Temporal Reasoning in the Situation Calculus

by

Javier Andrés Pinto

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

A fundamental problem in Knowledge Representation is the design of a logical language to express theories about actions and change. One of the most prominent proposals for such a language is John McCarthy's *situation calculus*, a formalism which views situations as branching towards the future. The situation calculus has been criticized for imposing severe representational limitations. For example, actions cannot be concurrent, properties change discretely, etc. In this thesis we show that many of these limitations can be overcome. Our work builds upon the *discrete situation calculus* and on Reiter's monotonic solution to the *frame problem.* A limitation of Reiter's approach is that it does not allow for *state constraints.* However, Lin and Reiter have made progress by providing a correctness criterion by which one can determine if an axiomatization can be said to solve the frame problem for theories that include state constraints.

In this thesis we extend Lin and Reiter's work on the ramification problem by providing mechanisms to deal with theories of action that include binary state constraints and/or stratified definitions. Furthermore, we show how to extend the situation calculus to deal with a wider range of representational issues. In particular, we provide an approach to represent determinate knowledge about the future. Also, we extend the situation calculus to deal with concurrent actions. This problem is addressed by separating the problem into a *precondition interaction* problem and an *effect interaction* problem. Moreover, we present an enriched ontology of the situation calculus that allows the representation of knowledge about continuous properties of the world. We introduce the notion of a *natural event*, which is the result of a process governed by the laws of nature. Finally, we show that the situation calculus provides a better logical foundation for reasoning about actions and time than some other popular temporal logics.

Acknowledgements

I would like to thank my thesis supervisor, Prof. Raymond Reiter, for his support and guidance during the development of this thesis. He has had a profound impact on my education which cannot be fully expressed. Among many other things, I have learned the value of clarity and precision in scientific work. Ray is and always will be a role model as a scientist.

I would also like to acknowledge the members of my thesis committee, Profs. Hector Levesque, Alberto Mendelzon, John Mylopoulos for their feedback and encouragement. Special thanks to my external examiners, Profs. Peter Apostoli and Vladimir Lifschitz. Both of them provided important feedback that led to improvements of my thesis.

It is difficult to find words to thank my friends Sheila McIlraith, Mike Gruninger, Steven Shapiro and Dale Schuurmans. They helped me through the many low points of graduate school and they shared my happiness in the good times. They provided me with a great deal of encouragement and support.

I would also like to thank Marc Denecker, Todd Kelley, Yves Lespérance, Fangzhen Lin, Rob Miller, Bart Selman, Murray Shanahan, Richard Scherl. They all provided help or feedback on my research.

Thanks to my wife Paz, and my sons Andrés and Tomás. Their generosity made possible the fulfillment of a dream. We have gone through rough times in the process and they never failed to be there with their love and understanding. They are the ones that give perspective to life.

Contents

Chapter 1

Introduction and Motivation.

In this thesis, we are concerned with the design of a logical language for the modeling of dynamic worlds; i.e., worlds that change with time. Surely, this is an old endeavor which has been tackled by many. In fact, many languages for describing dynamic phenomena have been proposed (e.g., STRIPS, Dynamic Logic, Modal Temporal Logics, Interval Temporal Logic, etc.). Our work builds upon the work of others who have put forward proposals to deal with this issue, but who have left many areas open for investigation. The research we rely upon has been carried out within the field of Computer Science and deals with issues that are relevant to Databases, Programming Languages, Software Engineering and Artificial Intelligence.

Our work is based on the *situation calculus*, which was originally conceived by John McCarthy [35]. McCarthy's objective was to provide a logical language to model the behavior of a single agent in a world that changes solely as consequence of the agent's actions. The original *situation calculus* is a first order theory that is based on the notion of *situation*. The intuition is that the world is in a given situation, which changes into another situation as a result of performing some action. It is assumed that the only way in which the world changes from one situation to another is by performing an action. Since an agent can perform a variety of different actions, there are many possible futures. These alternatives give rise to the *future-branching* structure of the situation calculus.

The situation calculus is a very appealing language which has been used to investigate many problems related to formal reasoning about change. For example, the language has been the formalism of choice for a great number of researchers interested in the so called *frame problem* (e.g., [40, 20, 21, 27, 54, 33, 9, 47]). So much so, that some researchers identify the frame problem as characteristic of theories based on the situation calculus, instead of as a problem inherent to the formalization of dynamic systems. Unfortunately, as has been pointed out elsewhere (e.g., [16, 23, 55]), the original situation calculus is a limited language that has several shortcomings. Nevertheless, as Gelfond, Lifschitz and Rabinov argue [16], these limitations can be overcome.

The objective of this thesis is twofold. On the one hand, we want to extend the expressiveness of the situation calculus to deal with several of its limitations. On the other, we want to address some open problems in the formalization of certain dynamic phenomena.

This document is organized as follows. In chapter 2, we present the formalization of the *discrete* situation calculus. This language has the same expressive power as early specifications of the situation calculus (e.g., McCarthy and Hayes [38]), with the notable exception that it restricts the set of existing situations to those that are reachable by performing finite sequences of actions from an initial situation. Also, we present the solution to the frame problem that was proposed by Reiter [47]. Reiter's approach is based on the derivation of a set of *successor state axioms*. Each successor state axiom specifies necessary and sufficient conditions to determine whether a property of the world holds after performing an action. These conditions refer to the state of the world in which the action is performed. Throughout this work, we will use this approach to deal with the *frame problem*. It is well understood that with Reiter's original framework, we cannot deal with *state constraints*, which are responsible for the *ramifications*. However, as discussed in chapter 2, Lin and Reiter [31] have studied the problem of dealing with theories of action that include state constraints. Their main result establishes that, under certain conditions, a set of successor state axioms can be used to replace the set of effect axioms and the state constraints.

In chapter 3, we consider the task of mechanically generating successor state axioms for theories of action containing a set of state constraints. To do this, we make use of Lin and Reiter's characterization of what constitutes a solution to the frame and ramification problems. The results provided are for theories whose state constraints are disjunctions with no more than two situation dependent literals. We argue that, under suitable conditions, reasoning with these theories is tractable. Furthermore, we show that a set of successor state axioms for theories that contain disjunctive state constraints with more than two situation dependent literals does not always exist. Finally, we show that reasoning with these theories is intractable. These results suggest that there is a fundamental limitation in reasoning about tasks requiring state constraints.

In chapter 4, we discuss some important limitations of the original situation calculus, namely its inability to express some basic temporal assertions; for example, that some event occurs in the future. Then, we present an extension to the situation calculus to address these limitations. This is done by augmenting the situation calculus with a notion of linear time, popular in temporal logics. The basic idea is to define the notion of an *actual* branch of situations. Thus, we identify one of the possible paths in the branching structure of situations as the *actual* branch. This *actual* branch depicts the way in which the world actually evolves through time. Based on this approach, an action is said to *occur* if and only if it is mentioned in the actual branch of situations. Also, we impose a time line on the path of actual situations, and extend the language to express temporal assertions like: *At three o'clock I will be at the office.* Finally, we argue that in order to completely characterize the actual branch of situations, we need to appeal to preferential semantics. In particular, we use circumscriptive semantics to identify the actual branch.

In chapter 5, we study how to augment the types of actions that can be used. There are two types of extensions that we consider. First, we take the notion of complex events, as proposed in [30], and study how it can be integrated with the actual line of situations. In particular, we show that, by appealing to the notion of occurrences, there is a natural way to specify complex actions, when they are built from primitive actions. A clear disadvantage of our approach based on occurrences is that it only works for the actual branch of situations. A second issue that we discuss in chapter 5, is how to extend the ontology of actions to incorporate *concurrent actions*. For this purpose, we introduce the binary function symbol $+$ to denote the concurrent execution of two actions or events. In order to study the problem of modeling concurrency, we divide the problem into three distinct cases. First, the trivial case in which the actions that are being performed concurrently are independent. Thus, they do not interact in any way. Second, the case in which there are interactions between the preconditions of the actions. For instance, cases in which the actions require the use of certain scarce resources (e.g., the dining philosophers example). Third, cases in which there are interactions between the effects of the concurrently executed actions. In these cases, such actions may either have effects that are not present when the actions are performed separately (i.e., the so called synergistic effects) or in which the actions cancel each others effects. Each of the three cases of concurrency is analyzed separately. Finally, we analyze how to integrate the notion of action occurrence with the ontology of actions extended with concurrent actions.

In chapter 6, we discuss the issue of continuity. That is, how to represent, within the situation calculus, properties that vary continuously with time; for example, the position of a moving ball. Gelfond, Lifschitz and Rabinov [16] proposed an extension to the situation calculus in which there is a continuum of situations. As discussed in chapter 2, the situation calculus that we use requires that the set of situations be enumerable. Therefore, a continuum of situations is incompatible with this view. In our approach, we describe a continuous property by using a fluent that states that this property's value is described by some continuous function of time. For example, we would have a fluent to state that *the motion of ball A is described by a parabola.* In the same manner as with other fluents, this fluent may change from situation to situation due to the occurrence of actions or events. Also in this chapter, a very important notion is introduced. This is the notion of a *natural event* or *natural action.* Intuitively, a natural event is one that occurs due to natural causes, and whose occurrence is not the result of an agent performing it. For example, if a ball is falling, the ball will naturally collide with the floor, barring intervention from an agent. The collision is considered natural and no agent "performs" the event. However, a *drop* action is not a natural action, rather, it is an action that requires an agent to bring it about. Of course, an agent may perform an action which leads to the occurrence of a natural action (e.g., a *drop* action initiates a situation in which $falling$ is true, which may lead to a *collide* event). The distinction between natural actions and agent-performed actions is essential if one wants to specify systems that reason about physical processes. For example, assume that we want to specify an agent that would have the mission of impeding the breakage of an egg. For the agent to be successful, it would need to infer that if the egg is falling it would hit the floor and probably break, unless the egg is stopped in midair. We also make use of the notion of *natural process*, which corresponds to a set of occurring natural actions.

In chapter 7, we look at three other approaches to temporal reasoning and study their relationship to the situation calculus. First, we look at the *interval temporal logic* [2, 3, 4, 5, 1], and discuss how its view of time can be, in most cases, modeled within the situation calculus. Second, we study in much greater detail the calculus of events [25, 24, 56], and argue that its formulation as a logic program has certain drawbacks. Also, we show that the functionality of the event calculus can be realized within the situation calculus. Furthermore, we present a logic program as an alternative to the logic programming formulation of the situation calculus. We analyze the correctness and completeness of the implementation with respect to a logical specification written in the language of the situation calculus extended with the notion of occurrences. Third, we discuss the so called modal logic of concurrency [19]. This logic deals with *pseudo*-concurrent actions, thus no true concurrency is possible. This logic, exemplifies the family of logics in which modal operators are utilized to model notions of propositions being *true now, true all times in the future, true in the next state*, etc. We show that the particular logic presented in [19] can be embedded within the extended situation calculus.

Finally, in chapter 8, we present the conclusions and discuss some of the avenues along which this research can be extended. We also summarize what we believe are the main contributions of this work.

Chapter 2

Formal Framework.

In this chapter we present the theoretical framework that constitutes the basis for the later work. First, we present the basic language of the situation calculus. Then, we discuss how to deal with two important problems in formalisms for reasoning about action and change, namely the *frame problem* and the *ramification problem.*

2.1 The Basic Situation Calculus.

2.1.1 Ontology.

The situation calculus is a logical language designed to represent theories of action and change. The language has its origins in the early work of John McCarthy [35]. The intuition behind the situation calculus language is that there is an *initial situation*, called S_0, and that the world changes from one situation to another when actions are performed. As will become clear later, we take the point of view that the only situations that exist are those that are reached by performing some sequence of actions in S_0. Furthermore, we consider the structure of situations to be a tree. That is, two different sequences of actions lead to different situations. Therefore, the tree structure allows for situations to be understood as a *state* along with the history of actions that led to it. In this context, the notion of *state* is equated with the situation dependent properties that hold in a situation. In this framework, situation dependent properties are called *fluents*, and a *state* corresponds to a set of fluents. Thus, the state of a situation is the set of fluents that *hold* in that situation. Notice that the branching factor of the tree of situations rooted at S_0 equals the number of distinct actions that exist.

In this section, we discuss a version of the situation calculus[1] which has evolved through a sequence of refinements [29, 47, 41, 31]. The situation calculus is a sorted second order language with equality. We use the standard operators conjunction, disjunction, implication and equivalence ($\wedge$, $\vee$, $\supset$ and $\equiv$ respectively), and standard universal and existential quantifiers. Scoping is indicated by parentheses or by the dot notation (e.g., $(\forall \mathbf{x}).\phi(\mathbf{x})$ is the same as $(\forall \mathbf{x})(\phi(\mathbf{x}))$). There are three domain independent sorts $\mathcal{A}, \mathcal{S}$, and $\mathcal{F}$, for action types, situations and propositional fluents[2]. We also include a sort $\mathcal{D}$ of domain objects, which may be further subdivided in subsorts depending on

[1]In the future, the term *discrete situation calculus* or simply *situation calculus* will refer to the language presented in this chapter.

[2]The term *propositional fluent* was used by McCarthy and Hayes [38] to mean a function of situations that would have as a range the set $\{true,\ false\}$. In the same spirit, we take a fluent to be a property of the world that may or may not *hold* at each situation in the tree.

the particular domain of application. Notice that all sorts are independent. Variables are denoted by lower case letters (with or without subscripts), and constants are denoted by upper case letters (with or without subscripts). Letters a, s, f, d (A, S, F, D) are used for variables (constants) of sorts $\mathcal{A}, \mathcal{S}, \mathcal{F}, \mathcal{D}$ respectively. We will have a single 1-place predicate variable φ over situations. Unless otherwise stated, other variables and constants will be assumed to be of sort $\mathcal{D}$. Also, when free variables appear in formulas, they are assumed to be universally quantified from the outside.

Strictly speaking, our language is second order. In fact, induction is needed in order to define the set of situations, as well as to define the notion of iterative actions discussed in chapter 5. Nevertheless, we restrict ourselves to a standard sorted first order sublanguage (with equality) to express all other axioms of our theories. If we wanted to eliminate the second order nature of the axiomatization of the situation calculus, we could simply consider "standard models" of the situation calculus axioms. Thus, we would replace the second order induction axiom with a first order induction schema, as done in first order number theory.

We include the special constant S_0 of sort $\mathcal{S}$, the function $do : \mathcal{A} \times \mathcal{S} \to \mathcal{S}$, the relation $< \subseteq \mathcal{S} \times \mathcal{S}$, the relation $Poss \subseteq \mathcal{A} \times \mathcal{S}$, the predicate $holds \subseteq \mathcal{F} \times \mathcal{S}$, along with other functions and relations introduced later. We use the abbreviation $s_1 \leq s_2$ to mean $s_1 < s_2 \vee s_1 = s_2$. Also, we use the notation $a < b < c$ to mean $a < b \wedge b < c$, and similarly when the expression combines $<$ and $\leq$. Another abbreviation we use is for terms of the form:

$$do(a_n, do(\ldots, do(a_1, s)\ldots)),$$

which we abbreviate as:

$$do([a_1, \ldots, a_n], s).$$

Keep in mind that $[a_1, \ldots, a_n]$ is not a term in the language and that it can only appear as a first argument in a *do* function term.

2.1.2 Basic Axioms and Language.

It has been pointed out (e.g., [29, 47]) that the sort of situations in the situation calculus can be formalized in much the same way as the sort of numbers is formalized in Peano's axioms in number theory. The formalization that we present here originates on Reiter's formalization that appears in [47].

The basic axioms for situations are:

$$(\forall \varphi).[\varphi(S_0) \wedge (\forall s, a)\ (\varphi(s) \supset \varphi(do(a, s)))] \supset (\forall s)\ \varphi(s), \tag{2.1}$$

$$(\forall a_1, a_2, s_1, s_2).do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2, \tag{2.2}$$

$$(\forall s_1, s_2, a).s_1 < do(a, s_2) \equiv s_1 \leq s_2, \tag{2.3}$$

$$(\forall s_1, s_2).s_1 < s_2 \supset \neg\, s_2 < s_1. \tag{2.4}$$

Axiom (2.1) is an induction axiom, necessary to prove properties true in all situations. See Reiter [49] for examples of the need for, and use of, induction in the situation calculus. The induction axiom (2.1) defines the universe of situations as the smallest set that includes S_0 and such that if s is in the set of situations and a is an element of $\mathcal{A}$ then $do(a, s)$ is also in the set of situations.

Axioms (2.1) and (2.1) establish the properties of $<$. Intuitively, $s < s'$ is true if and only if s and s' fall in the same branch of the tree of situations and s is closer to the root (S_0) than s'.

Axiom (2.1) along with the axioms for $<$ ensure that the structure of situations forms a tree. This is confirmed by the properties of the tree structure discussed by proposition 2.1 below.

Also, we utilize the predicate $Poss \subseteq \mathcal{A} \times \mathcal{S}$ that is true whenever an action can be performed in a situation. This is a predicate that has to be axiomatized as part of the background axioms of any application domain.

Our notion of $<$ differs from the one used by Reiter [47] in an important way. For us, $s < s'$ is true if and only if there exists a sequence of actions $a_1, a_2, \ldots, a_k$ such that:

$$s' = do([a_1, a_2, \ldots, a_k], s). \tag{2.5}$$

In contrast, Reiter's notion is stronger, in the sense that not only must a sequence of actions exist that satisfies (2.5), but also every action along the sequence must be possible in the situations in which they are *do*ne. In a later chapter, we use this alternative notion. Therefore, we introduce this second notion as a new relation $\prec \subseteq \mathcal{S} \times \mathcal{S}$ and with the abbreviation $\preceq$ such that:

$$s \preceq s' \equiv s \prec s' \vee s = s'. \tag{2.6}$$

Based on Lin and Reiter [31] $\prec$ is axiomatized as:

$$\neg s \prec S_0, \tag{2.7}$$
$$s_1 \prec do(a, s_2) \equiv Poss(a, s_2) \wedge s_1 \preceq s_2. \tag{2.8}$$

In what follows Σ_b will denote the axioms (2.1)-(2.1) and (2.7)-(2.7).

Proposition 2.1 *Some consequences of axioms Σ_b are*[3]*:*

$$(\forall s)\ S_0 \leq s, \tag{2.9}$$
$$(\forall a, s)\ \neg\, s < s, \tag{2.10}$$
$$(\forall s)\ \neg s < S_0, \tag{2.11}$$
$$(\forall s).s \neq S_0 \supset (\exists\, a, s')\ s = do(a, s'), \tag{2.12}$$
$$(\forall s_1, s_2, s_3).s_1 < s_2 \wedge s_2 < s_3 \supset s_1 < s_3, \tag{2.13}$$
$$(\forall a, s)\ s < do(a, s), \tag{2.14}$$
$$(\forall a_1, a_2, s_1, s_2).do(a_1, s_1) < do(a_2, s_2) \supset s_1 < s_2, \tag{2.15}$$
$$(\forall a, s, s').do(a, s) \leq s' \supset s < s', \tag{2.16}$$
$$(\forall a, s_1, s_2).do(a, s_1) = do(a, s_2) \supset s_1 = s_2, \tag{2.17}$$
$$(\forall a, s, s')\neg(s < s' < do(a, s)), \tag{2.18}$$
$$(\forall s_1, s_2).s_1 \prec s_2 \supset s_1 < s_2. \tag{2.19}$$

Finally, let us introduce McCarthy's *abnormality* predicate *ab* as:

$$ab(f, a, s) \equiv \neg[holds(f, do(a, s)) \equiv holds(f, s)]. \tag{2.20}$$

That is, a fluent f is abnormal with respect to an action a in a situation s if the action makes f change from s to $do(a, s)$. Many non-monotonic approaches to the frame problem are based on the minimization of this predicate (e.g., [9, 29]); however, we introduce it only as a shorthand. In particular, we use it to simplify the presentation of the following proposition:

[3]All proofs are in the appendix.

