of T.

An alternative characterization of the causally explained interpretations is given by the following proposition.

Proposition A.5 *An interpretation I is causally explained according to a CEL theory T if and only if*

$$I = \{L : T \cup I \models \mathsf{C}L\}.$$

Proof. For the left-to-right direction, suppose that for every set S of interpretations

$$(I,S) \models T \text{ iff } S = \{I\}. \tag{A.3}$$

Every model of $T \cup I$ has the form (I,S'), where S' is a set of interpretations that contains I. By (A.3), there is only one such model, $(I,\{I\})$. It follows that $L \in I$

if and only if $T \cup I \models \mathsf{C}L$. Therefore, $I = \{L : T \cup I \models \mathsf{C}L\}$. For the right-to-left direction, suppose that

$$I = \{L : T \cup I \models \mathsf{C}L\}. \tag{A.4}$$

In order to show that I is causally explained according to T, let S be a set of interpretations, and suppose $(I, S) \models T$. We will show that $S = \{I\}$. It follows that $(I, S) \models T \cup I$. By (A.4), $T \cup I \models \mathsf{C}L$, for every $L \in I$. Therefore, we know that for all $L \in I$, $(I, S) \models \mathsf{C}L$. So $S = \{I\}$. Now for the opposite direction, suppose that $S = \{I\}$. We will show that $(I, S) \models T$. Suppose instead that $(I, \{I\}) \not\models T$. Then $(I, \{I\}) \not\models T \cup I$. By (A.4), we know that $T \cup I$ has a model. So it must have the form (I, S'), for some S' other than $\{I\}$. Thus, there is a literal $L \in I$ such that $T \cup I \not\models \mathsf{C}L$, which contradicts (A.4). We conclude that $(I, S) \models T$. $\square$

A.2.2 A Modal Encoding for Causal Laws

A causal law can be encoded in CEL by the formula

$$\phi \supset \mathsf{C}\psi\,.$$

Intuitively, (A.5) says that: *if ϕ then the fact that ψ is caused*. This reading does not capture the necessity of causal laws. However, if we use CEL only to express propositions that are true in all possible worlds (that is, if we use CEL as an action description language, not also as an action query language), then we can safely read (A.5) instead as: *necessarily, if ϕ then the fact that ψ is caused.* This is the weaker of our two readings for $\phi \Rightarrow \psi$.

A.2.3 CEL and Causal Theories

We have described when an interpretation I is causally explained in two different frameworks. In the language of causal theories, an interpretation I is causally explained according to a causal theory D if I is the unique model of D^I. Here "model" is understood in the sense of propositional logic. In CEL, an interpretation I is causally explained according to a CEL theory T if $(I, \{I\})$ is the unique I-model of T. Here "model" is understood instead in the sense of S5 modal logic. In this section, we show, under the modal encoding defined in Section A.2.2, that the two definitions agree.

Let D be a causal theory. We define

$$T(D) = \{\phi \supset \mathsf{C}\psi : \phi \Rightarrow \psi \in D\}.$$

Geffner [1990, 1992] used the same representation of causal knowledge, but in the modal logic T rather than $S5$.

Proposition A.6 *An interpretation I is causally explained according to a causal theory D if and only if I is causally explained according to $T(D)$.*

The proof of this proposition is given below, using the following lemmas.

Lemma A.1 *For every causal theory D and interpretation I, $I \models D^I$ if and only if $(I, \{I\}) \models T(D)$.*

Proof. For the left-to-right direction, suppose $(I, \{I\}) \not\models T(D)$. Then there is a formula $\phi \supset \mathsf{C}\psi$ in $T(D)$ such that $(I, \{I\}) \not\models \phi \supset \mathsf{C}\psi$. It follows that $I \models \phi$ but $I \not\models \psi$. We know that $\phi \Rightarrow \psi$ is in D. Since $I \models \phi$, ψ is in D^I. Since $I \not\models \psi$, $I \not\models D^I$. For the right-to-left direction, suppose $I \not\models D^I$. Then there is a causal law $\phi \Rightarrow \psi$ in D such that $I \models \phi$ but $I \not\models \psi$. We know that the formula $\phi \supset \mathsf{C}\psi$ is in $T(D)$. Therefore, $(I, \{I\}) \not\models T(D)$. $\Box$

Lemma A.2 *Let D be a causal theory and I be an interpretation such that $I \models D^I$. There is no interpretation I' such that $I' \neq I$ and $I' \models D^I$ if and only if there is no set S of interpretations such that $S \neq \{I\}$ and $(I, S) \models T(D)$.*

Proof. For the left-to-right direction, suppose there is a set S of interpretations such that $S \neq \{I\}$ and $(I, S) \models T(D)$. Then for some I' in S such that $I' \neq I$ and for all formulas $\phi \supset \mathsf{C}\psi$ in $T(D)$, if $I \models \phi$ then $I' \models \psi$. It follows that $I' \models D^I$. For the right-to-left direction, suppose there is an interpretation I' such that $I' \neq I$ and $I' \models D^I$. By hypothesis, $I \models D^I$. Therefore, $(I, \{I, I'\}) \models T(D)$. So there is a set S of interpretations such that $S \neq \{I\}$ and $(I, S) \models T(D)$. □

Proof (of Proposition A.6). We will show that: I is the unique model of D^I if and only if $(I, \{I\})$ is the unique I-model of $T(D)$. (Let us call this proposition X.) By Lemma A.1, we know that $I \models D^I$ if and only if $(I, \{I\}) \models T(D)$. So if $I \not\models D^I$, we are done (both sides of proposition X are false). So let us assume that $I \models D^I$. By Lemma A.1, we know that $(I, \{I\}) \models T(D)$. Thus, proposition X follows by Lemma A.2. □

It follows by Proposition A.6 that every causal theory can be equivalently represented as a theory in CEL. The same does not hold in reverse, however, since in CEL we can write formulas of other forms than $\phi \supset \mathsf{C}\psi$. This possibility raises the following question: Is the additional expressiveness of CEL useful in formalizing action domains? This is a topic for future research.[1]

[1]Hudson Turner has explored some possible uses for the additional expressiveness of CEL in his recent dissertation [Turner, 1997a].