Proposition 2.2 *From axioms (2.1)-(2.1) and (2.20), it follows that:*

$$s_1 < s_2 \wedge holds(f, s_1) \wedge \neg holds(f, s_2) \supset \\ (\exists\, s, a)\,(s_1 < do(a, s) \leq s_2) \wedge ab(f, a, s). \tag{2.21}$$

That is, if a fluent changes its truth value between two situations s_1 and s_2, it must be the case that some action a was responsible for this change. Furthermore, a situation s must exist such that both, s and $do(a, s)$ lie in the path between s_1 and s_2.

2.1.3 Reification.

An important difference between this and other versions of the situation calculus (e.g., McCarthy and Hayes [38], Reiter [47]), is that we include a fluent sort. In other words, our fluents are *reified.* The use of situation calculus with reified fluents is also very common (e.g., [27, 9, 32] and others).

In non-reified formulations of the situation calculus, fluents correspond to predicates that take one or more arguments. The last argument is the only one of sort situation. For instance, let $\mathcal{L}_{nr}$ denote a classic situation calculus language with non-reified fluents. In such a language, fluents are predicates, like $onTable_{nr}(x, s)$, which would be true if the object x had the property of being *onTable* in the situation s. On the other hand, in a situation calculus language with reified fluents (which we call $\mathcal{L}_r$), fluents (like *onTable*) are awarded the status of objects. Thus, to each $(n+1)$-ary fluent predicate symbol in $\mathcal{L}_{nr}$ corresponds an n-ary function symbol of sort fluent in the language $\mathcal{L}_r$. Hence, in a language with reified fluents, $onTable(x)$ would be a fluent term and instead of writing $onTable_{nr}(x, s)$ we write $holds(onTable(x), s)$.

Interestingly, as Steven Shapiro has pointed out[4], in the classical situation calculus, actions are already reified objects. Indeed, the function *do* plays much the same role in the language $\mathcal{L}_r$ with respect to actions, as the predicate *holds* does with respect to fluents. In fact, in a purely non-reified language, actions can be understood as functions from situations to situations. Therefore, instead of writing $do(a, s)$, in a purely non-reified language $\mathcal{L}_{pnr}$ one would write $a(s)$. For example, if $Drop(x)$ is an action term in a language with reified actions (e.g. $\mathcal{L}_r$ or $\mathcal{L}_{nr}$), then the purely non-reified representation of that action is the binary function symbol $Drop(x, s)$ of sort situation. Therefore, while in $\mathcal{L}_r$ one may write:

$$holds(onTable(x), do(Drop(x), s)).$$

In $\mathcal{L}_{pnr}$ one would write:

$$onTable(x, Drop(x, s)),$$

in which *onTable* is a predicate (a fluent), and *Drop* is a situation function symbol (an action). The main difference between $\mathcal{L}_{pnr}$ and $\mathcal{L}_r$ is that in $\mathcal{L}_{pnr}$ we eliminate from the language the sorts for actions and fluents along with the predicate symbol *holds* and function symbol *do.*

The advantages gained with the reification of fluents are essentially the same as with the reification of actions; namely, the ability to treat fluents as objects in the language. Therefore, in the reified language $\mathcal{L}_r$ we have the ability to quantify over fluents as well as over actions. For instance, in order to illustrate an advantage of reification, we may use the notion of state and formally define a predicate that would be true of two situations when the same fluents hold in each of them:

[4]Personal communication.

Definition 2.1 *The state of a situation is equal to the set of fluents that hold in that situation. Therefore, we can define the predicate sameState* $\subseteq \mathcal{S} \times \mathcal{S}$ *as:*

$$sameState(s, s') \equiv [(\forall f).holds(f, s) \equiv holds(f, s')].$$

This definition is not generally possible in a non-reified language, unless we have a fixed set of fluent predicate symbols $f_1 \dots f_n$. If this were the case, in $\mathcal{L}_{nr}$ we would write:

$$sameState(s, s') \equiv (\forall \mathbf{x}_1 \dots \mathbf{x}_n)[(f_1(\mathbf{x}_1, s) \equiv f_1(\mathbf{x}_1, s')) \wedge \dots (f_n(\mathbf{x}_n, s) \equiv f_n(\mathbf{x}_n, s'))]. \quad (2.22)$$

However, from an axiomatizer's point of view, this is undesirable, since every time a new fluent is added to the language, a new conjunct must be added to the definition of *sameState* (or any definition in which we universally quantify over fluents). As further evidence for the convenience of using reified fluents, notice that proposition 2.2 could not have been written in $\mathcal{L}_{nr}$, unless it were written as a second order sentence (quantifying over fluent predicates). Furthermore, several approaches to deal with the frame problem appeal to the minimization of the predicate *ab* (defined in (2.20)). For instance, Baker's solution to the frame problem [9, 10] uses a circumscription policy in which the predicate *ab* is minimized with the interpretation of the function symbol *do* allowed to vary. Such a policy cannot be expressed without reifying fluents. This motivates the following:

Observation 2.1 *The language of the situation calculus without reified fluents is strictly less expressive than the reified counterpart.*

It is easy to understand why this is the case; given a theory in a non-reified language we can easily build a theory in a reified language that is equivalent to the original theory. This is done by simply replacing every predicate fluent symbol with a function fluent symbol, plus axioms stating that the only fluents that exist are those mentioned and that they are not equal. Furthermore, the definition of *sameState* shows a simple and interesting definition that is not possible in the non-reified language (unless a second order variable for fluent predicates were introduced).

In spite of what we have just said, it seems that the most important advantage of reification is a simple matter of notational convenience. In fact, in general one wants to talk about a specific, fixed, and finite set of fluents, in which case definitions like (2.22) would be sufficient. Therefore, what is left is the technical advantage of being able to write general truths about fluents, without having to write them as some sort of schemata.

In the past, there has been some confusion over what reification is and is not. In particular, Shoham [58] calls his logic a "reified logic", and, as argued in [45], it is unclear in which sense his logic is reified. In Shoham's logic, it is not possible to take properties as objects and predicate about them or quantify over them. This ability seems to be the major advantage of using reified logics.

Adding to the confusion created by Shoham's unorthodox use of the term "reify", Bacchus *et al.* [8] developed a non-reified temporal logic and proved that it subsumed the "reified" logic of Shoham's. Their paper can be seen as a strong argument against reification. However, it does not seem to apply to the more standard notion of reification [17]. Actually, as was argued before, a reified language is in general more expressive than its unreified counterpart.

Now, what is the price that one has to pay for the conveniences of having reification of fluents? One drawback is that one has to add the new *holds* predicate. This is less "elegant" than the use of predicate fluents, especially when writing sentences involving several fluents. For example, in the non-reified language one may have the sentence:

$$(\forall x, y, s).onTable(x, s) \wedge y \neq x \supset onFloor(y, s) \vee onHand(y, s),$$

which in the reified language is written with a proliferation of *holds* literals:

$$(\forall\, x, y, s).holds(onTable(x), s) \wedge y \neq x \supset holds(onFloor(y), s) \vee holds(onHand(y), s).$$

Of course, there might be syntactic variants that would help make things more succinct, but the added complexity is certainly present. For example, we extend the sort of fluents, as some other researchers do (e.g., [9]), with functions from fluents to fluents that mimic propositional operators. Thus, we write:

$$holds(f_1 \wedge f_2, s) =_{def} holds(f_1, s) \wedge holds(f_2, s). \tag{2.23}$$

$$holds(\neg f, s) =_{def} \neg holds(f, s), \tag{2.24}$$

and similarly for other propositional operators. Here the symbols $\wedge$, $\neg$, etc. are being overloaded. They are both logical symbols and non-logical function symbols of sort fluent. The intended meaning of each of these symbols should be obvious from the context. Unfortunately, things get more complex if one wants to introduce fluents with quantification in them, and we refrain from doing so. We point out that the fluent function symbols that correspond to the propositional operators are used as mere abbreviations. Thus, the properties of the language (e.g., expressiveness) remain unaltered. The advantage of the abbreviations is that we avoid writing an excessive number of *holds* literals. Notice that expressions like $f \wedge f'$ are not terms in the language. They can only be used in abbreviations inside the *holds* predicate.

A second drawback of reification is the need to introduce a separate sort for fluents. Furthermore, we also need to add unique names axioms for fluents, which we describe later.

2.2 The Frame Problem.

One of the hardest problems encountered in reasoning about actions has been the so called *frame problem.* This problem has attracted much interest in the AI community and many solutions for it have been proposed (e.g., [18, 20, 27, 58, 47, 54, 33]). Basically, the problem is understood as one of finding a way to *succinctly* specify the effects of actions, given the situations in which they are performed. Most solutions to the frame problem (e.g., [58, 27, 33]) are based on non-monotonic logics. Lately, there has been interest in the development of solutions based on standard monotonic logics. In particular, we appeal to the solution proposed by Reiter [47], which is based on the work of Haas [20], Pednault [40], and Schubert [54].

In this section, we present Reiter's monotonic solution to the frame problem. This presentation is slightly different from the original due to the differences between our language and Reiter's.

A theory of action Σ consists of the following sets of axioms:

- A set Σ_{bd}, which includes
  - The set Σ_b of basic situation calculus axioms (refer to section 2.1.2).
  - A set Σ_d of situation independent axioms formalizing the domain of interest.
- A set T_{pos} of action precondition axioms. In general, we have a set of necessary conditions for executing an action. For example, we may know that to repair an object it is necessary that the object be broken and that we have glue. This is written as an implication, for instance:

$$Poss(repair(r, x), s) \supset holds(hasglue(r), s) \wedge holds(broken(x), s).$$

Thus, for each action A we have a set of axioms of the form:

$$Poss(A, s) \supset \pi_A^j(s), \tag{2.25}$$

where $\pi_A^j(s)$ is a *simple* state formula with a unique free situation variable s. The adjective *simple* is used in the sense of Reiter's. That is, a simple formula is a first order formula which does not mention the predicate symbols $Poss$, $\prec$ or $\preceq$ (nor $<$ or $\leq$), that mentions a unique situation variable (e.g. s) and no other situation terms, and which does not quantify over variables of sort *state* ($\mathcal{S}$) [48]. Furthermore, we require that s not appear outside the scope of a *holds* predicate (as its second argument).

In this work, we assume that all the necessary conditions for the execution of an action are known. This completeness assumption allows us to write for each action A, an axiom of the form:

$$Poss(A, s) \equiv \Pi_A(s).$$

This is obtained by applying predicate completion to the axioms of the form (2.25). That is $\Pi_A(s)$ is the disjunction:

$$\Pi_A(s) = \pi_A^1(s) \vee \ldots \vee \pi_A^n(s), \tag{2.26}$$

where $\pi_A^i(s)$ appears in (2.26) iff there is an action precondition axiom of the form

$$Poss(A, s) \supset \pi_A^i(s).$$

Therefore, the set T_{pos} is a set of equivalences that establish necessary and sufficient conditions for actions to be executable.

- A set T_{S_0} of sentences about S_0. That is, a description of the initial situation.
- A set $T_{ef} = T_{ef}^+ \cup T_{ef}^-$, where T_{ef}^+ is a set of general positive effect axioms, e.g., for fluent function R and action a:

$$Poss(a, s) \wedge \gamma_R^+(a, s) \supset holds(R, do(a, s)), \tag{2.27}$$

and T_{ef}^- is a set of general negative effect axioms, for fluent R and action a:

$$Poss(a, s) \wedge \gamma_R^-(a, s) \supset \neg holds(R, do(a, s)). \tag{2.28}$$

Here, $\gamma_R^+(a, s)$ and $\gamma_R^-(a, s)$ are simple formulas which are used to provide conditions under which an action a produces an effect on a fluent R. In (2.27) and (2.28) we omit all but the action and state arguments.

- A set T_{una} of uniqueness of names axioms for actions. For distinct action names a and a':

$$a(x_1, \ldots, x_n) \neq a'(y_1, \ldots, y_m).$$

Identical actions have identical arguments. Thus, for each function symbol of sort $\mathcal{A}$ in the language, we write:

$$a(x_1, \ldots, x_n) = a(y_1, \ldots, y_n) \supset x_1 = y_1 \wedge \ldots \wedge x_n = y_n.$$

- A set T_{unf} of unique names axioms for fluents. For distinct fluent names f and f':

  $$f(x_1,\ldots,x_n) \neq f'(y_1,\ldots,y_m).$$

  Identical fluents have identical arguments. Thus, for each function symbol of sort $\mathcal{F}$ in the language, we write:

  $$f(x_1,\ldots,x_n) = f(y_1,\ldots,y_n) \supset x_1 = y_1 \wedge \ldots \wedge x_n = y_n.$$

  Notice that this assumption is necessary only because we have chosen to reify fluents. In the future we will consider these unique names axioms to be part of the set Σ_d of domain specific axioms.

Following Schubert [54], the assumption that the general (positive or negative) effect axioms characterize all the conditions for a state s under which action a makes $holds(R, do(a,s))$ true can be formalized with an *explanation closure axiom* of the form:

$$Poss(a,s) \wedge \neg holds(R,s) \wedge holds(R, do(a,s)) \supset \gamma_R^+(a,s).$$

Where $\gamma_R^+(a,s)$ can be understood as an explanation why R changed from not holding in s to holding in $do(a,s)$, as result of performing action a. Similarly, the assumption that the general effect axioms characterize all the conditions for a state s under which action a makes $\neg holds(R, do(a,s))$ true is formalized with:

$$Poss(a,s) \wedge holds(R,s) \wedge \neg holds(R, do(a,s)) \supset \gamma_R^-(a,s).$$

Reiter provides an approach to mechanically integrate Schubert's explanation closure axioms with the general effect axioms to derive a set of *successor state axioms.* A successor state axiom for fluent R is of the form:

$$Poss(a,s) \supset [holds(R, do(a,s)) \equiv \gamma_R^+(a,s) \vee holds(R,s) \wedge \neg\gamma_R^-(a,s)]. \tag{2.29}$$

That is, given a situation, if an action is possible we can determine whether the fluent R will hold in the subsequent situation. For this approach to work, it is necessary to have a finite set of effect axioms. Reiter's proposal is to replace the effect axioms with the successor state axioms, yielding a new theory $\Sigma_{bd} \cup T_{ss}(T_{ef}) \cup T_{pos} \cup T_{S_0} \cup T_{una}$. Where we write $T_{ss}(T_{ef})$ to denote the successor state axioms that result from utilizing explanation closure axioms and the effect axioms T_{ef}. So, for each function fluent in the language, we have a successor state axiom. Notice that if for some fluent F there are no effect axioms, the resulting successor state axioms would state that:

$$Poss(a,s) \supset [holds(F, do(a,s)) \equiv holds(F,s)].$$

Thus, F does not change due to the performance of any action.

Interestingly, Reiter's monotonic solution to the frame problem is equivalent to the solution of Lin and Shoham's [32] in the absence of state constraints. More precisely, as shown in [31] and in [57], the models of $\Sigma_{bd} \cup T_{ss}(T_{ef}) \cup T_{pos} \cup T_{S_0} \cup T_{una}$ correspond to the minimal models of $\Sigma_{bd} \cup T_{ef} \cup T_{pos} \cup T_{S_0} \cup T_{una}$ given by Lin and Shoham's minimality criterion, based on minimization of change [32]. The minimality criterion is discussed in more detail in the next section.

Reiter's proposal solves the frame problem by providing an axiomatization that requires a number of axioms proportional to the sum of fluent and action function symbols plus the number of actions. As already mentioned, a limitation of this approach is that it does not consider theories that include axioms that constrain the *legal* ways in which fluents can change. For example, consider an axiomatization in which a one handed robot can perform the action *Drop*, which has the effect of leaving its hand empty. If we had a constraint like: *an object is either in the hand or on the floor*, it should be possible to infer that a ramification of a *Drop* action is that whatever is dropped will be on the floor. The ramification problem is addressed later.

Notice that, by virtue of reification, instead of writing one axiom per fluent (like (2.29)) we could write a single general frame axiom of the form:

$$Poss(a, s) \supset [holds(f, do(a, s)) \equiv \gamma^+(a, f, s) \vee holds(f, s) \wedge \neg\gamma^-(a, f, s)]. \tag{2.30}$$

In which $\gamma^+(a, f, s)$ is a formula stating the conditions in which a fluent f is made to hold as a result of some action a, and $\gamma^-(a, f, s)$ is a formula stating the conditions in which a fluent f is made not to hold as a result of some action a. This single frame axiom would also constitute a solution to the frame problem, provided that the axiom is of finite length. In fact, not only do we need the axiom to be finite in length but we also want its length to be small. Otherwise, it could hardly be considered a solution to the frame problem. In theory, the length of the axiom would be proportional to the number of action function symbols times the number of fluent function symbols.

For example, suppose that we have a world in which there is a single movable object. Furthermore, assume that we describe the position of the object using a fluent function symbol *pos*, such that $pos(x)$ holds iff the object is at position x. Also, assume that we have a single action function symbol *move*, such that $move(x)$ is true iff the object is moved to position x. Then, we may write:

$$\gamma^+(a, f, s) \equiv a = move(x) \wedge f = pos(x),$$
$$\gamma^-(a, f, s) \equiv a = move(x) \wedge f = pos(y) \wedge x \neq y.$$

In most cases, causality relationships between fluents and actions will allow for axioms of shorter length. For example, we may be able to assume that each fluent is affected by at most k_a kinds of actions (each kind identified by a different action function symbol). In this case the length of the frame axiom would be proportional to the number of fluent function symbols in the language.

2.3 The Ramification Problem.

In [31], Lin and Reiter studied the problem of extending Reiter's approach to the frame problem to theories that include a set of state constraints. In particular, they came up with a set of sufficient conditions under which a set of successor state axioms can be said to correctly solve the frame problem. The notion of *correctness* of the solution is understood in terms of its equivalence to a specific non-monotonic solution to the frame problem. In this section, we summarize the definitions and results of [31] that we will use later. Our presentation deviates from the original in several minor ways. This is so since we need to accommodate the linguistic differences.

Formally, we extend the theories of action Σ with:

- A finite set $T_{sc}(s)$ of domain dependent state constraints[5], such that each constraint is a simple formula with a unique free variable s. In the future we will write T_{sc} to refer to the universal closure of $T_{sc}(s)$. We also use $T_{sc}(s)$ to refer to the conjunction of all the state constraints applied to the situation term s.

[5]For the purpose of this work, we consider only what Lin and Reiter call *effect-oriented* state constraints, ignoring the *precondition oriented* state constraints.

Now, we denote with Σ_{sc} the theories of action extended with state constraints. Thus, a theory of action Σ_{sc} is as follows:

$$\Sigma_{sc} = \Sigma_{bd} \cup T_{ef} \cup T_{pos} \cup T_{S_0} \cup T_{una} \cup T_{sc}. \tag{2.31}$$

In Lin and Reiter's proposal, a solution to the frame and ramification problem for a theory Σ_{sc} is a theory $\Sigma_{sc'}$ such that:

$$\Sigma_{sc'} = \Sigma_{bd} \cup T_{ss}(T_{ef}, T_{sc}) \cup T_{pos} \cup T_{S_0} \cup T_{una}, \tag{2.32}$$

where $T_{ss}(T_{ef}, T_{sc})$ represents a set of successor state axioms derived from the effect axioms and the state constraints. Naturally, not any set $T_{ss}(T_{ef}, T_{sc})$ will correspond to a correct solution. In order to assess the correctness of a proposed solution, Lin and Reiter take an alternative non-monotonic solution to the frame problem as a correct solution. Thus, a set of successor state axioms is regarded as a solution if and only if the models of the theory $\Sigma_{sc'}$ (with a simple extension) are the same as the non-monotonic models of Σ_{sc}. These non-monotonic models are defined using a minimal model semantics.