Appendix B

The Program Listing

```
/* File: satp.pl
   Language: SICStus Prolog (Release 3)
   Date: 12/24/96
   Author: Norman C. McCain

This program uses the satisfiability checker 'ntab' (previously
    known as 'tableau'), of Crawford and Auton (AAAI-93) to
    solve query answering and planning problems (in the
    style of Kautz and Selman (ECAI 92)) with respect to L_{CL}
    objective programs.

The three main user level procedures are:

    load_file(+Filename): reads in an objective program and
        asserts its literal completion in clausal form into
        the Prolog database.

    query:  prompts for facts Gamma and a query Q and using
        the literal completion of the objective program
        (previously loaded by load_file/1) determines
        whether  lcomp(D) \cup Gamma \cup -Q is satisfiable.

    plan:   prompts for facts Gamma about the initial state
        and a goal G and using
        the literal completion of the objective program
        (previously loaded by load_file/1) determines
        whether  lcomp(D) \cup Gamma \cup G  is satisfiable.
```

```
See Chapter 9 for additional details.  */

:- use_module(library(lists)).
:- use_module(library(ordsets)).
:- use_module(library(system)).

:- op(1150,fx,declare_types).
:- op(1150,fx,declare_variables).
:- op(700,xfx,'..').
:- op(1150,xfx,<-).
:- op(1150,fx,<-).
:- op(1170,xfx,where).
:- op(1000,xfy,&).
:- op(1155,xfx,<->).

:- dynamic
        '$foo$'/1, '$num$'/1, (<-)/2, type/2, var/2, atom/2,
        clause/1, display_list/1.


%%% load_file/1 %%%
load_file(File) :-
        write('% loading file '), absolute_file_name(File,A),
        write(A), nl, ttyflush,
        expand_file(File),
        write('% '),
        count_atoms(N), write(N), write(' atoms, '),
        count_rules(R), write(R), write(' rules, '),
        ttyflush, assert_law_clauses,
        count_clauses(M), write(M), write(' clauses loaded.').

expand_file(File) :-
        init, seen, see(File), read_pass, seen.

read_pass :-
        read_fail_loop.
read_pass :-
        find_atom(N,true), assertz((N<-[])),
        find_atom(M,false), C is 0-M, assertz((C<-[])).

read_fail_loop :-
```

```
        repeat,
          read_term(Term,[variable_names(Names)]),
          ( ( Term = (H<-B where Tests) ;
              Term = (H<-B), Tests = true ;
              Term = (<- B where Tests), H = false ;
              Term = (<- B), H = false, Tests = true )
              -> process_rule((H<-B where Tests),Names)
          ; Term = (:- declare_types B)
              -> process_types(B),
                 enumerate_atoms
          ; Term = (:- declare_variables B)
              -> bind_vars_to_names(Names),
                 process_variables(B)
          ; Term = (:- Com)
              -> call(Com)                  % other directives
          ; Term == end_of_file
              -> !, fail
          ; nl, write('unexpected input: '), write(Term), nl ),
        fail.

init :- seen, told,
        retractall(type(_ ,_)),
        retractall(var(_,_)),
        retractall(atom(_,_)),
        retractall((_<-_)),
        retractall(clause(_)),
        retractall('$num$'(_)),
        retractall(display_list(_)).

process_types((A,B)) :-
        !, process_types(A), process_types(B).
process_types(type(time,Cs)) :-
        !,
        ( Cs = [X..Y]
             -> expand_times(X,Y,Es) ; Es = Cs ),
        assertz(type(time,Es)).
process_types(type(Type,Cs)) :-
        expand_list(Cs,Es), assertz(type(Type,Es)).

enumerate_atoms :-
        type(atom,As), member(A,As), next_num(N), assertz(atom(N,A)),
        fail.
```

```
enumerate_atoms :-
        next_num(M), assertz(atom(M, true)),
        next_num(N), assertz(atom(N,false)).

expand_times(S,S,[S]) :-
        !.
expand_times(S,T,[S|Ts]) :-
        S1 is S + 1, expand_times(S1,T,Ts).

expand_list([C|Cs],Fs) :-
        subst_vars_for_type_names(C,D,Vs),
        retractall('$foo$'(_)),
        ( bind_vars_to_terms(Vs),
          assertz('$foo$'(D)),
          fail
        ; findall(D,'$foo$'(D),Ds),
          expand_list(Cs,Es), append(Ds,Es,Fs) ).
expand_list([],[]).

subst_vars_for_type_names(C,D,Vs) :-
        atom(C),
        !,
        ( type(C,Cs) -> Vs = [D/Cs] ; D = C, Vs = [] ).
subst_vars_for_type_names(C,D,Vs) :-
        functor(C,F,N),
        functor(D,F,N),
        subst_vars_for_type_names_arg(C,N,D,Vs).

subst_vars_for_type_names_arg(_C,0,_D,[]) :-
        !.
subst_vars_for_type_names_arg(C,N,D,Vs) :-
        arg(N,C,A),
        subst_vars_for_type_names(A,B,Xs),
        arg(N,D,B),
        M is N-1,
        subst_vars_for_type_names_arg(C,M,D,Ys),
        append(Xs,Ys,Vs).

bind_vars_to_terms([V/Cs|Vs]) :-
        member(V,Cs), bind_vars_to_terms(Vs).
bind_vars_to_terms([]).
```

```
process_variables((A,B)) :-
        !, process_variables(A), process_variables(B).
process_variables(var(V,S)) :-
        ( list(S)
            -> Cs = S
        ; type(S,Cs) ),
        ( list(V)
            -> assert_variables(V,Cs)
        ; assertz(var(V,Cs)) ).

bind_vars_to_names([=(Var,Var)|Names]) :-
        bind_vars_to_names(Names).
bind_vars_to_names([]).

list([_|_]).  list([]).

assert_variables([],_Cs).
assert_variables([V|Vs],Cs) :-
        assertz(var(V,Cs)), assert_variables(Vs,Cs).

process_rule((H<-B where Tests),Names) :-
        listify(B,Bs),
        bind_vars(Names),
        ( call(Tests)
           -> eval_time(H,H1),
              eval_times(Bs,Bs1),
              number_lit(H1,N),
              number_lits(Bs1,Ns),
              assertz((N<-Ns)) ),
        fail.
process_rule(_,_).

bind_vars([=(Name,Var)|Bs]) :-
        ( var(Name,Cs) -> member(Var,Cs) ; true ),
        bind_vars(Bs).
bind_vars([]).

eval_times([L|Ls],[L1|Ls1]) :-
        eval_time(L,L1), eval_times(Ls,Ls1).
eval_times([],[]).
```

```
eval_time(o(A,T),o(A,T1)) :-
        !, T1 is T.
eval_time(-o(A,T),-o(A,T1)) :-
        !, T1 is T.
eval_time(h(A,T),h(A,T1)) :-
        !, T1 is T.
eval_time(-h(A,T),-h(A,T1)) :-
        !, T1 is T.
eval_time(L,L).

assert_law_clauses :-
        atom(N,H), C is 0-N,
        ( H = true
            -> assertz(clause([N])),
               findall(Ms,(N<-Ms),Nss),
               member_of_each(Ns,Nss),
               assert_clause([C|Ns]),
               fail
        ; H = false
            -> assertz(clause([C])),
               findall(Ms,(N<-Ms),Nss),
               member(Ns,Nss),
               negate_lits(Ns,Cs),
               assert_clause([N|Cs]),
               fail
        ; ( findall(Ms,(N<-Ms),Nss),
            ( member(Ns,Nss),
              negate_lits(Ns,Cs),
              assert_clause([N|Cs]),
              fail
            ; member_of_each(Ns,Nss),
              assert_clause([C|Ns]),
              fail )
          ; findall(Ms,(C<-Ms),Nss),
            ( member(Ns,Nss),
              negate_lits(Ns,Cs),
              assert_clause([C|Cs]),
              fail
            ; member_of_each(Ns,Nss),
              assert_clause([N|Ns]),
              fail ) ) ).
assert_law_clauses.
```

```
assert_clauses([C|Cs]) :-
        assert_clause(C), assert_clauses(Cs).
assert_clauses([]).

assert_clause(Ns) :-
        sort(Ns,Cs),
        ( eliminable(Cs) -> true; assertz(clause(Cs)) ).

eliminable(Cs) :-
        find_atom(N,true), ord_member(N,Cs),
        Cs \== [N].
eliminable(Cs) :-
        find_atom(N,false), M is (0-N), ord_member(M,Cs),
        Cs \== [M].
eliminable(Cs) :-
        tautology(Cs).

tautology([N|_Ns]) :-
        N>0, !, fail.
tautology([N|Ns]) :-
        M is (0-N),
        ( ord_member(M,Ns)
            -> true
        ; tautology(Ns)  ).

listify(true,[]) :-
        !.
listify((A,B),[A|Bs]) :-
        !, listify(B,Bs).
listify(A,[A]).

member_of_each([N|Rs],[Ns|Nss]) :-
        member(N,Ns), member_of_each(Rs,Nss).
member_of_each([],[]).

negate_lits([N|Ns],[C|Cs]) :-
        C is 0-N, negate_lits(Ns,Cs).
negate_lits([],[]).

number_lits([L|Ls],[M|Ms]) :-
        number_lit(L,M), number_lits(Ls,Ms).
```

```
number_lits([],[]).

number_lit(-A,N) :-
        !, find_atom(M,A), N is 0-M.
number_lit(A,N) :-
        find_atom(N,A).

find_atom(N,A) :- atom(N,A), !.

next_num(I) :-
        retract('$num$'(N)),
        !, I is N+1,
        assert('$num$'(I)).
next_num(1) :-
        assert('$num$'(1)).

last_num(N) :- '$num$'(N), ! ; N = 0.


%%% query/0 %%%
query :-
        write('enter facts (then ctrl-d)'),
        nl,
        read_facts(Facts),
        clausify_list(Facts,FCs),
        nl,
        write('enter query'),
        nl,
        read_query(Query),
        clausify(-Query,QCs),
        told,
        tell('ntab.in'),
        write_law_clauses,
        write_clauses(FCs),
        write_clauses(QCs),
        told,
        ntab_call(Call),
        system(Call),
        seen,
        see('ntab.out'),
        read_line(Ans),
        ( atom_chars('UNSAT',Ans)
            -> seen
        ; read_line(_),
          read_line(Chars),
          seen,
          get_numbers(Chars,Model),
          display_model(Model), fail ).

ntab_call(Call) :-
        count_atoms(N), M is N+1,
        name(M,Cs),
        append("ntab -p -v",Cs,Half),
        append(Half," <ntab.in >ntab.out",Whole),
        name(Call,Whole).


%%% plan/0 %%%
plan :-
        write('enter facts (then ctrl-d)'),
        nl,
        read_facts(Facts),
        clausify_list(Facts,FCs),
        nl,
        write('enter goal'),
        nl,
        read_query(Goal),
        clausify(Goal,GCs),
        told,
        tell('ntab.in'),
        write_law_clauses,
        write_clauses(FCs),
        write_clauses(GCs),
        told,
        ntab_call(Call),
        system(Call),
        seen,
        see('ntab.out'),
        read_line(Ans),
        ( atom_chars('UNSAT',Ans)
           -> seen, fail
        ; read_line(_),
          read_line(Chars),
          seen,
          get_numbers(Chars,Model),
          display_model(Model),
          nl, nl,
          write('Verify plan?  '),
          get(X), skip_line,
          ( X = 0'y
              -> extract_plan(Model,PCs),
                 clausify(-Goal,QCs),
                 verify_plan(FCs,PCs,QCs)
          ; true ) ).

verify_plan(FCs,PCs,QCs) :-
        told,
        tell('ntab.in'),
        write_law_clauses,
        write_clauses(FCs),
        write_clauses(PCs),
        write_clauses(QCs),
        told,
        ntab_call(Call),
        system(Call),
        seen,
        see('ntab.out'),
        read_line(Ans),
        ( atom_chars('UNSAT',Ans)
            -> seen, write('plan verified.'),
               nl
        ; write('verification failed.'),  nl,
          read_line(_),
          read_line(Chars),
          seen,
          get_numbers(Chars,Model),
          display_model(Model), nl, !, fail ).

extract_plan(Model,PCs) :-
        findall([M],
                ( atom(N,o(_A,_T)),
                  ( member(N,Model) -> M = N ; M is 0-N ) ),
                PCs).

read_facts(Facts) :-
        findall(Fact,(repeat,
                        read_fact(Fact),
                        ( Fact == end_of_file -> !, fail; true)),
                Facts).

read_fact(Fact) :-
        read_term(Term,[variable_names(Names)]),
        ( Term == end_of_file
            -> !, Fact = end_of_file
        ; (Term = (Wff where Tests);
                Term = Wff, Tests = true)
             -> bind_vars(Names),
                call(Tests),
                number_atoms(Wff,Fact) ).

read_query(Fact) :-
        read_term(Term,[variable_names(Names)]),
        ( Term == end_of_file
            -> !, read_query(Fact)
        ; (Term = (Wff where Tests);