Lin and Reiter's key result states that the minimal models of Σ_{sc} are exactly the models of $\Sigma_{sc'}$ conjoined with the state constraints applied to the initial situation and the impossible situations[6]. The notion of minimality is based on Lin and Shoham's [32]. In order to define the minimality criterion more formally, let s, a, a', f and f' denote variables of sorts $\mathcal{S}$, $\mathcal{A}$, $\mathcal{F}$ (according to our convention). Also, let σ_s, σ_a and σ_f denote assignment functions from free situation, action and fluent variables to domain objects of the respective sorts. A model $\mathcal{M}'$ is preferred to a model $\mathcal{M}$ (i.e., $\mathcal{M}' < \mathcal{M}$) iff the following conditions hold:

- $\mathcal{M}'$ and $\mathcal{M}$ share the same domain and they interpret everything the same with the possible exception of the predicate *holds*.
- For some variable assignment to situations σ_s, $\mathcal{M}'$ and $\mathcal{M}$ are such that:
  - For any assignment σ_f, $\mathcal{M}'$ and $\mathcal{M}$ agree on $holds(f, s)$, that is:
  $$\mathcal{M}', \sigma_s, \sigma_f \models holds(f, s)$$
  if and only if:
  $$\mathcal{M}, \sigma_s, \sigma_f \models holds(f, s).$$
  - For any assignments σ_a, σ_f, if
  $$\mathcal{M}, \sigma_s, \sigma_a, \sigma_f \models s \prec do(a, s) \land \neg ab(f, a, s),$$
  then
  $$\mathcal{M}', \sigma_s, \sigma_a, \sigma_f \models \neg ab(f, a, s).$$
  - There are two assignments σ_a, σ_f such that:
  $$\mathcal{M}, \sigma_s, \sigma_a, \sigma_f \models s \prec do(a, s) \land ab(f, a, s),$$
  but
  $$\mathcal{M}', \sigma_s, \sigma_a, \sigma_f \models \neg ab(f, a, s).$$

[6]A situation s is impossible iff $\neg S_0 \preceq s$.

Finally, $\mathcal{M}$ is minimal if there is no $\mathcal{M}'$ such that $\mathcal{M}' < \mathcal{M}$. For further details, the reader is referred to [32, 31].

Lin and Reiter's result is formally stated as follows:

Theorem 2.3.1 *Given a theory Σ_{sc} as in (2.31), let $T_{ss}(T_{ef}, T_{sc})$ be a set of successor state axioms (one for each fluent function symbol) of the form:*

$$Poss(a, s) \supset holds(f(x_1, \dots, x_n), do(a, s)) \equiv [\Psi_f \vee (holds(f(x_1, \dots, x_n), s) \wedge \neg\Psi_{\neg F})],$$

where f is an n-ary fluent function symbol. Ψ_f and $\Psi_{\neg f}$ are simple state formulas whose free variables are among $a, s, x_1, \dots x_n$, and that satisfy the following:

$$\Sigma_{sc} \models (\forall\, a, s, x_1, \dots, x_n).Poss(a, s) \supset [\Psi_f \supset holds(f(x_1, \dots, x_n), do(a, s))],$$

and

$$\Sigma_{sc} \models (\forall\, a, s, x_1, \dots, x_n).Poss(a, s) \supset [\Psi_{\neg f} \supset \neg holds(f(x_1, \dots, x_n), do(a, s))].$$

Also, let $\Sigma_{sc'}$ be the set of axioms obtained from Σ_{sc} by replacing T_{ef} and T_{sc} with $T_{ss}(T_{ef}, T_{sc})$. Assume that the following conditions hold:

1.

$$T_{una} \models (\forall\, a, s, x_1, \dots, x_n).\neg(\Psi_f \wedge \Psi_{\neg f}),$$

2.

$$\Sigma_{sc'} \cup T_{sc}(S_0) \models (\forall\, s).S_0 \preceq s \supset T_{sc}(s).$$

Given these conditions, an interpretation $\mathcal{M}$ is a model of:

$$\Sigma_{sc'} \cup T_{sc}(S_0) \cup T_{sc}^{-}$$

if and only if $\mathcal{M}$ is a minimal model of Σ_{sc}, where T_{sc}^{-} is the set of state constraints restricted to states reached by performing impossible actions, thus:

$$T_{sc}^{-} = (\forall\, s).\neg S_0 \preceq s \supset T_{sc}(s).$$

The first condition in the theorem, $T_{una} \models (\forall\, a, s, x_1, \dots, x_n).\neg(\Psi_f \wedge \Psi_{\neg f})$, should be understood as a consistency condition. In fact, it is a requirement that no single action have the effect of making a fluent F true in some situation and the effect of making the fluent false in the same situation.

The second condition imposes the requirement that the new theory with a set of successor state axioms entail the state constraints whenever these are satisfied by the initial situation. Thus, we should guarantee that the updates performed using the successor state axioms preserve the consistency of the successor state with respect to the state constraints. Therefore, the state constraints are now implicit in the successor state axioms.

The relevance of Lin and Reiter's result is that it provides a criterion to judge whether or not a set T_{ss} of successor state axioms constitutes a correct solution to the frame problem. Unfortunately, this correctness criterion is not absolute. In fact, it relies completely upon a very specific solution to the frame problem based on a non-monotonic logic. Therefore, by using this result we are simply providing *evidence* to support the correctness of Reiter's solution based on successor state axioms. This evidence would be strengthened if Lifschitz's belief[7], that under certain circumstances all solutions to the frame problem based on model minimality are equivalent, were to be confirmed.

Also, we will need the following:

[7]As quoted in [31]

Observation 2.2 *The theory $\Sigma_{sc'} \cup T_{sc}(S_0) \cup T_{sc}^-$ is categorical in the following sense:*
Let $\mathcal{M}$ and $\mathcal{M}'$ be two models of the theory of action $\Sigma_{sc'} \cup T_{sc}(S_0) \cup T_{sc}^-$, such that:

- *$\mathcal{M}$ and $\mathcal{M}'$ share the same domain.*
- *$\mathcal{M}$ and $\mathcal{M}'$ interpret everything the same except for holds.*
- *$\mathcal{M}$ and $\mathcal{M}'$ agree on the state of S_0. Thus, if σ_f is an arbitrary variable assignment for fluents, then:*
$$\mathcal{M}, \sigma_f \models holds(f, S_0),$$
if and only if:
$$\mathcal{M}', \sigma_f \models holds(f, S_0).$$

If the previous conditions hold, then:

$$\mathcal{M}, \sigma_f, \sigma_s \models S_0 \preceq s \wedge holds(f, s),$$

if and only if:

$$\mathcal{M}', \sigma_f, \sigma_s \models S_0 \preceq s \wedge holds(f, s).$$

where σ_f and σ_s are arbitrary variable assignment functions for fluents and situations respectively. Thus, if two models agree on the initial state, then they agree on all states that are reachable from S_0 by performing possible actions only.

In the next chapter, we propose an approach to generate successor state axioms obtained from a set of effect axioms and a set of state constraints. In order to justify the correctness of the approach, we make use of theorem 2.3.1.

Chapter 3

State Constraints and Successor State Axioms.

In the previous chapter we noted that Reiter's original proposal to deal with the frame problem [47] was not general enough to deal with theories that include a set of state constraints. Then, we briefly outlined Lin and Reiter's results to deal with state constraints and solve the so called ramification problem. In summary, they provide sufficient conditions that a set of successor state axioms must satisfy in order for it to be considered a solution to the combined frame and ramification problems.

In this chapter, we extend this line of work by studying theories that have either of two kinds of state constraints:

The first kind includes a set of state constraints that are clauses containing at most two *holds* literals. We show that a simple syntactic manipulation can be used to generate a suitable set of successor state axioms. We also show that, in general, such a procedure cannot exist for theories that include clauses with more than two *holds* literals. In [47], Reiter taught us how to derive successor state axioms from theories that expressed the effects of actions in the so called *effect axioms.* Our approach to deal with theories that also include a set of *state constraints* is to find a way to replace these state constraints with new effect axioms along with some simple sentences about the initial state S_0. Thus, we look for a new theory that would fall under the class of theories for which Reiter's approach works and that would have the state constraints implicit in the effect axioms and in the axioms about S_0.

The second type of state constraint we consider are *definitions.* Basically, we argue that it is common to deal with theories in which there are *primitive* fluents, along with *defined* fluents. Primitive fluents, are those directly affected by actions. On the other hand, non-primitive fluents are defined in terms of the primitive ones. We characterize these types of theories by establishing the notion of *stratified definitions* and show how to deal with the frame and ramification problems within these theories.

3.1 Binary State Constraints.

Consider a theory Σ_{sc} in which the set T_{sc} of state constraints is written in semi-clausal form. That is, each constraint is a sentence of the form:

$$\rho(\mathbf{x}_r) \vee holds([\neg]f_1(\mathbf{x}_1), s) \vee holds([\neg]f_2(\mathbf{x}_2), s) \vee \ldots \vee holds([\neg]f_m(\mathbf{x}_m), s). \qquad (3.1)$$

In this sentence, $\rho(\mathbf{x}_r)$ represents a formula that does not contain any situation terms. We call this form semi-clausal because $\rho(\mathbf{x}_r)$ is not necessarily a literal. Here, each f_i is a fluent function symbol. The notation $[\neg]$ signifies that the symbol $\neg$ may or may not appear in front of the fluent term. Here, we write $holds(\neg R, s)$ as abbreviation for $\neg holds(R, s)$ as defined in (2.23). In order to simplify variable substitutions, we require that the tuples $\mathbf{x}_1 \ldots \mathbf{x}_m$ of variables be disjoint, and the tuple $\mathbf{x}_r$ include the variables in $\mathbf{x}_1 \ldots \mathbf{x}_m$. This requirement is not as restrictive as it may seem. For example, if we have the following state constraint:

$$holds(onHand(x), s) \vee holds(onTable(x), s),$$

we conform to the requirement by rewriting it as:

$$x_1 = x_2 \supset holds(onHand(x_1), s) \vee holds(onTable(x_2), s).$$

In the future, we will use the following:

Definition 3.1 *A state constraint of the form (3.1) is called an m-ary state constraint.*

Our goal in this section is to show that for the simplest case in which there are no state constraints with more than two *holds* literals (i.e., binary constraints) it is easy to derive a solution to the frame problem based on a set of successor state axioms. Thus, we provide a mechanical procedure to generate a set of successor state axioms to replace the effect axioms and the binary state constraints. We prove that the successor state axioms generated satisfy the properties necessary to apply Lin and Reiter's result discussed in the previous chapter.

The solution is derived as follows: We start with a theory Σ_{sc_2}:

$$\Sigma_{sc_2} = \Sigma_{bd} \cup T_{pos} \cup T_{una} \cup T_{S_0} \cup T_{ef} \cup T_{sc_2},$$

where T_{sc_2} is a set of binary state constraints. The first step is to find an alternative formulation of this theory without state constraints. In fact, we derive a set $T'_{ef}(T_{sc_2}, T_{ef})$ of new effect axioms and form the theory:

$$\Sigma'_{sc_2} = \Sigma_{bd} \cup T_{pos} \cup T_{una} \cup T_{S_0} \cup T_{ef} \cup T'_{ef}(T_{sc_2}, T_{ef}).$$

Later, we prove that if Σ_{sc_2} is consistent, then the theory Σ'_{sc_2} is equivalent to Σ_{sc_2}, given a simple condition on the initial situation. In the next step, we use Σ'_{sc_2} to derive a set of successor state axioms $T_{ss}(T'_{ef}(T_{sc_2}, T_{ef}))$. Thus, we finally obtain the theory:

$$\Sigma_{ss_2} = \Sigma_{bd} \cup T_{pos} \cup T_{S_0} \cup T_{una} \cup T_{ss}(T'_{ef}(T_{sc_2}, T_{ef})).$$

The set $T_{ss}(T'_{ef}(T_{sc_2}, T_{ef}))$ is derived directly following Reiter's approach to the frame problem in the absence of state constraints, described in section 2.2. Finally, we use theorem 2.3.1 to show that Σ_{ss_2} constitutes a solution to the frame and ramification problem for the original theory.

Thus, we describe the procedure to obtain $T'_{ef}(T_{sc_2}, T_{ef})$ from a theory Σ_{sc_2}:
Let

$$\rho(\mathbf{x}_r) \vee holds([\neg]f_1(\mathbf{x}_1), s) \vee holds([\neg]f_2(\mathbf{x}_2), s)$$

be an arbitrary binary state constraint in T_{sc_2}. In order to simplify the presentation, we describe how to obtain new effect axioms for state constraints of the form:

$$\rho(\mathbf{x}_r) \vee holds(f_1(\mathbf{x}_1), s) \vee holds(f_2(\mathbf{x}_2), s).$$

Thus, we ignore the case in which we have negative *holds* literals. However, the same discussion applies to constraints for negative literals. The only change is that if $f(\mathbf{x})$ is a fluent term with a $\neg$ in front, i.e. $\neg f(\mathbf{x})$, we reverse the roles of *negative* and *positive.*

Each negative effect axiom for f_1 in T_{ef} is of the form[1]:

$$Poss(a,s) \wedge \gamma_{f_1}^-(a,s) \supset \neg holds(f_1(\mathbf{x}_1), do(a,s)). \tag{3.2}$$

For each such axiom, we generate the following positive effect axiom for f_2:

$$Poss(a,s) \wedge \gamma_{f_1}^-(a,s) \wedge \neg\rho(\mathbf{x}_r) \supset holds(f_2(\mathbf{x}_2), do(a,s)). \tag{3.3}$$

Symmetrically, if we have an effect axiom:

$$Poss(a,s) \wedge \gamma_{f_2}^-(a,s) \supset \neg holds(f_2(\mathbf{x}_2), do(a,s)), \tag{3.4}$$

we add:

$$Poss(a,s) \wedge \gamma_{f_2}^-(a,s) \wedge \neg\rho(\mathbf{x}_r) \supset holds(f_1(\mathbf{x}_1), do(a,s)). \tag{3.5}$$

Notice that (3.3) and (3.5) are entailments of the original theory Σ_{sc_2}. The previous procedure is used with each and every state constraint, and is repeated until no new effect axioms can be generated.

Now, we have the following:

Proposition 3.1 *Let*

$$\Sigma_{sc_2} = \Sigma_{bd} \cup T_{pos} \cup T_{una} \cup T_{S_0} \cup T_{ef} \cup T_{sc_2},$$

be a theory of action as described before, and let

$$\Sigma'_{sc_2} = \Sigma_{bd} \cup T_{pos} \cup T_{una} \cup T_{S_0} \cup T_{ef} \cup T'_{ef}(T_{sc_2}, T_{ef}),$$

be the theory obtained from Σ_{sc_2}, in which the set T_{sc_2} is replaced with the new effect axioms $T'_{ef}(T_{sc_2}, T_{ef})$, according to the procedure described above. Then, the following holds[2]*:*

$$\Sigma'_{sc_2} \models T_{sc_2}(S_0) \supset (\forall s).S_0 \preceq s \supset T_{sc_2}(s).$$

That is, given a theory Σ_{sc_2} if we replace the binary state constraints with the set $T'_{ef}(T_{sc_2}, T_{ef})$ we obtain a theory that entails the state constraints whenever the initial state satisfies them. This result is proven by induction[3]*, using $\varphi(s) = S_0 \preceq s \supset T_{sc_2}(s)$.*

This proposition establishes that we can replace the binary state constraints with a new set of effect axioms and the state constraints applied to the initial state only. Thus, Σ'_{sc_2} is a theory suitable to apply Reiter's solution to the frame problem. Hence, we obtain:

$$\Sigma_{ss_2} = \Sigma_{bd} \cup T_{pos} \cup T_{una} \cup T_{S_0} \cup T_{sc_2}(S_0) \cup T_{ss}(T'_{ef}(T_{sc_2}, T_{ef})), \tag{3.6}$$

which constitutes the solution to the frame and ramification problems for the original theory Σ_{sc_2}. In order to guarantee that this is a solution to the problem, we have the following:

Theorem 3.1.1 *The set Σ_{ss_2} (as in (3.6)) is a solution to the frame and ramification problem for the theory Σ_{sc_2}, as long as the following consistency condition is satisfied:*

$$T_{una} \models (\forall a, s, x_1, \ldots, x_n).\neg(\Psi_f \wedge \Psi_{\neg f}),$$

where Ψ_f and $\Psi_{\neg f}$ are as defined in the statement of theorem 2.3.1.

This theorem follows directly from theorem 2.3.1 and proposition 3.1.

[1]It may be necessary to rename the variables in the effect axiom and state constraints to make the fluent term $f_1(\mathbf{x}_1)$ coincide. Since this renaming operation is straightforward, we ignore it to make the presentation more concise.

[2]Recall that $T_{sc_2}(s)$ denotes the conjunction of all the state constraints applied to the situation term s.

[3]See appendix.

Example: To illustrate how to derive the successor state axioms from a basic theory of action, consider the following example:

- Σ_d (situation independent axioms):

$$(\forall\, d).d = Saucer \lor d = Cup.$$

- T_{pos}:

$$\begin{aligned}&Poss(pickup(d), s) \equiv \neg(\exists\, d')holds(inHand(d'), s),\\ &Poss(drop(d), s) \equiv holds(inHand(d), s).\end{aligned}$$

- T_{ef}:

$$\begin{aligned}&Poss(pickup(d), s) \supset holds(inHand(d), do(pickup(d), s)),\\ &Poss(drop(d), s) \supset \neg holds(inHand(d), do(drop(d), s)).\end{aligned}$$

- T_{sc}:

$$\begin{aligned}&d = d' \supset holds(inHand(d), s) \lor holds(onFloor(d'), s),\\ &d = d' \supset \neg holds(inHand(d), s) \lor \neg holds(onFloor(d'), s).\end{aligned}$$

- T_{S_0}:

$$\begin{aligned}&holds(onFloor(Saucer), S_0),\\ &holds(onFloor(Cup), S_0),\\ &\neg holds(inHand(Cup, S_0)),\\ &\neg holds(inHand(Saucer, S_0)).\end{aligned}$$

Therefore, following the procedure outlined before, we can derive the following new effect axioms:

- $T'_{ef}(T_{sc}, T_e f)$:

$$\begin{aligned}&Poss(pickup(d), s) \land d = d' \supset \neg holds(onFloor(d'), do(pickup(d), s)),\\ &Poss(drop(d), s) \land d = d' \supset holds(onFloor(d'), do(drop(d), s)).\end{aligned}$$

Thus, the new theory is built by replacing axiom set T_{sc} with the set:

- $T_{ss}T'_{ef}((T_{sc}, T_e f))$:

$$\begin{aligned}&Poss(a, s) \supset holds(inHand(d), do(a, s)) \equiv\\ &\qquad\qquad a = pickup(d) \lor holds(inHand(d), s) \land \neg a = drop(d),\\ &Poss(a, s) \supset holds(onFloor(d), do(a, s)) \equiv\\ &\qquad\qquad a = drop(d) \lor holds(onFloor(d), s) \land \neg a = pickup(d).\end{aligned}$$

It is simple to check that $T_{sc}(S_0)$ is satisfied by $\Sigma_d \cup T_{S_0}$. Furthermore, the condition:

$$T_{una} \models (\forall a, s, x_1, \ldots, x_n).\neg(\Psi_f \wedge \Psi_{\neg f}),$$

gets translated into:

$$T_{una} \models (\forall a, d)\ \neg(a = pickup(d) \wedge a = drop(d)),$$

which is trivially true.