                Term = Wff, Tests = true)
             -> bind_vars(Names),
                call(Tests),
                number_atoms(Wff,Fact) ).

number_atoms((A <-> B),(M <-> N)) :-
        !,
        number_atoms(A,M), number_atoms(B,N).
number_atoms((A -> B),(M -> N)) :-
        !,
        number_atoms(A,M), number_atoms(B,N).
number_atoms((A & B),(M & N)) :-
        !,
        number_atoms(A,M), number_atoms(B,N).
number_atoms((A | B),(M | N)) :-
        !,
        number_atoms(A,M), number_atoms(B,N).
number_atoms(-A,-M) :-
        !, number_atoms(A,M).
number_atoms(A,M) :-
        eval_time(A,B), find_atom(M,B).

clausify_list([W|Ws],Cs) :-
```

```
        clausify(W,Cs1),
        clausify_list(Ws,Cs2),
        append(Cs1,Cs2,Cs).
clausify_list([],[]).

clausify(W,Cs) :-
        move_negations(W,W1),
        distribute(W1,W2),
        clause_list(W2,Cs).

move_negations(A,A1) :-
        iff_1(A,A2),
        !,
        move_negations(A2,A1).
move_negations((A & B),(A1 & B1)) :-
        !,
        move_negations(A,A1),
        move_negations(B,B1).
move_negations((A | B),(A1 | B1)) :-
        !,
        move_negations(A,A1),
        move_negations(B,B1).
move_negations(-A,N) :-
        !, N is 0 - A.
move_negations(A,A).

iff_1(- (A <-> B),((A & -B) | (-A & B))).
iff_1(  (A <-> B),((-A | B) & (-B | A))).
iff_1(- (A -> B),(A & -B)).
iff_1(  (A -> B),(-A | B)).
iff_1(- (A & B),(-A | -B)).
iff_1(- (A | B),(-A & -B)).
iff_1(- (-A),A).

distribute((A | B),R) :-
        !,
        distribute(A,A1),
        distribute(B,B1),
        ( A1 = (P & Q)
            -> dist1((P | B1),D1),
               dist1((Q | B1),D2),
               R = (D1 & D2)
        ; B1 = (P & Q)
            -> dist1((A1 | P),D1),
               dist1((A1 | Q),D2),
               R = (D1 & D2)
        ; R = (A1 | B1) ).
distribute((A & B),(A1 & B1)) :-
        !,
        distribute(A,A1),
        distribute(B,B1).
distribute(A,A).

dist1((A | B),R) :-
        ( A = (P & Q)
            -> dist1((P | B),D1),
               dist1((Q | B),D2),
               R = (D1 & D2)
        ; B = (P & Q)
            -> dist1((B | P),D1),
               dist1((B | Q),D2),
               R = (D1 & D2)
        ; R = (A | B) ).


clause_list((A & B),Cs) :-
        !,
        clause_list(A,As),
        clause_list(B,Bs),
        append(As,Bs,Cs).
clause_list(A,[As]) :-
        literal_list(A,As).

literal_list((A | B),Ls) :-
        !,
        literal_list(A,As),
        literal_list(B,Bs),
        append(As,Bs,Ls).
literal_list(A,[A]).

write_law_clauses :-
        clause(Cs), write_clause(Cs), nl, fail.
write_law_clauses.
```

```
write_clauses([C|Cs]) :-
        write_clause(C), nl, write_clauses(Cs).
write_clauses([]).

write_clause([L|Ls]) :-
        write(L), write(' '), write_clause(Ls).
write_clause([]).

report(Model) :-
        seen, see('ntab.out'),
        read_line(Ans),
        ( atom_chars('UNSAT',Ans)
            -> write('UNSAT'), nl, Model = none
        ; write('SAT'),
          read_line(_),
          read_line(Model),
          get_numbers(Model,Ns),
          display_model(Ns), nl  ),
        seen.

read_line([]) :-
        at_end_of_line, skip_line, !.
read_line([C|Cs]) :-
        get0(C), read_line(Cs).

get_numbers(String,Nums) :-
        drop_spaces(String,NewS),
        ( NewS = []
            -> Nums = []
        ; get_number(NewS,Num,Rest),
          Nums = [Num|Nums1],
          get_numbers(Rest,Nums1) ).

drop_spaces([C|Cs],Cs1) :-
        ( C = 32 -> drop_spaces(Cs,Cs1) ; Cs1 = [C|Cs] ).
drop_spaces([],[]).

get_number(String,Num,Rest) :-
        get_until(String,[32,10],Chars,Rest),
        number_chars(Num,Chars).

get_until([Char|Cs],Delimiters,[],Cs) :-
```

```
        member(Char,Delimiters), !.
get_until([C|Cs],Delimiters,[C|Chars],Rest) :-
        get_until(Cs,Delimiters,Chars,Rest).
get_until([],_,[],[]).

display_model(Ns) :-
        display_literals(Ls),
        ( type(action,As) -> true; As = [] ),
        ( type(event,Es) -> true; Es = [] ),
        ( type(fluent,Fs) -> true; Fs = [] ),
        ( type(time,Ts) -> true; Ts = [] ),
        member(T,Ts),
        nl,
        ( nl, write(T), write('. '),
          member(F,Fs),
          find_atom(N,h(F,T)),
          ( member(N,Ns)
              -> (member(F,Ls)
                   -> write(' '), write(F), write(' '), fail )
          ; (member(-F,Ls)
               -> write('-'), write(F), write(' '), fail )
          )
        ; findall(E,(member(E,Es),
                         find_atom(N,o(E,T)),member(N,Ns)),Es1),
          ( Es1 \== []
              -> nl, write('Events: '),
                 ( member(E,Es1), write(E), write(' '), fail ) )
        ; findall(A,(member(A,As),
                         find_atom(N,o(A,T)),member(N,Ns)),As1),
          nl, write('Actions: '),
          ( member(A,As1), write(A), write(' '), fail )
        ).
display_model(_).

count_atoms(N) :- findall(x,atom(_,_),S), length(S,N).
count_rules(N) :- findall(x,(_<-_),S), length(S,N).
count_clauses(N) :- findall(x,clause(_),S), length(S,N).

display_literals(V) :-
        var(V),
        !,
        ( display_list(V)
            -> true
        ; display_literals(all), display_list(V) ).
display_literals(all) :-
        !,
        retractall(display_list(_)),
        type(fluent,Fs),
        complement_list(Fs,Ls),
        append(Fs,Ls,Zs),
        assert(display_list(Zs)).
display_literals(positive) :-
        !,
        retractall(display_list(_)),
        type(fluent,Fs),
        assert(display_list(Fs)).
display_literals(Ls) :-
        retractall(display_list(_)),
        assert(display_list(Ls)).

complement_list([F|Fs],[G|Gs]) :-
        comp(F,G),
        complement_list(Fs,Gs).
complement_list([],[]).

comp(-F,F) :-
        !.
comp(F,-F).
```

Bibliography

[Apt and Bezem, 1990] Krzysztof Apt and Marc Bezem. Acyclic programs. In David Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 617–633, 1990.

[Apt *et al.*, 1988] Krzysztof Apt, Howard Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, San Mateo, CA, 1988.

[Baker, 1989] Andrew Baker. A simple solution to the Yale Shooting Problem. In Ronald Brachman, Hector Levesque, and Raymond Reiter, editors, *Proc. of the First Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 11–20, 1989.

[Baker, 1991] Andrew Baker. Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence*, 49:5–23, 1991.

[Baral and Gelfond, 1993] Chitta Baral and Michael Gelfond. Representing concurrent actions in extended logic programming. In *Proc. of IJCAI-93*, pages 866–871, 1993.

[Baral *et al.*, 1995] Chitta Baral, Michael Gelfond, and Alessandro Provetti. Representing actions I: (Laws, observations and hypotheses). In *Working Notes: AAAI*

Spring Symposium on Extending Theories of Action: Formal Theory and Practical Applications, pages 17–22, 1995.

[Baral, 1995] Chitta Baral. Reasoning about actions: non-deterministic effects, constraints, and qualification. In *Proc. of IJCAI-95*, pages 2017–2023, 1995.

[Boutilier and Friedman, 1995] Craig Boutilier and Nir Friedman. Nondeterministic actions and the frame problem. In *Working Notes: AAAI Spring Symposium on Extending Theories of Action: Formal Theory and Practical Applications*, pages 39–44, 1995.

[Brewka and Hertzberg, 1993] Gerhard Brewka and Joachim Hertzberg. How to do things with worlds: On formalizing actions and plans. *Journal of Logic and Computation*, 3(5), 1993.

[Burks, 1951] Arthur W. Burks. The logic of causal propositions. *Mind*, 60:363–382, 1951.

[Carnap, 1947] Rudolph Carnap. *Meaning and Necessity.* University of Chicago Press, Chicago, Ill., 1947.

[Chellas, 1971] Brian. Chellas. Imperatives. *Theoria*, 37:114–129, 1971.

[Clark, 1978] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Crawford and Auton, 1993] J.M. Crawford and L.D. Auton. Experimental results on the cross-over point in satisfiability problems. In *Proc. AAAI-93*, pages 21–27, 1993.

[Crawford and Etherington, 1992] James Crawford and David Etherington. Formalizing reasoning about change: A qualitative reasoning approach. In *Proc. AAAI-92*, pages 577–583, 1992.