The previous results have interesting computational consequences. In fact, a very important advantage of theories of action described using successor state axioms for each fluent is that they lend themselves to easy update procedures. For instance, if we know the state of a situation s, then determining whether or not a fluent f holds in a successor situation $do(a, s)$ depends only on the evaluation of the simple formulas Ψ_f and $\Psi_{\neg f}$. Thus, the computational cost of updating the value of a fluent equals the cost of determining the value of the Ψ formulas.

These theories are also interesting because, as shown below, extending them to allow for more general disjunctive state constraints (i.e., m-ary state constraints with $m > 2$) leads to non-tractability. Furthermore, we also show that, in general, it is not possible to develop a procedure that would generate successor state axioms for these extended theories. Thus, we have:

Proposition 3.2 *Let Σ_{sc+} be a theory Σ_{sc}, defined as in (2.31), such that T_{sc} contains m-ary state constraints with $m > 2$. A set $T_{ss}(T_{ef}, T_{sc})$ of successor state axioms satisfying the conditions of theorem 2.3.1 for Σ_{sc+} need not exist.*

Proof: Assume a theory in which T_{sc} contains the single constraint:

$$holds(X, s) \vee holds(Y, s) \vee holds(Z, s),$$

with T_{S_0}:

$$holds(X, S_0) \wedge \neg holds(Y, S_0) \wedge \neg holds(Z, S_0),$$

and single effect axiom:

$$Poss(A, s) \supset \neg holds(X, do(A, s)).$$

Also, assume that the only fluents that exist are those denoted by the constants X, Y and Z. It is easy to see that this theory has no unique minimal models. In fact, there would be at least two minimal models, one in which $holds(Y, do(A, S_0))$ is true and another in which $holds(Z, do(A, S_0))$ is true. In other words, the effects of performing action A are *uncertain*. Therefore, given observation 2.2 and the fact that the state of S_0 is fixed, a set of successor state axioms that solves the frame and ramification problems does not exist.
□

As discussed before, the limitation of binary constraints is suggestive of a more essential barrier. In fact, we show below that propositional theories that contain ternary state constraints are in general intractable. By *propositional*, in this context, we mean theories in a situation calculus language whose fluent function symbols have arity 0 (i.e., they are all constants):

Definition 3.2 *A situation calculus theory is called propositional if its language is such that there is a fixed set of fluent constants $f_1, \ldots, f_n$, and there are no other fluent function symbols. Also, the theory must contain a closure axiom stating that there are no fluents other than those denoted by $f_1, \ldots, f_n$.*

Proposition 3.3 *Let Σ_{sc} be a propositional theory of action as defined in (2.31), such that T_{sc} contains m-ary state constraints of the form:*

$$holds([\neg]f_1(\mathbf{x}_1), s) \vee holds([\neg]f_2(\mathbf{x}_2), s) \vee \ldots \vee holds([\neg]f_m(\mathbf{x}_m), s). \tag{3.7}$$

Consider the problem of constructing a model $\mathcal{M}$ for Σ_{sc}. Assume that s and a are a situation and action in the domain of $\mathcal{M}$. Furthermore, assume that the state of s has been determined. Also, assume that we have a truth assignment for all situation independent atoms. Under these circumstances, we have that the task of finding a state for the situation $do^{\mathcal{M}}(a, s)$[4] has the following properties:

1. *If $m \leq 2$, the time that it takes to perform this task is polynomial in the number of fluent constants in the language and in the size of T_{ef} and T_{sc}.*
2. *If $m > 2$, this task is intractable.*

Therefore, there are important reasons why ternary constraints are difficult. Unless the set of ternary constraints was equivalent to a set of binary constraints they introduce ambiguity. Having ambiguity precludes the use of Reiter's monotonic solution to the frame problem. Furthermore, reasoning with ternary constraints is intractable.

In [28], Vladimir Lifschitz presented a nonmonotonic solution to the frame problem in the presence of ramifications. Lifschitz uses a reified situation calculus and circumscription to formalize his approach. The main result is that, under certain conditions, the result of the circumscription is a complete characterization (explicit definitions) of the circumscribed predicates. Thus, when these conditions are satisfied, the complete characterization can be used as a monotonic solution to the frame problem.

Interestingly, one of the conditions in Lifschitz's result is that the state constraints be expressible through a formula[5]:

$$Compatible(f_1, f_2)$$

where the fluent variables f_1 and f_2 are the only free variables in $Compatible(f_1, f_2)$ and there is no mention of the *holds* predicate. Furthermore, it is required that the following is satisfied:

$$S_0 \leq s \wedge holds(f_1, s) \wedge holds(f_2, s) \supset Compatible(f_1, f_2).$$

Thus, assume that s is a situation reached by performing a sequence of possible actions. This condition requires that the state of s be such that any pair of fluents that hold in it are *Compatible*. The interesting aspect of this requirement is that a set of binary state constraint can be written in such a way that Lifschitz's conditions are satisfied.

It appears that, under certain circumstances, Lifschitz solution to the frame problem, in the presence of ramifications, is equivalent to our approach to the frame problem in the presence of binary state constraints. A formal analysis of the relationship between the two frameworks is left for further research.

3.2 Stratified Definitions.

In this section we consider a different kind of extension, in which we allow the introduction of non-recursive definitions as a new set T_{def} of state constraints. We identify a set of fluents as *primitive*

[4] $do^{\mathcal{M}}$ denotes the interpretation of *do* in model $\mathcal{M}$.

[5] We have adapted and simplified Lifschitz's condition to our language.

fluents, such that the non-recursive definitions bottom out on the primitive fluents. Furthermore, we require that the effect axioms refer only to primitive fluents. This is a rather straightforward problem. However, as argued later, theories with definitions arise often and we need a systematic way to deal with them.

First, we introduce the notion of *stratified definitions*[6].

Definition 3.3 *A set T_{def} is a set of stratified definitions if it is composed of a set of sentences, each of the form:*

$$holds(f_0(\mathbf{x}), s) \equiv \Phi_0(f_1, \ldots, f_n, \mathbf{x}, s),$$

where Φ_0 is a simple formula, and $f_0, \ldots, f_n$ are fluent function symbols[7]*. f_0 is called a* defined *fluent. The set T_{def} must be such that there exists a numbering scheme that assigns a non-negative integer to all the fluent function symbols, such that for each sentence in the set T_{def} the number associated with the defined fluent is strictly greater than the number assigned to the fluents on the right side of the definition.*

Definition 3.4 *Let S and S' be two different numbering schemes for a set T_{def} of stratified definitions. Also, let $S(f)$ and $S'(f)$ denote the numbers that each scheme associate with the fluent f. We define the ordering $<$ between two numbering schemes as:*

$$S < S',$$

if and only if there exist a fluent f_0 such that $S(f_0)$ is strictly less than $S'(f_0)$, and for every fluent f, other than f_0, $S(f)$ is less than or equal to $S'(f)$.

Definition 3.5 *A numbering scheme is called a* stratification, *iff it is minimal with respect to $<$.*

It should be obvious that there exists at most one stratification for any set of definitions. Given a stratification, we call the number associated with a fluent its *stratum*. Primitive fluents are those that are assigned a stratum of zero.

Definition 3.6 *A set T_{def} is a normal set of definitions iff it has a stratification that assigns a stratum of 1 to all its defined fluents.*

Observation 3.1 *Given a set T_{def} of stratified definitions there is a normal set of definitions T'_{def} that is equivalent to T_{def}.*

Thus, without loss of generality, in the rest of this chapter, we simply assume that the set of definitions is *normal.*

Now, we restrict our attention to theories Σ_{def} of the following kind:

$$\Sigma_{def} = \Sigma_{bd} \cup T_{pos} \cup T_{una} \cup T_{S_0} \cup T_{ef} \cup T_{def},$$

in which the effect axioms refer only to effects on primitive fluents. Thus, if we have an effect axiom like:

$$Poss(a, s) \wedge \gamma_F^+(a, s) \supset holds(F, do(a, s)), \qquad (3.8)$$

[6]The term *stratified* is used in the same spirit as the notion of stratification applied to logic programs and deductive databases [42].

[7]Fluent terms may only appear as a first argument of a *holds* predicate.

then F must be a primitive fluent. We are interested in finding a solution to the frame and ramification problems for a theory Σ_{def}. We propose two ways in which this can be done.

First, the simplest solution is to consider a language without the defined fluents. Given a set T of sentences, let the set $\mathcal{T}_{T_{def}}(T)$ denote the set of sentences in which the defined fluents have been eliminated from T. This elimination is trivial, each *holds* literal in T, whose first argument is a defined fluent, is replaced by the right side of its definition in T_{def}. Since T_{def} is normal, the resulting set contains only primitive fluents. Let Σ_{-def} be:

$$\Sigma_{-def} = \mathcal{T}_{T_{def}}(\Sigma_{bd}) \cup \mathcal{T}_{T_{def}}(T_{pos}) \cup T_{una} \cup \mathcal{T}_{T_{def}}(T_{S_0}) \cup \mathcal{T}_{T_{def}}(T_{ef}).$$

Then, we obtain the set of successor state axioms $T_{ss}(\mathcal{T}_{T_{def}}(T_{ef}))$ utilizing Reiter's solution to the frame problem and obtain the theory:

$$\Sigma'_{def} = \Sigma_{bd} \cup T_{ss}(\mathcal{T}_{T_{def}}(T_{ef})) \cup T_{pos} \cup T_{S_0} \cup T_{una} \cup T_{def}, \tag{3.9}$$

as solution to the frame and ramification problems. The correctness of this solution is a direct consequence of theorem 2.3.1. Obviously, the theory $\Sigma'_{def} - T_{def}$ represents a correct solution of the frame and ramification problems for the theory Σ_{-def}, as long as the consistency condition

$$T_{una} \models (\forall\, a, s, x_1, \ldots, x_n).\neg(\Psi_f \wedge \Psi_{\neg f})$$

is met. Therefore, given the same consistency condition, Σ'_{def} can be considered a solution for Σ_{def}.

A second approach to solve the frame and ramification problems for the theory Σ_{def} is to derive successor state axioms for the defined fluents as well as for the primitive fluents. To do so, we take the set $T_{ss}(\mathcal{T}_{T_{def}}(T_{ef}))$ (from (3.9)) which are successor state axioms for the primitive fluents. We obtain a set T^{ss}_{def} of successor state axioms for the defined fluents by applying regression, as defined in [48], to the primitive fluents in each definition. Thus, if f_0 is a defined fluent, and its definition is of the form:

$$\mathit{holds}(f_0(\mathbf{x}), s) \equiv \Phi_0(f_1, \ldots, f_n, \mathbf{x}, s),$$

where fluents $f_1, \ldots, f_n$ are primitive, then:

$$\mathit{holds}(f_0(\mathbf{x}), \mathit{do}(a, s)) \equiv \Phi_0(f_1, \ldots, f_n, \mathbf{x}, \mathit{do}(a, s)), \tag{3.10}$$

Now, we may use Reiter's regression operator $\mathcal{R}$ [48], which is meant to reduce the depth of nesting of the function symbol *do*. Thus, assume the following is a successor state axiom for primitive fluent f_i in $T_{ss}(\mathcal{T}_{T_{def}}(T_{ef}))$:

$$Poss(a, s) \supset [\mathit{holds}(f_i(\mathbf{x}_i, \mathit{do}(a, s)) \equiv \Gamma_{f_i}(s)].$$

where $\Gamma_{f_i}(s)$ is a simple state formula. Then, to apply the operator $\mathcal{R}$ to Φ_0, each literal

$$\mathit{holds}(f_i(\mathbf{y}_i), \mathit{do}(a, s))$$

in Φ_0 is replaced by Γ_{f_i} after performing the obvious variable substitutions. This is done with each primitive fluent in Φ_0. We obtain the formula

$$\mathcal{R}(\Phi_0(f_1, \ldots, f_n, \mathbf{x}, \mathit{do}(a, s))),$$

which is a simple formula that does not mention the function symbol *do*. Therefore, from (3.10) we obtain:

$$Poss(a,s) \supset holds(f_0(\mathbf{x}), do(a,s)) \equiv \mathcal{R}(\Phi_0(f_1,\ldots,f_n,\mathbf{x},do(a,s))), \tag{3.11}$$

which is a suitable successor state axiom for f_0. For each defined fluent in T_{def} we obtain a successor state axiom. Hence, we postulate that the theory:

$$\Sigma^{ss}_{def} = \Sigma_{bd} \cup T_{pos} \cup T_{una} \cup T_{S_0} \cup T_{ss}(\mathcal{T}_{T_{def}}(T_{ef})) \cup T^{ss}_{def},$$

represents a correct solution to the frame and ramification problem for Σ_{def}. To guarantee the correctness of this solution, we use theorem 2.3.1. In fact, using induction, (3.10) and (3.11) it follows that:

$$\Sigma^{ss}_{def} \models (\forall\, s).S_0 \preceq s \supset T_{def}(s),$$

where $T_{def}(s)$ denotes the conjunction of the definitions, each instantiated with the term s. Therefore, if the consistency condition:

$$T_{una} \models (\forall\, a, s, x_1, \ldots, x_n).\neg(\Psi_f \wedge \Psi_{\neg f})$$

is satisfied, the theory Σ^{ss}_{def} is a solution to the frame and ramification problem for Σ_{def}.

Theories with stratified definitions are of practical relevance. Changes in defined fluents are ramifications of the actions in this type of theories. Certainly, it is the simplest case of ramifications. Definitions or equivalences of this kind also arise when completeness assumptions are made. For example, given a set of necessary conditions under which some fluent holds, we may want to assume that these conditions are also sufficient. We may formalize this assumption by *predicate completion* or some such mechanism, deriving an equivalence, which behaves like a simple definition. For example, assume that we have the following set of axioms:

$$\begin{aligned}
&holds(Sick,s) \supset holds(Sad,s),\\
&holds(Trouble,s) \supset holds(Sad,s),\\
&\neg holds(Love,s) \supset holds(Sad,s).
\end{aligned}$$

These axioms provide sufficient conditions for *Sad* to hold. Thus, a subject is *Sad* if he is *Sick* or if he is in *Trouble* or if he is not in *Love*. Now, we may want to make the assumption that this sufficient conditions are also necessary, and use completion to obtain:

$$holds(Sad,s) \equiv holds(Sick,s) \vee holds(Trouble,s) \vee \neg holds(Love,s),$$

Now, for this to work properly with our approach, it must be the case that no action is such that it has the effect of making *Sad* true directly. Thus, we disallow an effect axiom of the form:

$$Poss(a,s) \wedge \gamma^{+}_{Sad}(a,s) \supset holds(Sad, do(a,s)). \tag{3.12}$$

Otherwise, if we eliminate the fluent *Sad*, we would obtain:

$$Poss(a,s) \wedge \gamma^{+}_{Sad}(a,s) \supset holds(Sick,s) \vee holds(Trouble,s) \vee \neg holds(Love,s), \tag{3.13}$$

which does not conform to the standard form of effect axioms.

As another example, consider the problem of modeling the behavior of an arbitrary combinational circuit without feedback, in which time delays are ignored. Such a device can be abstracted as a black box with a set of input signals and a set of output signals. Each signal corresponds to a Boolean value. For instance, we can have a circuit with two inputs (*Ga* and *Gb*) and two outputs (Gc and Gd), in which Gc is $Ga \vee Gb$ and Gd is $Ga \wedge Gb$. We represent each signal with a propositional fluent. The input signals will be considered to be the *primitive* fluents. In this simple example, we only have switch actions for the input gates: *SwitchGa* and *SwitchGb*, with the following set T_{ef} of effect axioms:

$$Poss(a,s) \wedge a = SwitchGa \wedge holds(Ga,s) \supset \neg holds(Ga, do(a,s)), \tag{3.14}$$

$$Poss(a,s) \wedge a = SwitchGa \wedge \neg holds(Ga,s) \supset holds(Ga, do(a,s)), \tag{3.15}$$

$$Poss(a,s) \wedge a = SwitchGb \wedge holds(Gb,s) \supset \neg holds(Gb, do(a,s)), \tag{3.16}$$

$$Poss(a,s) \wedge a = SwitchGb \wedge \neg holds(Gb,s) \supset holds(Gb, do(a,s)). \tag{3.17}$$

Switching actions are always possible:

$$Poss(SwitchGa, s) \wedge Poss(SwitchGb, s).$$

Finally the set T_{def} of definitions is simply:

$$\begin{aligned} holds(Gc,s) &\equiv holds(Ga,s) \vee holds(Gb,s), \\ holds(Gd,s) &\equiv holds(Ga,s) \wedge holds(Gb,s). \end{aligned}$$

The set $T_{ss}(T_{ef})$ derived from the set T_{ef} is:

$$\begin{aligned} & Poss(a,s) \supset holds(Ga, do(a,s)) \equiv \\ & \qquad \neg holds(Ga,s) \wedge a = SwitchGa \vee holds(Ga,s) \wedge \neg a = SwitchGa, \\ & Poss(a,s) \supset holds(Gb, do(a,s)) \equiv \\ & \qquad \neg holds(Gb,s) \wedge a = SwitchGb \vee holds(Gb,s) \wedge \neg a = SwitchGb. \end{aligned}$$

That is, $T_{ss}(T_{ef})$ consists of successor state axioms for the primitive fluents only. Obviously, the defined fluents can be uniquely determined from the primitive fluents and the definitions. Also, by following the procedure outlined before, it is possible to derive successor state axioms for the defined fluents from the definitions and the successor state axioms for the primitive fluents. An advantage of doing so is that we would eliminate the definitions, maintaining the derived successor state axioms for non primitive fluents. This would guarantee that the non-primitive fluent's values be consistent with the original definitions. The successor state axiom for the defined fluent G_c is simply:

$$\begin{aligned} & Poss(a,s) \supset holds(G_c, do(a,s)) \equiv \\ & \qquad [\neg holds(Ga,s) \wedge a = SwitchGa \vee holds(Ga,s) \wedge \neg a = SwitchGa] \vee \\ & \qquad [\neg holds(Gb,s) \wedge a = SwitchGb \vee holds(Gb,s) \wedge \neg a = SwitchGb]. \end{aligned}$$

The successor state axiom for G_d is analogously determined. However, it seems simpler to keep the definitions and use them to evaluate non-primitive fluents after actions are performed.

3.3 Combining Classes of Constraints in a Single Theory.

In the previous sections we described ways in which we can deal with classes of constraints that were either binary state constraints or sets of stratified definitions. Both cases were discussed independently. Now, it is important to answer the questions: What happens if we combine both types of constraints in the same theory? Under what circumstances can we apply the previous approaches? To answer these questions, assume that we have a theory:

$$\Sigma_{sc} = \Sigma_{bd} \cup T_{pos} \cup T_{una} \cup T_{S_0} \cup T_{ef} \cup T_{def} \cup T_{sc},$$

where T_{def} are stratified definitions[8] and T_{sc} are other arbitrary state constraints. To deal with these theories, we first eliminate the defined fluents from Σ_{sc} following the procedure explained in section 3.2. After replacing all defined fluents by their definitions, we end up with a theory

$$\Sigma'_{sc} = \Sigma'_{bd} \cup T'_{pos} \cup T'_{S_0} \cup T_{una} \cup T'_{ef} \cup T'_{sc}.$$

Here, the theory $\Sigma_{sc'}$ is equivalent to Σ_{sc} in the sense that for any sentence that does not mention a defined fluent, the sentence is entailed by one theory iff it is entailed by the other. Now, the set T'_{sc}, can be divided into T'_{sc_2} and T'_{sc+}, where T_{sc_2} are all the state constraints that can be written as disjunctive state constraints with no more than two *holds* literals and such that $T'_{sc} = T'_{sc_2} \cup T'_{sc+}$. Now, we can eliminate the state constraints in T'_{sc_2} by using the approach described in section 3.1. This leads to a another theory:

$$\Sigma''_{sc} = \Sigma'_{bd} \cup T'_{pos} \cup T''_{S_0} \cup T_{una} \cup T''_{ef} \cup T'_{sc+}.$$

Therefore, we reduce the problem by eliminating state constraints and replacing them by other axioms (i.e., effect axioms and axioms about S_0). Now, we can directly use Reiter's approach to the frame problem if T'_{sc+} is empty. Otherwise, the problem remains open as to whether or not we can come up with a systematic approach to derive the successor state axioms. However, the point we make here is that if such an approach exists, it would cleanly integrate with the ones discussed previously in this chapter.