[Davidson, 1980] Donald. Davidson. Agency. In Donald Davidson, editor, *Essays on Actions and Events*, pages 43–62. Oxford University Press, 1980.

[Davis and Putnam, 1960] M Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[Denecker and De Schreye, 1993] Marc Denecker and Danny De Schreye. Representing incomplete knowledge in abductive logic programming. In Dale Miller, editor, *Logic Programming: Proceedings of the 1993 Int'l Symposium*, pages 147–163, 1993.

[Elkan, 1992] Charles Elkan. Reasoning about action in first-order logic. In *Proc. of the 1992 Canadian Conf. on Artificial Intelligence*, 1992.

[Fages, 1994] François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[Faye *et al.*, 1994] J. Faye, U. Scheffler, and Urchs M. *Logic and Causal Reasoning.* Akademie Verlag, Berlin, 1994.

[Fikes and Nilsson, 1971] Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[Geffner, 1989] Hector Geffner. Default reasoning, minimality and coherence. In Ronald Brachman, Hector Levesque, and Raymond Reiter, editors, *Proc. of the First Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 137–148, 1989.

[Geffner, 1990] Hector Geffner. Causal theories of nonmonotonic reasoning. In *Proc. AAAI-90*, pages 524–530, 1990.

[Geffner, 1992] Hector Geffner. *Reasoning with defaults: causal and conditional theories.* MIT Press, Cambridge, MA, 1992.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.

[Gelfond and Lifschitz, 1990] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 579–597, 1990.

[Gelfond and Lifschitz, 1992] Michael Gelfond and Vladimir Lifschitz. Representing actions in extended logic programming. In Krzysztof Apt, editor, *Proc. Joint Int'l Conf. and Symp. on Logic Programming*, pages 559–573, 1992.

[Gelfond and Lifschitz, 1993] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[Gelfond *et al.*, 1991a] Michael Gelfond, Vladimir Lifschitz, Halina Przymusińska, and Mirosław Truszczyński. Disjunctive defaults. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Second Int'l Conf.*, pages 230–237, 1991.

[Gelfond *et al.*, 1991b] Michael Gelfond, Vladimir Lifschitz, and Arkady Rabinov. What are the limitations of the situation calculus? In Robert Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 167–179. Kluwer Academic, Dordrecht, 1991.

[Gelfond, 1987] Michael Gelfond. On stratified autoepistemic theories. In *Proceedings of AAAI-87*, pages 207–211, 1987.

[Gelfond, 1988] Michael Gelfond. Autoepistemic logic and formalization of commonsense reasoning: preliminary report. In *Proc. 2nd Int'l Workshop on Non-Monotonic Reasoning*, pages 176–186, 1988.

[Gibbard and Harper, 1981] Allan Gibbard and William L. Harper. Counterfactuals and two kinds of expected utility. In W.L Harper, R. Stalnaker, and G. Pearce, editors, *Ifs: conditionals, belief, decision, chance, and time*, pages 153–190. D. Reidel Publishing Company, Dordrecht, Holland, 1981.

[Ginsberg and Smith, 1988a] Matthew L. Ginsberg and David E. Smith. Reasoning about action I: a possible worlds approach. *Artificial Intelligence*, 35(2):165–195, 1988.

[Ginsberg and Smith, 1988b] Matthew L. Ginsberg and David E. Smith. Reasoning about action II: the qualification problem. *Artificial Intelligence*, 35(3):311–342, 1988.

[Giunchiglia and Lifschitz, 1995] Enrico Giunchiglia and Vladimir Lifschitz. Dependent fluents. In *Proc. IJCAI-95*, pages 1964–1969, 1995.

[Giunchiglia and Lifschitz, 1997] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal logic. Unpublished manuscript, 1997.

[Giunchiglia *et al.*, 1995] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Actions with indirect effects (extended abstract). In *Working Notes of the Symposium on Extending Theories of Actions*, 1995.

[Goldman, 1970] Alvin I. Goldman. *A theory of human action.* Prentice-Hall, Englewood Cliffs, N.J., 1970.

[Grice, 1989] Paul Grice. *Studies in the way of words.* Harvard University Press, Cambridge, Mass., 1989.

[Haas, 1987] Andrew Haas. The case for domain-specific frame axioms. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence, Proc. of the 1987 Workshop*, 1987.

[Hanks and McDermott, 1986] Steve Hanks and Drew McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proc. AAAI-86*, pages 328–333, 1986.

[Hanks and McDermott, 1987] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[Herre and Wagner, 1996] Heinrich Herre and Gerd Wagner. Stable generated models of generalized logic programs. This was a talk delivered at the Dagstuhl Seminar 9627, Disjunctive Logic Programming and Databases: Nonmonotonic Aspects, July 1–5, 1996.

[Jeffrey, 1965] Richard C. Jeffrey. *The logic of decision.* McGraw-Hill, New York, 1965.

[Kamp, 1979] J. A. W. Kamp. The logic of historical necessity, part I. unpublished typescript, 1979.

[Kartha and Lifschitz, 1994] G. Neelakantan Kartha and Vladimir Lifschitz. Actions with indirect effects (preliminary report). In *Proc. of the Fourth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 341–350, 1994.

[Kartha, 1993] G. Neelakantan Kartha. Soundness and completeness theorems for three formalizations of action. In *Proc. of IJCAI-93*, pages 724–729, 1993.

[Kartha, 1995] G. Neelakantan Kartha. A mathematical investigation of reasoning about actions. Dissertation, The University of Texas at Austin, 1995.

[Kautz and Selman, 1992] Henry Kautz and Bart Selman. Planning as satisfiability. In J. Lloyd, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92)*, Vienna, Austria, 1992.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, 1996.

[Kautz, 1986] Henry Kautz. The logic of persistence. In *Proc. of AAAI-86*, pages 401–405, 1986.