[8]We assume that the right side of the effect axioms only refer to primitive fluents.

Chapter 4

Adding a Time Line to the Situation Calculus.

4.1 Why a Time Line?

The situation calculus is a temporal logic in which time is seen as a branching structure. Intuitively, the structure of situations is a tree rooted at the initial situation S_0, as illustrated by figure 4.1. In general, the initial situation S_0 is taken as the present time. Thus, each branch that starts in S_0 can be understood as a *hypothetical future.* The tree structure of the situation calculus shows all possible ways in which the events in the world could unfold. Therefore, any arbitrary sequence of actions identifies a branch in the tree of situations.

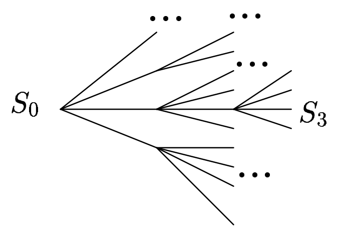


Figure 4.1: A tree of situations.

Thus, the situation calculus serves as a tool for the representation of *hypothetical knowledge.* As such it has several important drawbacks. In particular, it does not provide for ways to constrain the manner in which events occur. Also, it does not provide a notion of linear time. Therefore, propositions like "I will be home at three o'clock" cannot be represented in the language. We cannot even express the sentence: "I will die". This phenomenon is due to the branching nature of the underlying structure of situations. In fact, the language of the situation calculus does not provide the elements to *prefer* one development of events over others. For example, in figure 4.1 we may know that situation S_3 is real, in the sense that the actions that lead to it from S_0 have occurred. However, the original language lacks the power to make such a statement.

In contrast, formalisms based on linear time provide ways in which to express definite temporal knowledge about a given domain. For example, not only can we state that "I am at the office but I will be home at three o'clock," but we can also conclude that I will have to perform some action to take me home, and that such an action will have to be performed between now and three o'clock.

In the situation calculus, on the other hand, the most we can do is conclude that *if* I am going to be home at three o'clock, then I must do something that will take me there.

In this chapter we extend the language of the situation calculus by incorporating the basic elements of a linear temporal logic. This is an extension, in the sense that we introduce new distinctions. In particular, we talk about a subset of situations that are *actual* and provide an axiomatization that describes its properties. The set of actual situations represents the situations that arise in the actual world. This is illustrated with the continuous line in figure 4.2. This extension does not impose any additional constraints on the situations. Thus, we retain the expressive power of the original situation calculus.

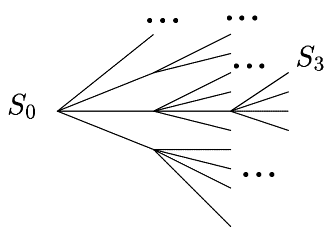


Figure 4.2: An actual line of situations.

As we see later in the chapter, the new distinctions introduced have a wide range of applicability. The main expressive enhancement is the ability to state that a sequence of events has occurred or will occur. One of the consequences of such an ability is that we can write sentences describing constraints among action occurrences. For example, we can write that one of the effects of some action is the triggering of other actions. Also, we may write that some actions have occurred without specifying completely which actions have occurred.

The discussion that follows concentrates on non-concurrent actions. How to apply the new notions introduced to concurrent actions is discussed later in this document.

4.2 An Actual Path of Situations.

So far, we have introduced axioms that allow for the specification of what is true and what truths change along different paths that start in the initial situation S_0. Here, we address the problem of describing the evolution of the world as it actually unfolds. To this end, we incorporate a predicate *actual* for situations. The intended meaning is that a situation is *actual* if it lies on the path that describes the world's real evolution. The axioms for *actual* are:

$$actual(S_0), \tag{4.1}$$

$$(\forall a, s).actual(do(a, s)) \supset actual(s) \wedge Poss(a, s), \tag{4.2}$$

$$(\forall a_1, a_2, s).actual(do(a_1, s)) \wedge actual(do(a_2, s)) \supset a_1 = a_2. \tag{4.3}$$

Axioms (4.1)-(4.1) express that the initial situation is always actual, and if a situation is actual, then its immediate predecessor must also be. Axiom (4.1) says that an actual situation has at most one actual successor situation. An important characteristic of actual situations is that they all lie on the same path (i.e., they constitute a *time line*), as the following shows:

Proposition 4.1 *From axioms (4.1)-(4.1) it follows:*

$$(\forall s_1, s_2).s_2 \leq s_1 \supset [actual(s_1) \supset actual(s_2)],$$
$$(\forall s_1, s_2).actual(s_1) \wedge actual(s_2) \supset s_1 < s_2 \vee s_2 < s_1 \vee s_1 = s_2.$$

We also introduce a notion of time, which allows us to establish a direct relationship between the situation calculus and linear temporal logics (e.g., the calculus of events [25]). Intuitively, each situation has a starting time and an ending time. During the time span of a situation no fluents change truth values[1]. We incorporate the sort $\mathcal{T}$, interpreted as a continuous time line, into our language. The sort $\mathcal{T}$ is considered isomorphic to the non-negative reals. We work with an interpreted theory, that is, we assume standard interpretations for the real numbers. Also, we introduce the functions *start* from $\mathcal{S}$ to $\mathcal{T}$, and *end*, from $\mathcal{S} \times \mathcal{A}$ to $\mathcal{T}$. Events/actions occur at the ending time of situations. This is captured by the following axioms[2]:

$$(\forall\, s, a)\ end(s, a) = start(do(a, s)), \tag{4.4}$$

$$(\forall\, a, s)\ start(s) < start(do(a, s)), \tag{4.5}$$

$$start(S_0) = 0. \tag{4.6}$$

Thus, each situation is labeled with a start time and these times begin at 0 in S_0 and increase monotonically away from the initial situation. Hence, we have the immediate consequence:

Proposition 4.2 *From (4.4)-(4.4) it follows that:*

$$(\forall\, s, s').s < s' \supset start(s) < start(s').$$

There exists an obvious asymmetry between the *end* and *start* functions. This is a reflection of the temporal asymmetry of the situation calculus. In fact, any given situation has a *unique* past. Therefore, for any single situation, there exists a unique *start*. On the other hand, any situation has multiple *possible* futures. Each possible future is reached by performing different sequences of actions. Thus, the end of a situation will depend on which future we consider. Hence, the term $end(s, a)$ denotes the time that would end the situation s if action a were to be performed. Since each actual situation has a unique actual successor situation, it turns out that the end time of actual situations is determinate.

Occurrences are introduced as a relation between *event types* and situations. For example:

$$occurs(pickup(D), S),$$

says that the event of picking up D occurred in situation S. Occurrences are defined in terms of the actual path as follows:

$$occurs(a, s) \equiv actual(do(a, s)). \tag{4.7}$$

Thus, each actual situation s, after S_0, is related to a unique action that must have occurred and which leads to s. For example, if $occurs(pickup(D), S)$ is true, the situation tree may look like the one in figure 4.3.

[1]This restricts the properties of the world that can be represented as fluents, for instance, the position of a moving ball cannot be represented by a fluent. This problem is addressed in chapter 6.

[2]In what follows, we use $<$ and $\leq$ as relations between situations (as defined earlier), as well as for the standard ordering relation between the elements of $\mathcal{T}$.

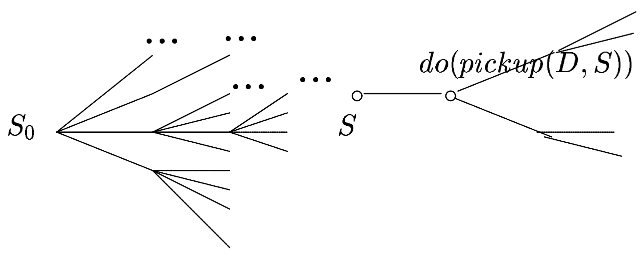


Figure 4.3: A tree of situations in which $occurs(pickup(D), S)$ is true.

In some cases, it might be convenient to establish a relationship between events/actions that occur and the time at which they occur (rather than the situation). For this purpose, we introduce a predicate $occurs_{\mathcal{T}} \subseteq \mathcal{A} \times \mathcal{T}$, defined as:

$$occurs_{\mathcal{T}}(a, t) \equiv (\exists\, s).occurs(a, s) \wedge start(do(a, s)) = t. \tag{4.8}$$

Also, we define a relation $holds_{\mathcal{T}}$ between fluents and time points and a relation *during* between time points and situations:

$$holds_{\mathcal{T}}(f, t) \equiv (\exists\, s).actual(s) \wedge during(t, s) \wedge holds(f, s), \tag{4.9}$$

$$during(t, s) \equiv actual(s) \wedge start(s) < t \wedge (\forall\, a)\ [occurs(a, s) \supset end(s, a) \geq t]. \tag{4.10}$$

Intra-state persistence is derivable:

Proposition 4.3 *From axioms (4.1)-(4.9) it follows:*

$$(\forall\, f, s, t, t').during(t, s) \wedge during(t', s) \supset holds_{\mathcal{T}}(f, t) \equiv holds_{\mathcal{T}}(f, t').$$

We can also introduce the notion of an event occurring between situations as follows:

$$occursBet(a, s_1, s_2) \equiv (\exists\, s).s_1 < s < s_2 \wedge occurs(a, s). \tag{4.11}$$

We can also show that if s_1 and s_2 are actual situations, and nothing occurs between them, then one must be the result of performing some action on the other. This is stated in the following:

Proposition 4.4 *From axioms (4.1)-(4.11) it follows:*

$$actual(s_2) \wedge s_1 < s_2 \wedge \neg(\exists\, a')\ occursBet(a', s_1, s_2) \supset (\exists\, a)\ s_2 = do(a, s_1).$$

In the next section we will consider theories that include sentences in which an ordering between situations is provided. For this purpose, we introduce the following:

Definition 4.1 *A formula $\mathcal{O}_<(x_1, x_2, \ldots, x_n)$ is an* **ordering formula** *if it only mentions the terms $x_1, x_2, \ldots, x_n$, does not include any quantifiers and all its literals are $<$ literals.*

For example, if $\mathcal{O}_<(x_1, x_2) = x_1 < x_2$ then $\mathcal{O}_<(s_1, s_2) = s_1 < s_2$ where $<$ refers to the ordering between situations, and $\mathcal{O}_<(t_1, t_2) = t_1 < t_2$ where $<$ refers to the less than operator for the real numbers.

Now, we have the following:

Proposition 4.5 *From proposition 2.1 and axioms (4.1)-(4.11):*

$$\begin{array}{l}(\exists s_1,\ldots,s_n).occurs(A_1,s_1)\wedge\ldots\wedge occurs(A_n,s_n)\wedge\mathcal{O}_<(s_1,\ldots,s_n)\\ \equiv\\ (\exists t_1,\ldots,t_n).occurs_{\mathcal{T}}(A_1,t_1)\wedge\ldots\wedge occurs_{\mathcal{T}}(A_n,t_n)\wedge\mathcal{O}_<(t_1,\ldots,t_n).\end{array}$$

where $\mathcal{O}_<$ is an ordering formula.

The expanded ontology for the situation calculus introduced in this section provides the essential features of other temporal logics. In fact, as we analyze in chapter 7, we can realize most of the representational features of some well known temporal logics based on a linear view of time. For example, Kowalski and Sergot's [25] Logic Programming based Calculus of Events and Allen's interval temporal logic [3].

4.3 What Occurs?

4.3.1 Motivation.

Adding the *occurs* predicate to the language does not introduce any complications in the case in which all the events that occur are completely determined. However, problems do arise when this is not the case. For instance, consider the Yale Shooting Problem [21]; it is common to pose it as the problem of determining the truth value of the literal:

$$holds(Alive, do(Shoot, do(Wait, do(Load, S_0)))).$$

That is, we want to know whether a fluent holds in a completely determined situation.

This problem can be formulated in a different fashion. In particular, using the situation calculus extended with a time line, we can formalize it as follows:

$$Poss(Load,s)\wedge Poss(Shoot,s), \tag{4.12}$$

$$Poss(Unload,s)\equiv holds(Loaded,s), \tag{4.13}$$

$$Poss(a,s)\wedge a=Load\supset holds(Loaded,do(a,s)), \tag{4.14}$$

$$Poss(a,s)\wedge a=Shoot\wedge holds(Loaded,s)\supset \neg holds(Alive,do(a,s)), \tag{4.15}$$

$$Poss(a,s)\wedge a=Unload\supset\neg holds(Loaded,do(Unload,s)), \tag{4.16}$$

$$holds(Alive,S_0), \tag{4.17}$$

$$(\exists s_1,s_2,s_3,s_4).occurs(Load,s_1)\wedge occurs(Wait,s_2)\wedge occurs(Shoot,s_3)\wedge s_1<s_2<s_3<s_4\wedge actual(s_4). \tag{4.18}$$

Axioms (4.12) and (4.12) specify T_{pos}. (4.12)-(4.12) are the effect axioms. (4.12) establishes the initial conditions. (4.12) lists all the actions that we know have occurred and provides an ordering among the situations in which the actions occur.

Following Reiter's mechanism to derive the successor state axioms, we obtain:

$$Poss(a,s)\supset[holds(Loaded,do(a,s))\equiv a=Load\vee\neg a=Unload\wedge holds(Loaded,s)], \tag{4.19}$$

$$Poss(a,s)\supset[holds(Alive,do(a,s))\equiv\neg(a=Shoot\wedge holds(Loaded,s))\wedge holds(Alive,s)]. \tag{4.20}$$

Then, we may ask whether *Alive* holds after the shooting. By formulating the *YSP* in this manner, there are two distinct problems that have to be studied:

- First, we have the *frame problem* whose solution allows us to determine the effects of each action in the environment, given the state of the situation in which they are *do*ne. Here, we assume that the frame problem is properly dealt with using Reiter's approach described in an earlier chapter. Thus, we assume that for each fluent we have a successor state axiom. It is important to point out that we assume that the frame problem is solved before we address the second problem described below. This requirement should be understood as giving higher priority to solving the frame problem.
- The second problem that we have to address is the one of determining exactly which actions occur. For instance, in the formulation of the *YSP* above, it could be the case that an *Unload* action occurred after the *Load* and before the *Shoot*. Also, between two situations there may be an infinite number of unrelated actions that may occur. However, we would like to infer that these extra actions do not occur. To deal with this problem, we take the view that some sort of minimization has to be used. In fact, our approach to address this problem follows a similar intuition to the one that serves as the basis for Reiter's solution to the frame problem. That is, in the solution to the frame problem, it is assumed that each action has no more effects than those that necessarily follow from the effect axioms. In a similar vein, we take the point of view that, after the frame problem is solved, no actions occur unless their occurrence is dictated by the axioms of the theory.

A very important issue arises when the frame problem is separated from the problem of determining the set of occurrences. To illustrate it, we introduce a simple abstract example. Consider a theory of action whose language includes a fluent constant F, situation constants S' and S'' and action constants A_1 and A_c. Also, assume that the only effect axiom is:

$$Poss(A_c, s) \supset \neg holds(F, do(A_c, s)).$$

Furthermore, assume that the theory satisfies the following sentence:

$$holds(F, S') \wedge \neg holds(F, S'') \wedge S' < S''.$$

That is, F has changed from S' to S''. We can easily conclude that some action must have occurred which negatively affected F (see proposition 2.2). So, assume that action A_1 is known to have occurred at S' or between S' and S'', that is:

$$occurs(A_1, S') \vee occursBet(A_1, S', S'').$$

Also, assume that there are no effect axioms relating A_1 to F. Now, given that this represents all we know about our domain, we have two possible ways of understanding the nature of the change in F:

1. Action A_1 has made F change negatively from S' to S''. Thus, A_1 affects F in a way that is not sanctioned by the effect axioms.
2. Some other action (e.g. A_c), whose occurrence we did not know about, occurred between S' and S'' and made F change from S' to S''.

Interestingly, the literature on the situation calculus and the frame problem is concerned only with the first sort of explanation. In fact, if all we know is that A_1 occurred, this is usually expressed as:

$$S'' = do(A_1, S').$$

Most non-monotonic solutions to the frame problem will conjecture that there is an unstated qualification which makes A_1 affect F negatively. In fact, non-monotonic solutions are concerned with explaining change due to A_1, and the only possible explanation (given that other actions are not postulated) is that A_1 is responsible for the change. In contrast, Reiter's monotonic solution to the frame problem would lead to a contradiction when using 1 as hypothesis. This is so since in the absence of effect axioms relating F and A_1, we will obtain a successor state axiom stating that F never changes from a situation s to a situation $do(A_1, s)$.

An alternative to explaining change by assuming unstated qualifications to known actions (e.g., assuming that a $Wait$ event may have unstated effects such as $Unload$ing guns) is to postulate the existence of intervening actions that explain the change. For instance, given the effect axiom that states that A_c always makes F false, Reiter's solution to the frame problem will allow for models in which such an action explains the changes in F. On the other hand, if there were no effect axioms relating actions to negative change in F, then using Reiter's solution to the frame problem would lead to a contradiction. This contradiction seems to be the only appropriate conclusion. In fact, if the theory does not mention any actions that would change F, then F should not change.

As has been described in this section, we follow the approach of *assuming* that we do not know all the actions that occur. However, we also assume that any action whose occurrence is not necessary to explain the information at hand does not occur. Thus, we divide the problem of reasoning about actions and change into two parts:

- First, given the state of a situation s, what is the state of a situation $do(a, s)$? This question is answered using Reiter's solution to the frame problem. Thus, we use *successor state axioms.*

- Second, given a possibly incomplete description of the actions that occur, what are the actions that occur? This is answered using the *non-intervening events* assumption. Thus, we choose models that contain minimal sets of occurring actions. This is formalized using circumscription.

Interestingly, Miller and Shanahan [39] study the formalization of *narratives* within the situation calculus. Narratives are conceptually similar to ordered sets of actions that are known to have occurred. In their work, they present an approach related to the one discussed here. They solve the first problem mentioned above by appealing to Baker's circumscriptive solution to the frame problem [10]. The second problem is also solved using a circumscriptive policy. They choose to minimize a predicate $Happens$, which plays a conceptually similar role to the predicate $occurs_{\mathcal{T}}$.

4.3.2 The Preferential Policy.

As mentioned before, we use a minimization policy in order to select models of a theory in which nothing occurs unless its occurrence is *necessary.* Thus, assume that Σ_{occ} is a theory of action with occurrences. Also, assume that Σ_{occ} has a model $\mathcal{M}$ in which all the actions that occur are as depicted in the time line of figure 4.4. Hence, in $\mathcal{M}$, the only action occurrences are A, B and C at

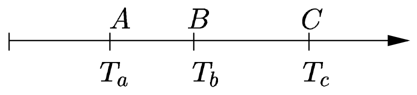


Figure 4.4: Time line in model $\mathcal{M}$.

times T_a, T_b and T_c respectively. Furthermore, assume that $\mathcal{M}'$ is another model of Σ_{occ} in which the occurrences are depicted in the time line of figure 4.5. In the model $\mathcal{M}'$, we have that the only

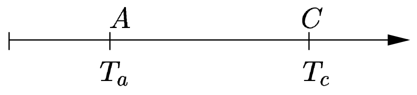


Figure 4.5: Time line in model $\mathcal{M}'$.

action occurrences are A and C at times T_a and T_c respectively. Our preferential semantics is such that the model $\mathcal{M}'$ should be preferred to $\mathcal{M}$. As partial justification for this semantics we will show that it yields the expected results when complete information can be assumed. Furthermore, in some cases this policy subsumes the policies stated by other researchers in the area. For example, Shanahan's assumption "that the only events which occur are those which are known to occur" [56] is subsumed by our policy when the chronological ordering between the actions is known. Thus, if it is consistent to make Shanahan's assumption, then our policy yields the same results when the chronological ordering of the actions is known.

In order to formalize the minimization, we use *Circumscription* [36, 37, 26]. To simplify the presentation of the circumscription policy, we consider the predicates *occurs*, $holds_{\mathcal{T}}$, *during*, and *occursBet* and the function *end* to be abbreviations (as defined by (4.7), (4.9), (4.9), (4.11) and (4.4)), which are not part of the extended language of the situation calculus. By doing this, we need not include these elements in the list of symbols whose interpretation is made variable in the circumscription policy.

In the following discussion, Σ_{occ} denotes a theory of action that includes:

- Axiom sets as in section 2.3, in which the sets T_{ef} and T_{sc} can be replaced with a set $T_{ss}(T_{ef}, T_{sc})$ of successor state axioms according to the criterion presented in section 2.3.
- Axioms (4.1)-(4.11).
- A set T_{occ} of occurrence axioms. These axioms are meant to constrain the interpretations of the actual line of situations. First, we consider the case in which T_{occ} includes axioms that directly constrain the the structure of the actual line of situations. These are of the form:

$$(\exists\, s_1, \ldots, s_n).occurs(A_1, s_1) \wedge \ldots \wedge occurs(A_n, s_n) \wedge \mathcal{O}_<(s_1, s_2, \ldots, s_n). \tag{4.21}$$

  where $\mathcal{O}_<$ is an ordering formula. Later we extend the classes of formulas contained in T_{occ}.

The circumscription policy is as follows[3]:

$$Circ(\Sigma_{occ}; occurs_{\mathcal{T}}; actual, start), \tag{4.22}$$

Thus, we select the models of Σ_{occ} that have a minimal extension for the predicate $occurs_{\mathcal{T}}$. Below, we present a precise semantic characterization of (4.22). The predicate *actual* and the function

[3]A general discussion about circumscription can be found in appendix B.

start are variable elements in the circumscription. Clearly, these elements need to vary since $occurs_{\mathcal{T}}$ is defined in terms of them. Notice that the situation tree is fixed except for the elements that determine what the actual line looks like. As mentioned before, an interesting property of this policy is that if we know all the actions that occur, and if it is consistent to believe that they are all that occurred, then the circumscription will select those models in which the actual line of situations contain all and only those actions.

A remarkable feature of this circumscription is that it makes the choice of the actual path of situations depend only on the time line. That is, the minimization of the actions does not directly depend on the situations in which the actions are performed.

In the examples that follow, we make use of model-theoretic arguments to justify our conclusions. Therefore, based on Lifschitz's results [26], we present the model-theoretic meaning of the above circumscription.

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two arbitrary models of a theory of action Σ_{occ}. Model $\mathcal{M}_1$ is preferred to model $\mathcal{M}_2$, written $\mathcal{M}_1 < \mathcal{M}_2$ iff:

1. $\|\mathcal{M}_1\| = \|\mathcal{M}_2\|$. That is, $\mathcal{M}_1$ and $\mathcal{M}_2$ share the same domain.
2. The interpretation for every function symbol other than *start*, and the interpretation of every predicate other than *actual* and $occurs_{\mathcal{T}}$ is the same.
3. $occurs_{\mathcal{T}}^{\mathcal{M}_1} \subset occurs_{\mathcal{T}}^{\mathcal{M}_2}$. That is, the extension of the predicate $occurs_{\mathcal{T}}$ in $\mathcal{M}_1$ is a proper subset of the extension of $occurs_{\mathcal{T}}$ in $\mathcal{M}_2$.

A model $\mathcal{M}$ of Σ_{occ} is minimal with respect to $<$, if there is no other model $\mathcal{M}'$ of Σ_{occ} such that $\mathcal{M}' < \mathcal{M}$. Finally, from Lifschitz result, it follows that $\mathcal{M}$ is a model of

$$Circ(\Sigma_{occ}; occurs_{\mathcal{T}}; actual, start)$$

if and only if $\mathcal{M}$ is a minimal model of Σ_{occ} according to $<$ as defined previously.

In what follows we introduce theories that contain skolemized versions of axioms of the form (4.21). In these cases, and as justified by theorem B.2.1 in appendix B, we replace (4.22) with:

$$Circ(\Sigma_{occ}; occurs_{\mathcal{T}}; actual, start, \mathbf{S_k}). \tag{4.23}$$

Where $\mathbf{S_k}$ denotes the set of skolem constants introduced.

As mentioned before, we claim that if we have complete information about the actions that occurred then the circumscription policy gives the expected results. This is formally stated as follows:

Observation 4.1 *If Σ_{occ} is a theory in which the set T_{occ} contains a single axiom of the form:*

$$occurs(A_1, S_1) \wedge \ldots \wedge occurs(A_n, S_n) \wedge S_1 < \ldots < S_n. \tag{4.24}$$

and the theory Σ_{occ} has a model $\mathcal{M}$ that satisfies:

$$occurs(a, s) \equiv a = A_1 \wedge s = S_1 \vee a = A_2 \wedge s = S_2 \vee \ldots \vee a = A_n \wedge s = S_n. \tag{4.25}$$

Then, every model of $Circ(\Sigma_{occ}; occurs_{\mathcal{T}}; actual, start, \mathbf{S_k})$ will satisfy (4.25) and will also satisfy

$$S_n = do(A_{n-1}, S_{n-1}) \wedge \ldots \wedge S_2 = do(A_1, S_1) \wedge S_1 = S_0. \tag{4.26}$$

The importance of this observation is that it establishes that if it is consistent to assume that all the actions that are known to occur are the only ones that occur, and these actions are totally ordered, then the models of the circumscription will not include any other occurrences.

Furthermore:

Observation 4.2 *Let Σ_{occ} be a theory in which T_{occ} contains a single sentence of the form:*

$$occurs(A_1, S_1) \wedge \ldots \wedge occurs(A_n, S_n) \wedge \mathcal{O}_<(S_1, S_2, \ldots, S_n).$$

Where $\mathcal{O}_<$ is an ordering formula. Assume that there is a model $\mathcal{M}$ of Σ_{occ} that satisfies:

$$occurs(a, s) \equiv a = A_1 \wedge s = S_1 \vee a = A_2 \wedge s = S_2 \vee \ldots \vee a = A_n \wedge s = S_n \tag{4.27}$$

Then, $\mathcal{M}$ is a model of:

$$Circ(\Sigma_{occ}; occurs_{\mathcal{T}}; actual, start, \mathbf{S_k}). \tag{4.28}$$

However, it is not necessarily the case that for any model $\mathcal{M}'$ of (4.28) $\mathcal{M}'$ satisfies (4.27).

This property of our minimization policy is extremely important. In fact, it serves to illustrate the fact that it does not necessarily correspond to the intuition that *if it is consistent to assume that no occurrences other than the known ones exist, then there are no other occurrences.* As an example, consider a situation calculus language with action constants A, B and C, fluent F_c and a theory Σ_{occ} with the following axioms:

- Axioms about *Poss*:

$$Poss(B, s) \equiv holds(F_c, s)$$
$$Poss(A, s) \wedge Poss(C, s).$$

  Thus, B is possible only if F_c is true. A and C are always possible.

- Successor state axioms:

$$Poss(a, s) \supset [holds(F_c, do(a, s)) \equiv (holds(F_c, s) \wedge \neg a = A)]. \tag{4.29}$$

  Thus, F_c is falsified by action A and is otherwise unaffected.

- Initial state:

$$holds(F_c, S_0).$$

- T_{occ}:

$$occurs(A, S_1) \wedge occurs(B, S_2).$$

  Thus, we know that A and B have occurred, but we don't know in which order. Here, S_1 and S_2 are skolem constants.

Clearly, $\Sigma_{occ} \models S_2 < S_1$. Thus, any preferred model will satisfy this ordering.

Now, let us extend the language with a new action C. Also, let us change the example by replacing the successor state axiom (4.29) with:

$$Poss(a, s) \supset [holds(F_c, do(a, s)) \equiv a = C \vee (holds(F_c, s) \wedge \neg a = A)]. \tag{4.30}$$

Thus, C changes F_c positively. In this situation, there are models of Σ_{occ} that satisfy the ordering $S_1 < S_2$. Therefore, the previous proposition establishes that there will be models of the circumscription in which A occurred before B. It is not difficult to see that in these models C occurs in between. Thus, this is an example in which it is not the case that the circumscriptive theory entails that the known actions are all the actions that occur, even if it is consistent to assume so. Notice that there is a model in which $S_2 < S_1$ and C does not occur in it.

In our view, it would be incorrect to force an ordering among actions to ensure that fewer actions occur. For example, assume that we are told that: *John and Mary will meet at their place at some time. If Mary gets there before John, then she will call Sue.* If this is all that is known we do not want to infer that John will get home first just to enforce a minimization of actions (disallowing models in which Mary makes her call).

In the previous discussion we have shown that the preferential policy that we presented gives the correct answers when the set T_{occ} of occurrence axioms are of the form (4.24). Given proposition 4.5, this is also the case when T_{occ} is a sentence of the form:

$$(\exists\, t_1, \ldots, t_n).occurs_{\mathcal{T}}(A_1, t_1) \wedge \ldots \wedge occurs_{\mathcal{T}}(A_n, t_n) \wedge \mathcal{O}_{<}(t_1, \ldots, t_n).$$

when $\mathcal{O}_{<}$ provides a complete order. Unfortunately, sentences of this form are too restrictive and do not provide the expressiveness required by many examples (as shown below). Thus, we would like to be able to introduce a more general kind of occurrence statements in the set T_{occ}. Now, given such a set of sentences, we need some argument to the effect that the preferential policy presented gives the right answers. The only argument that we provide is the fact that the minimization policy selects those models which seem *intuitively* better in a set of examples. Unfortunately, this is unsatisfactory. A completely satisfactory solution to this problem would be to present some *independent justification* for the policy utilized. However, *independent justifications* are hard to come by. Thus, as other researchers in theories of action, we must use examples that would provide *evidence* for the correctness of the solution proposed. This we do in the next section.

Aside from axioms of the form (4.21), we will use axioms stating conditional occurrences. This axioms are of the form:

$$\Phi(s, a) \supset (\exists\, s').s \leq s' \wedge occurs(a, s'), \tag{4.31}$$

where $\Phi(s, a)$ is a simple state formula. Conditional occurrence axioms of this form provide further constraints on the actual line of occurrences. These axioms can also be written by utilizing situation free sentences that contain time terms. For example:

$$holds(Hungry, t) \supset (\exists\, t').t < t' \wedge occurs_{\mathcal{T}}(Eat, t')$$

states that if in the line of actual situations there is a time in which the fluent constant $Hungry$ holds, then there must be an action Eat in the actual line after the time at which $Hungry$ holds.

Finally, we also use axioms of the form:

$$(\exists\, t_1, \ldots, t_n).occurs_{\mathcal{T}}(A_1, t_1) \wedge \ldots \wedge occurs_{\mathcal{T}}(A_n, t_n) \wedge \mathcal{O}'(t_1, \ldots, t_n).$$

In which $\mathcal{O}'(t_1, \ldots, t_n)$ is an extended ordering formula that mentions temporal constants aside from the terms $t_1, \ldots, t_n$.

In the next section we apply our approach to several examples from the literature. In particular, we show that the intended conclusions for the YSP example above are indeed what follows from the circumscription.

4.4 Examples.

In this section we discuss several examples. These examples are taken from the literature in temporal reasoning in Artificial Intelligence. Most of these are used as benchmarks to test various theoretical proposals for reasoning about actions and change, and were compiled by Sandewall in [53]. Each problem is formalized as a circumscriptive theory (4.23) and differs in the domain axioms that are specified with each example.

4.4.1 The YSP.

The Yale Shooting problem [21] was presented in the previous section. The relevant axioms we repeat here in a skolemized form:

$$Poss(Load, s) \wedge Poss(Shoot, s), \tag{4.32}$$

$$Poss(Unload, s) \equiv holds(Loaded, s), \tag{4.33}$$

$$holds(Alive, S_0), \tag{4.34}$$

$$occurs(Load, S_1) \wedge occurs(Wait, S_2) \wedge occurs(Shoot, S_3), \tag{4.35}$$

$$S_1 < S_2 < S_3 < S_4, \tag{4.36}$$

$$actual(S_4), \tag{4.37}$$

$$\begin{aligned} Poss(a, s) \supset [&holds(Loaded, do(a, s)) \equiv \\ &a = Load \vee \neg a = Unload \wedge holds(Loaded, s)], \end{aligned} \tag{4.38}$$

$$\begin{aligned} Poss(a, s) \supset [&holds(Alive, do(a, s)) \equiv \\ &\neg(a = Shoot \wedge holds(Loaded, s)) \wedge holds(Alive, s)]. \end{aligned} \tag{4.39}$$

The objective of the *YSP* is to determine whether or not *Alive* holds after the shooting. First, it is not difficult to see that there is a model $\mathcal{M}_{int}$ of the circumscription such that:

$$\begin{aligned} \mathcal{M}_{int} \models S_1 = S_0 \wedge S_2 = do(Load, S_1) \wedge \\ S_3 = do(Wait, S_2) \wedge S_4 = do(Shoot, S_3). \end{aligned}$$

Furthermore,

$$\mathcal{M}_{int} \models \neg holds(Alive, S_4).$$

This model is depicted by figure 4.6. However, we need to show that there is no model of the

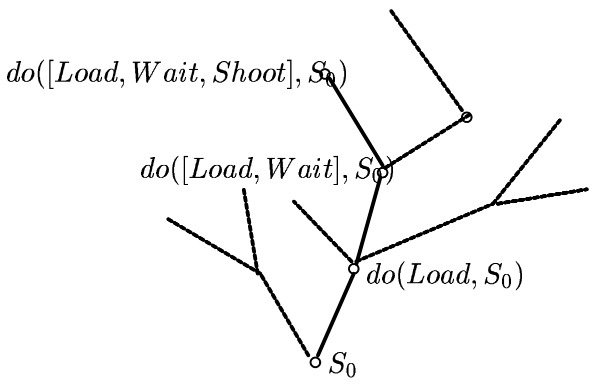


Figure 4.6: The situation tree in an intended model for the *YSP*.

circumscriptive theory that satisfies $holds(Alive, S_4)$. We do so by contradiction. That is, assume that there is a model $\mathcal{M}$ such that

$$\mathcal{M} \models holds(Alive, S_4)$$

is true.

From the *occurs* and ordering statements (4.32)-(4.32), we infer that $\mathcal{M}$ satisfies:

$$(\exists\, t_1, t_2, t_3).occurs_{\mathcal{T}}(Load, t_1) \wedge occurs_{\mathcal{T}}(Wait, t_2) \wedge occurs_{\mathcal{T}}(Shoot, t_3) \wedge \\ t_1 < t_2 < t_3.$$

By induction and using the successor state axiom for *Alive* (4.32), it follows that $\mathcal{M}$ satisfies:

$$(\forall\, s, s').s' > s \wedge \neg holds(Alive, s) \supset \neg holds(Alive, s').$$

That is, no resuscitations are possible. Therefore, if *Alive* holds in S_4, it follows that *Alive* holds in all previous situations. In particular, it follows that $holds(Alive, do(Shoot, S_3))$. Therefore, the successor state axiom for Alive tells us that $holds(Loaded, S_3)$ must be false. Using the successor state axiom for *Loaded* it follows that an *Unload* action must have occurred between the *Load* and the *Shoot*. So, we obtain:

$$\mathcal{M} \models (\exists\, t_1, t_2, t_3, t_4).occurs_{\mathcal{T}}(Load, t_1) \wedge occurs_{\mathcal{T}}(Wait, t_2) \wedge \\ occurs_{\mathcal{T}}(Shoot, t_3) \wedge occurs_{\mathcal{T}}(Unload, t_4) \wedge \\ t_1 < t_2 < t_3 \wedge t_1 < t_4 < t_3.$$

Thus, in $\mathcal{M}$, *Unload* occurred some time between the *Load* and the *Shoot*. $\mathcal{M}$ is depicted by figure 4.7. However, from this model, we can easily obtain a model $\mathcal{M}'_{int}$ (e.g., the one depicted by figure 4.6) without the *Unload* occurrence and that preserves the rest of the interpretation. Essentially, we only change the actual line of situations from one branch to another. Clearly, $\mathcal{M}'_{int} < \mathcal{M}$,

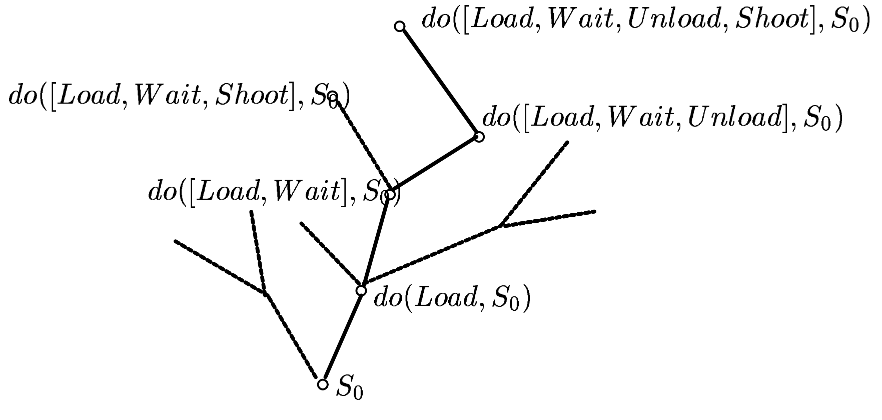

$do([Load, Wait, Unload, Shoot], S_0)$
$do([Load, Wait, Shoot], S_0)$
$do([Load, Wait, Unload], S_0)$
$do([Load, Wait], S_0)$
$do(Load, S_0)$
S_0

Figure 4.7: The situation tree in a non-minimal model for the *YSP*

therefore, $\mathcal{M}$ cannot be a model of (4.23). It follows that in all the models of the circumscription, *Alive* does not hold in S_4.