[Kim, 1974] Jaegwon Kim. Noncausal connections. *Noûs*, 8:41–52, 1974.

[Koons, 1995] Robert C. Koons. The logic of causation. Unpublished manuscript, 1995.

[Korf, 1985] Richard E. Korf. Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[Kowalski, 1974] Robert Kowalski. Predicate logic as a programming language. In J.L. Rosenfeld, editor, *Information Processing, 1974*, pages 569–574. North Holland, 1974.

[Lewis, 1973] David Lewis. *Counterfactuals*. Harvard University Press, Cambridge, Massachusetts, 1973.

[Lewis, 1986a] David Lewis. Causal decision theory. In David Lewis, editor, *Philosophical Papers, Volume II*, pages 305–339. Oxford University Press, New York, 1986.

[Lewis, 1986b] David Lewis. A subjectivist's guide to objective chance. In David Lewis, editor, *Philosophical Papers, Volume II*, pages 83–132. Oxford University Press, New York, 1986.

[Lifschitz and Rabinov, 1989] Vladimir Lifschitz and Arkady Rabinov. Things that change by themselves. In *Proc. of IJCAI-89*, pages 864–867, 1989.

[Lifschitz, 1987] Vladimir Lifschitz. Formal theories of action. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence, Proc. of the 1987 Workshop*, pages 35–58, 1987.

[Lifschitz, 1990] Vladimir Lifschitz. Frames in the space of situations. *Artificial Intelligence*, 46:365–376, 1990.

[Lifschitz, 1991] Vladimir Lifschitz. Towards a metatheory of action. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the Second Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 376–386, 1991.

[Lifschitz, 1994] Vladimir Lifschitz. Nested abnormality theories. Submitted for publication, 1994.

[Lifschitz, 1995] Vladimir Lifschitz. Two components of an action language. Unpublished manuscript, 1995.

[Lifschitz, 1996] Vladimir Lifschitz. Foundations of logic programming. In *Principles of Knowledge Representation*, volume 3, pages 69–127. CSLI publications, 1996.

[Lifschitz, 1997] Vladimir Lifschitz. A logic for causal reasoning. Unpublished manuscript, 1997.

[Lin and Reiter, 1994] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation, Special Issue on Actions and Processes*, 4(5):655–678, 1994.

[Lin, 1995] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI-95*, 1995.

[Manna and Waldinger, 1987] Zohar Manna and Richard Waldinger. How to clear a block: A theory of plans. *Journal of Automated Reasoning*, 3:343–377, 1987.

[Marek and Truszczyński, 1994] W. Marek and M. Truszczyński. Revision specifications by means of revision programs. In *Logics in AI. Proceedings of JELIA '94*. Lecture Notes in Artificial Intelligence. Springer-Verlag, 1994.

[McAllester and Rosenblitt, 1991] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proc. AAAI-91*, 1991.

[McCain and Turner, 1995] Norman McCain and Hudson Turner. A causal theory of ramifications and qualifications. In *Proc. of IJCAI-95*, pages 1978–1984, 1995.

[McCain and Turner, 1997] Norman McCain and Hudson Turner. On relating causal theories to other formalisms. Unpublished manuscript, 1997.

[McCarthy and Hayes, 1969] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969. Reproduced in [McCarthy, 1990].

[McCarthy, 1959] John McCarthy. Programs with common sense. In *Proc. of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationery Office. Reproduced in [McCarthy, 1990].

[McCarthy, 1963] John McCarthy. Situations, actions and causal laws. Stanford Artificial Intelligence Project, Memo 2, 1963.

[McCarthy, 1980] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1, 2):27–39,171–172, 1980. Reproduced in [McCarthy, 1990].

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986. Reproduced in [McCarthy, 1990].

[McCarthy, 1990] John McCarthy. *Formalizing common sense: papers by John McCarthy.* Ablex, Norwood, NJ, 1990.

[McDermott and Doyle, 1980] Drew McDermott and Jon Doyle. Nonmonotonic logic I. *Artificial Intelligence*, 13(1,2):41–72, 1980.

[Mendez *et al.*, 1996] G. Mendez, J. Lobo, J. Llopis, and C. Baral. Temporal logic in action description languages. In M. Zelkowitz and P. Straub, editors, *Proc. the XVI International Conference of the Chilean Computer Science Society*, pages 10–21, Valdivia, Chile, 1996.

[Montague, 1968] Richard Montague. Pragmatics. In R. Klibansky, editor, *Contemporary Philosophy a Survey*, pages 102–122. Florence, 1968.

[Moore, 1985a] Robert Moore. A formal theory of knowledge and action. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World*, pages 319–358. Ablex, Norwood, N.J., 1985.

[Moore, 1985b] Robert Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.

[Morris, 1988] Paul Morris. The anomalous extension problem in default reasoning. *Artificial Intelligence*, 35(3):383–399, 1988.

[Pearl, 1988] Judea Pearl. Embracing causality in default reasoning. *Artificial Intelligence*, 35:259–271, 1988.

[Pednault, 1987] Edwin Pednault. Formulating multi-agent, dynamic world problems in the classical planning framework. In Michael Georgeff and Amy Lansky, editors, *Reasoning about Actions and Plans*, pages 47–82. Morgan Kaufmann, San Mateo, CA, 1987.

[Pednault, 1989] Edwin Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In Ronald Brachman, Hector Levesque, and

Raymond Reiter, editors, *Proc. of the First Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 324–332, 1989.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132, 1980.

[Reiter, 1991] Raymond Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Robson, 1973] J. M. Robson. *The Collected Works of John Stuart Mill.* Univ. of Toronto Press, Toronto and Buffalo, 1973.

[Sandewall, 1992a] Erik Sandewall. *Features and fluents.* Oxford University Press, 1992.

[Sandewall, 1992b] Erik Sandewall. Features and fluents: A systematic approach to the representation of knowledge about dynamical systems. Technical Report LiTH-IDA-R-92-30, Linköping University, 1992.

[Sandewall, 1995] Erik Sandewall. Systematic comparison of approaches to ramification using restricted minimization of change. Technical Report LiTH-IDA-R-95-15, Linköping University, 1995.

[Scherl and Levesque, 1993] Richard B. Scherl and Hector Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the AAAI National Conference*, pages 689–695, 1993.

[Schubert, 1990] Lenhart Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H.E. Kyburg, R. Loui, and G. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, 1990.

[Selman *et al.*, 1994] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proceedings of AAAI-94*, pages 337–343, Seattle, WA, 1994.

[Shanahan, 1997] Murray Shanahan. *Solving the frame problem: a mathematical investigation of the common sense law of inertia.* MIT Press, Cambridge, MA, 1997.

[Shoham, 1986] Yoav Shoham. Chronological ignorance: Time, nonmonotonicity, necessity and causal theories. In *Proc. of AAAI-86*, pages 389–393, 1986.

[Shoham, 1987] Yoav Shoham. *Reasoning about change.* MIT Press, Boston, MA, 1987.

[Shoham, 1990] Yoav Shoham. Nonmonotonic reasoning and causation. *Cognitive Science*, 14:213–252, 1990.

[Stalnaker, 1968] Robert C. Stalnaker. A theory of conditionals. *American Philosophical Quarterly*, 2:98–112, 1968.

[Stalnaker, 1981] Robert C. Stalnaker. Letter to David Lewis. In W.L Harper, R. Stalnaker, and G. Pearce, editors, *Ifs: conditionals, belief, decision, chance, and time*, pages 153–190. D. Reidel Publishing Company, Dordrecht, Holland, 1981.

[Thielscher, 1994] Michael Thielscher. Representing actions in equational logic programming. In *Proc. ICLP-94*, pages 207–224, 1994.

[Thielscher, 1995a] Michael Thielscher. Computing ramifications by postprocessing. In *Proc. of IJCAI-95*, pages 1994–2000, 1995.

[Thielscher, 1995b] Michael Thielscher. On the logic of dynamic systems. In *Proc. of IJCAI-95*, pages 1956–1962, 1995.

[Thielscher, 1996] Michael Thielscher. Ramification and causality. Technical Report TR-96-003, International Computer Science Institute, 1996.

[Turner, 1994] Hudson Turner. Signed logic program. In *Logic Programming: Proceedings of the 1994 International Symposium*, pages 61–75, 1994.

[Turner, 1996] Hudson Turner. Splitting a default theory. In *Proceedings of AAAI-96*, pages 645–651, 1996.

[Turner, 1997a] Hudson Turner. Inference rules and causality in representations of common sense reasoning about actions. Dissertation, The University of Texas at Austin, 1997.

[Turner, 1997b] Hudson Turner. Representing actions in logic programs and default theories: A situation calculus approach. *Journal of Logic Programming*, 31(1–3):245–298, 1997.

[Van Belleghem *et al.*, 1996] Kristof Van Belleghem, Marc Denecker, and Daniele Dupré. Dependencies and ramifications in an event-based language. Draft, submitted, 1996.

[van Emden and Kowalski, 1976] Maarten van Emden and Robert Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[Van Gelder *et al.*, 1990] Allen Van Gelder, Kenneth Ross, and John Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, pages 221–230, 1990.

[von Wright, 1975] G. H. von Wright. The logic and the epistemology of the causal relation. In Ernest Sosa, editor, *Causation and Conditionals*, pages 95–113. Oxford University Press, London, 1975.

[Winslett, 1988] Marianne Winslett. Reasoning about action using a possible models approach. In *Proc. AAAI-88*, pages 89–93, 1988.

Vita

Norman Clayton McCain was born in Springfield, Missouri on December 3, 1950, the son of Norman Clayton McCain, Sr. and Dorothy Lou McCain. After obtaining a B.A. degree in philosophy from Baker University, he first attended graduate school in philosophy at the University of Kansas and then worked in bookstores for some years before returning to study computer science. After obtaining an M.S. degree in computer science from the University of Kansas in 1982, he worked in the Central Research Lab at Texas Instruments in Dallas for three and half years. He began the Ph.D. program at the University of Texas at Austin in 1986 and was married to Nancy Poteet Kaul in 1992. He has the following publications.

Norman McCain and Hudson Turner. Causal Theories of Action and Change. In the *Proceedings of AAAI-97*, 1997. To appear.

Norman McCain and Hudson Turner. A Causal Theory of Ramifications and Qualifications. In the *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, pages 1978–1984, 1995. (A preliminary version appeared in the *Working Notes: AAAI Spring Symposium on Extending Theories of Action*, pages 130–135, 1995.)

Vladimir Lifschitz, Norman McCain, Teodor Przymusinski, and Robert Stärk. Loop Checking and the Well-Founded Semantics, in *Logic Programming and Non-Monotonic Reasoning: Proceedings of the Third International Conference*, pages 127–142, 1995.

Norman McCain and Hudson Turner. Language Independence and Language Tolerance in Logic Programs. In the *Proceedings of the Eleventh International Conference on Logic Programming* (Pascal Van Hentenryck, editor), pages 38–57, MIT Press, 1994.

Vladimir Lifschitz, Norman McCain, and Hudson Turner. Automated Reasoning about Action: A Logic Programming Approach (abstract). In *Logic Programming: Proceedings of the 1993 International Symposium* (Dale Miller, editor), page 641, MIT Press, 1993.

Aditya Srivastava and Norman McCain. The Explorer PROLOG Toolkit, in the *Texas Instruments Engineering Journal*, pages 93–107, Vol. 3, No. 1, 1986.

Permanent Address: 812 Avondale Rd.
Austin, TX 78704

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1]LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.