4.4.2 The Hiding Turkey.

The Hiding Turkey Problem is a simple variation of the *YSP* and was proposed by Sandewall [52] as a test for theories whose initial state is incompletely specified. The variation is that the *turkey* that is being shot may or may not be deaf. If the turkey is not deaf, then it will go on hiding if it hears the loading of the gun. There are many ways in which this problem can be formalized. A simple formulation is as follows:

$$Poss(Load, s) \wedge Poss(Shoot, s), \tag{4.40}$$
$$Poss(Unload, s) \equiv holds(Loaded, s), \tag{4.41}$$
$$Poss(a, s) \wedge a = Load \supset holds(Loaded, do(a, s)), \tag{4.42}$$
$$Poss(a, s) \wedge a = Load \wedge \neg holds(Deaf, s) \supset holds(Hiding, do(a, s)), \tag{4.43}$$
$$Poss(a, s) \wedge a = Shoot \wedge holds(Loaded, s) \wedge \neg holds(Hiding, s) \supset \neg holds(Alive, do(a, s)), \tag{4.44}$$
$$Poss(a, s) \wedge a = Unload \supset \neg holds(Loaded, do(a, s)), \tag{4.45}$$
$$holds(Alive, S_0), \tag{4.46}$$
$$\neg holds(Hiding, S_0), \tag{4.47}$$
$$S_0 \leq S_1 < S_2 < S_3 < S_4, \tag{4.48}$$
$$occurs(Load, S_1) \wedge occurs(Wait, S_2) \wedge occurs(Shoot, S_3), \tag{4.49}$$
$$actual(S_4). \tag{4.50}$$

A more accurate formalization would include *Hide* actions and would have these actions occurring after noisy situations. However, doing so adds complexity that would distract us from the main purpose of the example. First, we have added the fluents *Hiding* and *Deaf* to the language of the *YSP* example. Axiom (4.40) says that if the turkey is not deaf and *Load* is performed, the turkey will go under. Axiom (4.40) says that the turkey is not alive after shooting a loaded gun when the turkey is not hiding. Given that there are no effect axioms for *Deaf*, it follows that the successor state axiom for *Deaf* is simply:

$$Poss(a, s) \supset holds(Deaf, do(a, s)) \equiv holds(Deaf, s).$$

So *Deaf*ness is a property that always persists. Alternatively, *Deaf* could be simply a state independent atom instead of a fluent. For *Hiding*, we obtain:

$$Poss(a, s) \supset [holds(Hiding, do(a, s)) \equiv a = Load \wedge \neg holds(Deaf, s) \vee holds(Hiding, s)].$$

In this problem, the goal is to show that either the turkey is deaf and dead or non-deaf and alive after the shooting. Indeed, this is trivially the case. First, notice that *holds* is one of the fixed predicates in the circumscription. Therefore, there are two classes of minimal models, one class in which $holds(Deaf, S_0)$ is true, and one in which $\neg holds(Deaf, S_0)$. If the former holds, then the problem is identical to the *YSP*. Otherwise, it is obvious that the occurrence of *Load* will always make the turkey go hiding. Therefore, in the second class of models the turkey remains *Alive* and *Hiding*.

4.4.3 The Stanford Murder Mystery.

This is yet another variation of the *YSP*. The difference is that the occurrence of the *Load* event is eliminated. Therefore, there are only two occurrences explicitly stated, the *Wait* and the *Shoot*. Also, the literal:

$$\neg holds(Alive, S_4)$$

is added to the axiomatization. The problem is to determine whether or not the gun was loaded in S_0.

Interestingly, there are two possible solutions to this problem. The first one corresponds to minimal models in which $\neg holds(Loaded, S_0)$ is true. In this class of models some *Load* event must occur some time before the *Shoot*. For example, there will be models in which *Wait*, *Load*, and *Shoot* occur in sequence. Here, the *Load* action is necessary to explain $\neg holds(Alive, S_4)$.

The second solution corresponds to a different class of models, in which $holds(Loaded, S_0)$ is true. In this case, no *Load* event is postulated.

Thus, our preferential semantics does not tell anything about the state of the gun in S_0. Interestingly, according to the proponents of this example, the intended interpretation is such that $holds(Loaded, S_0)$ must be true. While it is unclear whether this should be the *intuitively correct* solution, it is interesting to note that this effect can be obtained by allowing for the predicate *holds* to vary. If we were to adopt this policy, then the second solution would be preferred to the first one. Nevertheless, we argue that both types of models for this problem should be "intended."

4.4.4 Ferryboat Connection Scenario.

This example was first proposed in [52]. It is as follows:

> "A motorcycle is initially driving along a road on island Fyen, in the direction of a ferryboat landing. The ferryboat departs at time 100. If the motorcycle reaches the ferryboat landing before time 100, it will be in Jutland from time 110, otherwise it stays. The motorcycle is known to reach the landing sometime between time 99 and 101."

This problem serves as a test to determine how well our language allows to express temporal relations between events. We extend the language with a fluent function *At* with one argument. We fix the domain objects as those denoted by *Fyen, Landing, Boat* and *Jutland*, which we assume to be distinct. Aside from the four primitive fluents $At(Fyen)$, $At(Landing)$, $At(Boat)$ and $At(Jutland)$, we introduce the fluents *Gone* and *Arrived.* The fluent *Gone* holds as a consequence of the departure of the ferry. The fluent *Arrived* holds after the ferryboat arrives in Jutland. Also, we introduce a two-place action function *move*, such that $move(d_1, d_2)$ would denote the action of moving from location d_1 to location d_2. Other actions are denoted by the constants *Depart* and *Arrive.* The axioms are as follows:

- T_{sc}:

$$d_1 \neq d_2 \supset \neg holds(At(d_1), s) \vee \neg holds(At(d_2), s). \tag{4.51}$$

- T_{pos}:

$$Poss(move(d_1, d_2), s) \equiv holds(At(d_1), s) \wedge \tag{4.52}$$
$$[d_1 = Boat \supset holds(Arrived, s)] \wedge [d_2 = Boat \supset \neg holds(Gone, s)],$$
$$Poss(Arrive, s) \equiv holds(Gone, s) \wedge \neg holds(Arrived, s), \tag{4.53}$$
$$Poss(Depart, s) \equiv \neg holds(Gone, s). \tag{4.54}$$

- T_{ef}:

$$Poss(Arrive, s) \supset holds(Arrived, do(Arrive, s)), \tag{4.55}$$
$$Poss(Depart, s) \supset holds(Gone, do(Depart, s)), \tag{4.56}$$
$$Poss(move(d_1, d_2), s) \supset holds(At(d_2), do(move(d_1, d_2), s)). \tag{4.57}$$

- T_{S_0}:
$$holds(At(Fyen), S_0) \wedge \neg holds(Gone, S_0) \wedge \neg holds(Arrived, S_0). \tag{4.58}$$

- T_{occ}:

$$occurs_{\mathcal{T}}(move(Fyen, Landing), T_1) \wedge 99 \leq T_1 \leq 101, \tag{4.59}$$
$$occurs_{\mathcal{T}}(Depart, 100), \tag{4.60}$$
$$occurs_{\mathcal{T}}(Arrive, 110), \tag{4.61}$$
$$\begin{aligned} &holds_{\mathcal{T}}(At(Landing), t) \wedge t < 100 \supset \\ &\qquad (\exists\, t').occurs_{\mathcal{T}}(move(Landing, Boat), t') \wedge t < t' < 100, \end{aligned} \tag{4.62}$$
$$\begin{aligned} &holds_{\mathcal{T}}(Arrived, t) \wedge holds_{\mathcal{T}}(At(Boat), t) \supset \\ &\qquad (\exists\, t').t \leq t' \wedge occurs_{\mathcal{T}}(move(Boat, Jutland), t'). \end{aligned} \tag{4.63}$$

State constraint (4.51) asserts that the motorcycle cannot be in two places simultaneously. Following the procedure indicated in section 3.1, the effect axiom:

$$Poss(move(d_1, d_2), s) \wedge d_1 \neq d_2 \supset \neg holds(At(d_1), do(move(d_1, d_2), s)), \tag{4.64}$$

can be generated from this state constraint and the effect axiom (4.55). Therefore, we obtain the following successor state axiom:

$$\begin{aligned} &Poss(a, s) \supset holds(At(d), do(a, s)) \equiv \\ &\qquad a = move(d_1, d\ ) \vee \neg(d_1 \neq d\ \wedge a = move(d, d_1)) \wedge holds(At(d), s), \end{aligned} \tag{4.65}$$

which means that the motorcycle is at d after moving to d, or if it is at d and did not move elsewhere. The successor state axioms for *Gone* and *Arrived* are:

$$Poss(a, s) \supset holds(Arrived, do(a, s)) \equiv a = Arrive \vee holds(Arrived, s), \tag{4.66}$$
$$Poss(a, s) \supset holds(Gone, do(a, s)) \equiv a = Depart \vee holds(Gone, s). \tag{4.67}$$

Axiom (4.52) says that to move anywhere from d_1 the motorcycle has to be at d_1, and if the motorcycle is in the *Boat* then the move is only possible if the *Boat* has *Arrived.* Also, if the motorcycle moves to the *Boat*, then the *Boat* must not be *Gone*. Axiom (4.59) says that the motorcycle will board the boat if it is at *Landing* before the boat departs. Axiom (4.59) says that the motorcycle leaves the boat after the boat has arrived. Axioms (4.59) and (4.59) state that *Depart* and *Arrive* occur at times 100 and 110 respectively. Notice that *Depart* and *Arrive* can each occur only once. The other axioms are self explanatory.

Using our framework, we obtain two classes of minimal models. Each class is characterized by the sets of situations that are actual in each. In order to abbreviate the sentences that we will write below, we extend the language with a new set of situation constants S_1, S_2, S_3, S_4, S_5, S_1', S_2', and S_3'. We fix the interpretation of these situation constants[4]:

$$S_1 = do(move(Fyen, Landing), S_0), \tag{4.68}$$
$$S_2 = do(move(Landing, Boat), S_1), \tag{4.69}$$

[4]These constants are introduced as abbreviations to avoid writing very long sentences that involve *do* terms with many other nested *do* terms. Since the interpretation for these constants is fixed, its participation in the circumscription policy as fixed or variable is of no consequence.

$$S_3 = do(Depart, S_2), \tag{4.70}$$
$$S_4 = do(Arrive, S_3), \tag{4.71}$$
$$S_5 = do(move(Boat, Jutland), S_4), \tag{4.72}$$
$$S'_1 = do(Depart, S_0), \tag{4.73}$$
$$S'_2 = do(move(Fyen, Landing), S'_1), \tag{4.74}$$
$$S'_3 = do(Arrive, S'_2). \tag{4.75}$$

As we just said, there are two classes of models. In the first class of models, we have:

$$actual(s) \equiv s = S_0 \vee s = S_1 \vee s = S_2 \vee s = S_3 \vee s = S_4 \vee s = S_5, \tag{4.76}$$
$$start(S_1) < start(S_2) < start(S_3) < start(S_4) < start(S_5). \tag{4.77}$$

Whereas the second class of models is characterized by:

$$actual(s) \equiv s = S_0 \vee s = S'_1 \vee s = S'_2 \vee s = S'_3, \tag{4.78}$$
$$start(S'_1) < start(S'_2) < start(S'_3). \tag{4.79}$$

Obviously the ordering (4.76) and (4.78) of the temporal constants is derivable from the equality relationships defined with the axioms (4.68)-(4.68).

The objective of this example is to show that the circumscriptive theory characterized by (4.23) entails that either the first set of sentences (4.76)-(4.76) is true, or the second set of sentences (4.78)-(4.78) is true. We do so by showing that there are minimal models that satisfy each set of sentences. Then we show that any model that does not satisfy either set of sentences cannot be minimal.

Observation 4.3 *There exists a minimal model $\mathcal{M}$ that satisfies sentences (4.76)-(4.76).*

$\mathcal{M}$ may be obtained by setting $T_1 = start(S_1) = 99$, $start(S_2) = 99.5$, $start(S_3) = 100$, $start(S_4) = 110$, $start(S_5) = 111$. Clearly, there is a model in which the actual line is such that these temporal constrains are satisfied. This model is minimal with respect to the circumscription policy. In fact, if we eliminate any element from the extension of $occurs_{\mathcal{T}}$ we violate one of the axioms.

Observation 4.4 *There exists a minimal model $\mathcal{M}'$ that satisfies sentences (4.78)-(4.78).*

For the second class of models, we set $start(S'_1) = 100$, $start(S'_2) = T_1 = 101$, $start(S'_3) = 110$. Again, it is easy to check that a model $\mathcal{M}'$ that satisfies this constraints exists. Also, this is a minimal model, since eliminating any one of the elements of the extension of $occurs_{\mathcal{T}}$ leads to a non-model.

Finally:

Observation 4.5 *Any minimal model must satisfy either (4.76)-(4.76) or (4.78)-(4.78).*

This can be easily checked by realizing that all models include the occurrences of *Depart* and *move(Fyen, Landing)*. The only ambiguity with respect to these events is which occurs first. There are only two possibilities (we assume that they may not occur concurrently). From each possibility, a different class of models arise. First, assume that $\mathcal{M}$ is an arbitrary model in which *move(Fyen, Landing)* occurs before *Depart*. Thus, in this model $T_1 < 100$, given that the structure of time is dense and from proposition 4.2 it follows that this model satisfies:

$$(\exists\, s, t)\; occurs(move(Fyen, Landing), s) \wedge start(do(move(Fyen, Landing), s)) = T_1 \wedge \\ during(t, s) \wedge T_1 < t < 100. \tag{4.80}$$

Therefore, the left hand side of (4.59) is satisfied. Hence, the motorcycle moves to the *Boat*. Once in the *Boat*, no *move* actions are possible until the ferryboat has arrived, as inferred from (4.52). Thus, (4.76)-(4.76) must hold in the minimal models arising from this case. Second, it is simple to check that if $\mathcal{M}'$ is a model in which $move(Fyen, Landing)$ occurs after *Depart*, then all the occurrences in the second class of models must arise. Hence, it is easy to see that the minimal models arising from this class must satisfy (4.78)-(4.78).

It is important to point out that the axiomatization of the problem is extremely simplified. For example, in a more general axiomatization, the *Poss* axioms for *move* actions should refer to the spatial interrelationships between the origin and destination. Also, the axiomatization forces the behavior of the ferry and the motorcyclist. For example, after arriving at *Landing*, if the motorcyclist is on time she must board. However, in a more general case, there is no guarantee that the motorcyclist will indeed board the ferry once in *Landing*. In fact, without formalizing notions of agent's goals, desires, intentions, etc., we may not have any way to predict what will happen.

4.4.5 The Russian Turkey Scenario.

This is yet yet another variation of the *YSP*. This problem is essentially one in which there is uncertainty with respect to some event. In particular, the gun is a revolver that has a partially loaded cylinder which is spun. Depending on how far the spinning of the cylinder goes, a bullet may or may not end up aligned with the gun's barrel. Normally, this type of uncertainty is labeled as uncertain effects of a deterministic action. As such, Reiter's approach using successor state axioms is not viable. However, this problem may be seen in a slightly different manner. We can think of the spinning action as being a class of actions. One element of the class is *SpinOhNo* in which case the spinning leaves a bullet aligned with the barrel. Another element of the class is *SpinSigh*, in which no bullet is left aligned with the barrel. It can be argued that this is a more accurate axiomatization. In particular, uncertainty arises because of a lack of knowledge of all the physical parameters involved in the spinning process. Therefore, it is more accurate to say that we are uncertain of exactly which action was performed (i.e. the exact values of the spinning action parameters) than to say that we are uncertain about the effects of a deterministic action. The axiomatization that follows takes that point of view:

$$
\begin{aligned}
&Poss(SpinOhNo, s) \wedge Poss(SpinSigh, s),\\
&Poss(Load, s) \wedge Poss(Shoot, s),\\
&Poss(a, s) \supset [holds(Loaded, do(a, s)) \equiv a = Load \vee holds(Loaded, s)],\\
&Poss(a, s) \supset [holds(Aligned, do(a, s)) \equiv\\
&\qquad a = SpinOhNo \vee holds(Aligned, s) \wedge \neg a = SpinSigh],\\
&Poss(a, s) \supset [holds(Alive, do(a, s)) \equiv\\
&\qquad holds(Alive, s) \wedge \neg(a = Shoot \wedge holds(Loaded, s) \wedge holds(Aligned, s))],\\
&occurs(Load, S_1) \wedge occurs(Shoot, S_3),\\
&occurs(SpinOhNo, S_2) \vee occurs(SpinSigh, S_2).
\end{aligned}
$$

Thus, we are uncertain of what occurred in S_2. Obviously, there will be two classes of models. The first one in which *Load*, *SpinOhNo* and *Shoot* occur in sequence, and the second, in which *Load*, *SpinSigh* and *Shoot* occur in sequence. Each of these models is minimal (we cannot have a model with fewer occurring actions) and the effects of each action are given by the successor state axioms. So, in one class of axioms the shooting is done with the bullet aligned and the other with the bullet not aligned.

4.4.6 The Stolen Car Problem.

The stolen car problem was proposed by Kautz [22] to illustrate problems that arise when an incomplete set of events is provided in a reasoning task. The basic idea is that an individual parks a car in the street. Later, the person goes to get the car and finds that the car is no longer where she parked it. The intended conclusion is that the car has been stolen sometime after it was parked and before it was noticed missing. We change the problem slightly by extending the possible explanations of the car disappearance with a *TowAway* action.

Assume that the car is parked in the morning:

$$holds(Parked, S_m) \wedge actual(S_m).$$

The observation that some time later the car is no longer parked is written as:

$$actual(S_e) \wedge \neg holds(Parked, S_e) \wedge S_m < S_e.$$

Furthermore, assume that our theory includes the following successor state axiom (derived from the corresponding effect axioms):

$$\begin{aligned} &Poss(a,s) \supset holds(Parked, do(a,s)) \equiv \\ &\qquad [holds(Parked, s) \wedge a \neq Steal \wedge a \neq TowAway] \vee a = Park. \end{aligned}$$

Then, the following is a consequence of the axioms:

$$(\exists\, s).(S_m \leq s < S_e) \wedge [occurs(Steal, s) \vee occurs(TowAway, s)].$$

Again, as in the ferryboat example, there are two classes of minimal models. In the first class *Steal* occurred and in the second *TowAway* occurred. In both cases, S_m is identified with S_0. Nothing else occurs.

4.5 Modeling Sequential Circuits.

In this section we discuss the use of the extended situation calculus as a framework to model sequential circuits. This is an application that is interesting for two reasons. First, it deals with a real application domain in which the dynamic aspect of the world is an essential aspect. The availability of a formal framework to specify these circuits is important in order to analyze the dynamic behavior of different circuit designs. Second, it shows that the situation calculus is suitable for modeling knowledge in a domain in which the formal framework of choice seems to be some form of modal temporal logic [34]. Thus, it serves to illustrate the fact that the expressive power of the situation calculus makes it an alternative to modal logics of time.

To illustrate our approach, we first present a general theory used to model sequential circuits of a restricted class. The restrictions are introduced to simplify the presentation. In fact, the approach can be extended to deal with a more general class of circuits. Thus, we first introduce an extended language to represent sequential circuits. We show how we can model the behavior of individual components using definitions and effect axioms. Finally, we derive successor state axioms that describe how non-defined fluents change due to changes in the input to the circuit.

Later on, we present a simple circuit (a 1-bit counter with an unable signal) and show how to prove that the circuit's design is correct. Thus, we show that the predicted behavior corresponds to the intended behavior of the device. The intended behavior of the device is described with a sentence in the language. The design is correct if this sentence follows from the axiomatization that describes the circuit.

4.5.1 General Axiomatization.

A sequential circuit is built from a set of *components* which are either *gates* or *flip-flops*. Each one of these components is a *black box* which has a set of wires that are used to interconnect them. Each wire is either an *input* wire or an *output* wire. Furthermore, the circuit as a whole has certain wires that are considered to be *inputs* to the circuit. The circuit's output is identified with some of its component's output wires. At any given time, a wire is energized over some threshold or is de-energized. When the wire is energized, it is said to carry a positive signal, variously identified as *true*, *on* or 1. When the wire is de-energized it is said to carry a negative signal, identified as *false*, *off* or 0. There are several types of gates, we only consider gates that have a single output and one or two inputs. The output of a gate is a logical function of its inputs. The gates we consider are *not* (with one input), *and* and *or* (each with two inputs). There are several types of flip-flops, we only consider one of them, the D flip-flops (in the rest of this chapter, we refer to D flip-flops simply as flip-flops). A flip-flop has a single output and two inputs. One of the inputs is a *clock*. A *clock* is a signal that changes from *on* to *off* and vice-versa in a periodic fashion. The other input of the flip-flop is its *data* input. When the clock input of a flip-flop changes from *off* to *on*, the flip-flop reads its *data* input and sets the output at the same value as this input. If the clock does not change, then the output of the flip-flop does not change either. Thus, it is commonly said that a flip-flop *memorizes* the data input for an entire cycle of its clock. Another simplification that we make, is that we only consider circuits in which all the flip-flops are fed with external clocks.

If we consider that any sequential circuit may be in a finite number of possible states (defined as the value carried by each one of the wires in the circuit), we may show by induction that for any of these states and any possible action certain properties hold. For example, if we have the specifications for a given circuit, we may use this approach to prove the correctness of a design.

We subdivide the domain objects into two subdomains: $\mathcal{G}$ of components and $\mathcal{W}$ of wires. Furthermore, we subdivide the sort $\mathcal{G}$ into four independent subsorts: $\mathcal{G}_{and}$, $\mathcal{G}_{or}$, $\mathcal{G}_{not}$, $\mathcal{G}_{ff}$. Also, we need the wire functions $in : \mathcal{G} \times \{1,2\} \to \mathcal{W}$ and $out : \mathcal{G} \to \mathcal{W}$. These functions identify the input and output wires of a component. Thus, if g is a component then $out(g)$ denotes its output wire. Components have one or two *input* wires, the term $in(g,i)$ is used to denote the input wire number i of component g. Thus, a literal $holds(val(in(g,2)),s)$ would be true if the input numbered 2 of g was *on* in s. If a component does not have a wire numbered 2, then the denotation of the term $in(g,2)$ is irrelevant.

We also introduce the fluent function $val : \mathcal{W} \to \mathcal{F}$. The idea is that if w is a wire, then $holds(val(w),s)$ is true if and only if the wire w is *on*.

In what follows, we use the variable names g and w for components (sort $\mathcal{G}$) and wires (sort $\mathcal{W}$). Also, variables g_a, g_o, g_n, and g_f, are used for subsorts $\mathcal{G}_{and}$, $\mathcal{G}_{or}$, $\mathcal{G}_{not}$, and $\mathcal{G}_{ff}$. As usual, we use these names with other subscripts and markers as variables of the same types. Upper case is used for constants. The wires that do not correspond to inputs or outputs of gates correspond to the circuit's inputs. The actions in these domain are toggling actions. Thus, we introduce the function symbol *toggle* : $\mathcal{W} \rightarrow \mathcal{A}$. Hence, *toggle* takes an element of the sort $\mathcal{W}$ of wires and yields an element of the sort $\mathcal{A}$ of actions. Also, we will use the predicate *input* $\subseteq \mathcal{W}$ to identify the circuit's input wires.

Depending on the type of component, there is a different relationship between the inputs and the output of the component. A very important distinction must be made between the behavior of gates and flip-flops. The output value of a gate[5] in a situation s is determined by the values of its input wires in s. In contrast, the output value of a flip-flop in a state s depends on the value of its inputs in the previous state (except for the initial state S_0, where the output value of the flip-flops is given as part of the description of the initial state). The following axioms describe the relationships between inputs and outputs of gates:

$$holds(val(out(g_{and})), s) \equiv holds(val(in(g_{and}, 1)), s) \wedge holds(val(in(g_{and}, 2)), s), \quad (4.81)$$
$$holds(val(out(g_{or})), s) \equiv holds(val(in(g_{or}, 1)), s) \vee holds(val(in(g_{or}, 2)), s), \quad (4.82)$$
$$holds(val(out(g_{not})), s) \equiv \neg holds(val(in(g_{not}, 1)), s). \quad (4.83)$$

Thus, we have simple definitions for the values of the output connectors in terms of the input connectors. Notice that an expression of this type cannot be used to describe the behavior of a flip-flop. As mentioned before, the reason is that the output value of a flip-flop in a situation s is given by the state of the circuit in the situation previous to s and the action that led to s. Later, we will describe the behavior of a flip-flop using effect axioms.

In order to describe the connections among components of a circuit, we use equality literals. For example, the literal:

$$in(G_a, 1) = out(G_f). \quad (4.84)$$

states that the first input of G_a is connected to the output of G_f. An important constraint is that the outputs of different components must not be connected. Thus, we need:

$$(\forall\, g_1, g_2)\ out(g_1) = out(g_2) \supset g_1 = g_2. \quad (4.85)$$

If this constraint were not satisfied, the circuit would be anomalous. In fact, this corresponds to a shorted circuit. A shorted circuit would, in most cases, lead to a contradictory theory.

The wires that correspond to the circuit's input can be characterized as follow:

$$input(w) \equiv (\forall\, g)\ \neg\ w = out(g).$$

Furthermore, we require that toggling actions be performed only on input wires, thus:

$$Poss(toggle(w), s) \equiv input(w).$$

Interestingly, we can view theories that model circuits of the type presented here as theories of the type studied in chapter 3. In the theories designed for the modeling of digital circuits, the *primitive* fluents are all the fluents that denote values of wires that are inputs to the circuit or that are outputs of flip-flops. In the rest of this chapter, wires that are inputs to the circuit or outputs of flip-flops will be called *primitive wires*. The behavior of primitive fluents is described by effect axioms. In contrast, non-primitive fluents correspond to fluents that denote the value of wires that are the output of gates.

[5]The output or input value of a gate refers to the value carried by the corresponding wire.

Problems arise if the circuits are not properly constructed. For example, if the output of a *not* gate is connected to its input. To eliminate these type of anomalies, we should precisely axiomatize the allowable circuit topologies. For simplicity, we leave this distinction at a metatheoretical level. We consider that a circuit is anomalous if and only if there exists a loop in it that does not go through a flip-flop[6], or if two outputs are connected together. In our theory, we do not deal with anomalous circuits.

The only actions that we have in this domain are the toggling of the wires that are circuit's inputs. Obviously, one of the effects of toggling an input wire is to modify the value of the wire. This is modeled with the following effect axioms:

$$\begin{aligned} &Poss(toggle(w), s) \wedge holds(val(w), s) \supset \\ &\qquad \neg holds(val(w), do(toggle(w), s)), \qquad (4.86)\\ &Poss(toggle(w), s) \wedge \neg holds(val(w), s) \supset \\ &\qquad holds(val(w), do(toggle(w), s)). \qquad (4.87)\end{aligned}$$

An important characteristic of *D*-flip-flops is that they have a *state* which changes when the clock changes from negative to positive. The *state* of a *D*-flip-flop corresponds to the state (value) of its output wire. Since this behavior is dynamic, and depends on a *tic* of the clock, it is better characterized using effect axioms. Thus, given the toggle of a wire, we have:

$$\begin{aligned} &Poss(toggle(w), s) \wedge \neg holds(val(w), s) \wedge holds(val(in(g_f, 1)), s) \wedge \\ &\qquad w = in(g_f, 2) \supset holds(val(out(g_f)), do(toggle(w), s)), \qquad (4.88)\\ &Poss(toggle(w), s) \wedge \neg holds(val(w), s) \wedge \neg holds(val(in(g_f, 1)), s) \wedge \\ &\qquad w = in(g_f, 2) \supset \neg holds(val(out(g_f)), do(toggle(w), s)). \qquad (4.89)\end{aligned}$$

The values of all non-primitive wires are given by the definitions (4.81)-(4.81). Given that we have identified a set of primitive fluents and given that non-primitive fluents are defined in terms of the primitive ones, we would like to utilize the results of chapter 3. Remember that for this to be possible, we need that the the right side of the effect axioms refer only to primitive fluents. Unfortunately, this is not the case. In fact, effect axioms (4.86)-(4.88) refer to the effects of *toggle* actions on fluent terms $val(w)$, where w is an unrestricted variable of sort $\mathcal{W}$. Thus, these effect axioms do not refer only to the effects of actions on primitive fluents. To deal with this difficulty we introduce a syntactic variant of the original theory that will conform to the requirement of having effect axioms refer only to primitive fluents. First, we introduce the following abbreviation:

$$prim(w) \equiv input(w) \vee (\exists g_f)\ w = out(g_f).$$

Thus, $prim(w)$ is true only if w is a primitive wire. We also introduce a new fluent function symbol $val' : \mathcal{W} \to \mathcal{F}$, such that:

$$prim(w) \supset holds(val'(w), s) \equiv holds(val(w), s). \qquad (4.90)$$

Thus, val' is the same as val for primitive wires. Now, the primitive fluents are the fluents val' and the val fluents are defined. The definition of val is derived from (4.81)-(4.81) and (4.90):

$$holds(val(w), s) \equiv \qquad (4.91)$$

[6]We can associate a digraph with any digital circuit. The digraph's nodes are the gates and the digraph's directed links are the circuit's connections (as specified by the equality literals, e.g. (4.84)). The directionality of the links is always towards the input of a gate. A loop is any cycle in the digraph.

$$\begin{aligned}
&[(\exists\, g_{and})[w = out(g_{and}) \wedge holds(val(in(g_{and},1)),s) \wedge holds(val(in(g_{and},2)),s)] \vee \\
&(\exists\, g_{or})[w = out(g_{or}) \wedge (holds(val(in(g_{or},1)),s) \vee holds(val(in(g_{or},2)),s))] \vee \\
&(\exists\, g_{not})[w = out(g_{not}) \wedge \neg holds(val(in(g_{not},1)),s)] \vee \\
&prim(w) \wedge holds(val'(w),s)].
\end{aligned}$$

Finally, we replace the effect axioms (4.86)-(4.88) with:

$$\begin{aligned}
&Poss(toggle(w),s) \wedge holds(val(w),s) \supset \\
&\qquad\qquad \neg holds(val'(w),do(toggle(w),s)), \\
&Poss(toggle(w),s) \wedge \neg holds(val(w),s) \supset \\
&\qquad\qquad holds(val'(w),do(toggle(w),s)), \\
&Poss(toggle(w),s) \wedge \neg holds(val(w),s) \wedge holds(val(in(g_f,1)),s) \wedge \\
&\qquad w = in(g_f,2) \supset holds(val'(out(g_f)),do(toggle(w),s)), \\
&Poss(toggle(w),s) \wedge \neg holds(val(w),s) \wedge \neg holds(val(in(g_f,1)),s) \wedge \\
&\qquad w = in(g_f,2) \supset \neg holds(val'(out(g_f)),do(toggle(w),s)).
\end{aligned}$$

From these effect axioms we derive the following successor state axiom:

$$\begin{aligned}
&Poss(a,s) \supset holds(val'(w),do(a,s)) \equiv \\
&\quad (a = toggle(w) \wedge \neg holds(val(w),s)) \vee \\
&\quad (\exists\, w').(a = toggle(w') \wedge w = out(g_f) \wedge w' = in(g_f,2) \wedge \\
&\qquad holds(val(in(g_f,1)),s) \wedge \neg holds(val(w'),s)) \vee \\
&\quad holds(val(w),s) \wedge [\neg a = toggle(w) \wedge \neg(w = out(g_f) \wedge a = toggle(w') \wedge \\
&\qquad\quad \neg holds(val(w'),s) \wedge \neg holds(val(in(g_f,1)),s) \wedge w' = in(g_f,2))].
\end{aligned} \tag{4.92}$$

Thus, we have derived a successor state axiom for the primitive fluents *val'*. This successor state axiom tells us what the values of the primitive wires will be as the result of some input. In order to determine the behavior of non-primitive wires, we make use of the definitions (4.81)-(4.81) and (4.90). To illustrate how this theory can be used to prove properties of circuits, in the next subsection we present a simple example.

4.5.2 An Example.

Consider the circuit of figure 4.8. In this circuit, the output corresponds to the wire $out(G_f)$. The circuit is a simple 1-bit counter with an unable signal (wire W_{I_a}). Thus the output changes once (from *on* to *off* or from *off* to *on*) for every complete cycle of the clock signal w. This behavior is disabled if the wire W_{I_a} is *off*, in which case, the output of the flip-flop is fixed. We use the language introduced in the previous subsection to describe the circuit and then show that this circuit exhibits the intended behavior. The intended behavior is described with a sentence in the same language, as shown later.

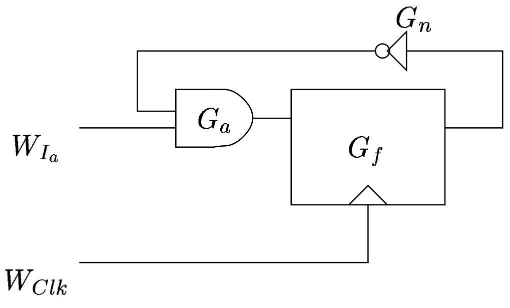


Figure 4.8: A Circuit.

In this circuit, there are two external input signals, W_{I_a} and W_{Clk}, along with three gates G_n and G_f and G_a. The circuit is described as follows:

$$\begin{aligned}
&in(G_a, 1) = W_{I_a},\\
&in(G_a, 2) = out(G_n),\\
&in(G_n, 1) = out(G_f),\\
&in(G_f, 1) = out(G_a),\\
&in(G_f, 2) = W_{Clk},\\
&\neg W_{I_a} = W_{Clk}.
\end{aligned}$$

In this example there is only one external clock (W_{Clk}). The behavior of this external clock may be modeled as:

$$actual(s) \supset occurs(toggle(W_{Clk}), s). \tag{4.93}$$

This axiom states that $toggle(W_{Clk})$ occurs repeatedly in an infinite sequence. Furthermore, it says that nothing else occurs. To see why, assume that $occurs(toggle(W_{I_a}), S)$ is true for some arbitrary situation S. From the definition of $occurs$ (4.7) it follows that:

$$occurs(toggle(W_{I_a}), S) \supset actual(do(toggle(W_{I_a}), S)).$$

It also follows that $actual(S)$. Thus, from (4.93) it follows that $occurs(toggle(W_{Clk}), S)$ as well as $actual(do(toggle(W_{Clk}), S))$, which contradicts (4.1), given that $\neg W_{I_a} = W_{Clk}$. With this axiomatization, it is easy to prove that the wire W_{I_a} never changes its value.

The initial situation is given by:

$$\neg holds(val(out(G_f)), S_0) \wedge \neg holds(val(W_{Clk}), S_0).$$

Thus, it is unknown whether $holds(val(W_{I_a}), S_0)$ is true. The flip-flop's state is initially *off* and so is the clock. The intended behavior of this device is that if W_{I_a} is *on*, then the output of the flip-flop is a clock signal with half the frequency of the input clock. When W_{I_a} is *off*, the output of the flip-flop is a steady *off* signal. Formally, we need to write a sentence that describes this behavior. We do so by cases:

- In the first case, we have the following:

Proposition 4.6 *From the theory Σ_{occ} that includes the axioms of sections (4.5.1) and (4.5.2) it follows that:*

$$\begin{aligned}
&holds(val(W_{I_a}), S_0) \wedge actual(s) \supset \\
&\qquad [ab(val(out(G_f)), toggle(W_{Clk}), s) \equiv \neg holds(val(W_{Clk}), s)].
\end{aligned} \tag{4.94}$$

Where the predicate ab is as defined in (2.20), that is:

$$ab(f, a, s) \equiv \neg[holds(f, do(a, s)) \equiv holds(f, s)].$$

Sentence (4.94) states that the output of the flip-flop changes (i.e., is abnormal) due to a toggle of the clock only when the clock is *off* at the time of the toggle. This necessarily implies that the rate of change of the flip-flop's output is half of the clock's rate. While this may not be immediately obvious, it is the most concise way to write this property without having to introduce other notions (as rate of change) in the language. To prove that this sentence holds, we simply assume that $val(W_{I_a})$ is on and use the induction axiom (2.1) with:

$$\varphi(s) = actual(s) \supset ab(val(out(G_f)), toggle(W_{Clk}), s) \equiv \neg holds(val(W_{Clk}), s).$$

The proof is fairly straightforward but very tedious. An essential element of the proof is achieved by showing that:

$$\begin{aligned}
&a = toggle(W_{Clk}) \wedge w = out(G_f) \supset \\
&\qquad holds(val(w), do(a, s)) \equiv \\
&\qquad\qquad [\neg holds(val(w), s) \wedge \neg holds(val(W_{Clk}), s))] \vee \\
&\qquad\qquad [holds(val(w), s) \wedge holds(val(W_{Clk}), s)].
\end{aligned}$$

which is derived from (4.92).

- The second case is much simpler and can be written as:

Proposition 4.7 *From the theory Σ_{occ} that includes the axioms of sections (4.5.1) and (4.5.2) it follows that:*

$$\neg holds(val(W_{I_a}), S_0) \supset actual(s) \supset \neg holds(val(out(G_f)), s).$$

This proposition states that if W_{I_a} is *off* in S_0 then the output of the flip flop is always *off*. This is also proven by induction with:

$$\varphi = actual(s) \supset \neg holds(val(out(G_f)), s).$$

The approach that we have presented here has many limitations. One of them is that we cannot deal with concurrent events. However, in the next chapter we propose an approach to accommodate concurrent actions in the language. This further extension to the situation calculus will allow the possibility of having input signals that change concurrently. On the other hand, other extensions are possible within the language of the situation calculus with occurrences. For example, in many circuits there are delays that retard the changes in the output of some components. These delays can be modeled introducing a new sort of *delay* component. Each delay component would have a single input wire and one output wire. Thus, if W_I and W_O denote the input and output of a delay component, we would write:

$$occurs_{\mathcal{T}}(toggle(W_I), t) \supset occurs_{\mathcal{T}}(toggle(W_O), t + \delta),$$

where δ is some positive real number. Of course, we need to extend the axiomatization to allow for the new actions and components. Furthermore, we would need to consider situations in which the input is toggled with a frequency that is higher than δ changes per unit time. There are many ways in which this can be resolved depending on the actual behavior of the delay component.

4.6 Other Features.

As we have seen throughout this chapter, there are many expressive enhancements derived from our extension to the situation calculus. Many of these enhancements have not been described with the previous examples. In this section, we briefly mention some of these.

An interesting kind of expressiveness that we gain by having an actual line in the situation calculus is the possibility to specify behavioral rules, for example: "Never cross the street against a red light." This can be expressed as:

$$holds(Red, s) \supset \neg occurs(Cross, s).$$

Another example is "If you drink, don't drive," formalized as:

$$\neg occurs(Drive, do(Drink, s)). \tag{4.95}$$

Notice that these constraints could be alternatively modeled as *preconditions* for actions. For instance, we could model the "If you drink, don't drive," as:

$$Poss(Drive, s) \equiv \neg holds(Drunk, s),$$

That is, if you are drunk you cannot possibly drive. However, this is in general too strong. In fact, by writing (4.95) instead, we allow for drinking on alternative branches (non-actual branches), and can fantasize about the consequences of drinking and driving. Using the latter strategy precludes us from doing so.

We also have a natural notion of prevention. To prevent F from ever becoming false:

$$actual(s) \supset holds(F, s).$$

Interestingly, this could also be modeled as:

$$holds(F, s),$$

which could be interpreted as a *precondition oriented state constraint* in the terminology of Lin and Reiter [31]. For example, if we have the effect axiom:

$$Poss(A, s) \wedge \gamma_F^-(A, s) \supset \neg holds(F, do(A, s)),$$

then we would obtain the precondition:

$$Poss(A, s) \supset \neg\gamma_F^-(A, s).$$

Again, modeling prevention in this manner precludes us from reasoning about non-actual lines in which F is made false. These issues are related to *deontic* notions like *obligations*, *commitments*, *permissions*, etc, which are beyond the scope of this work. However, it is possible to incorporate some simple forms of hypothetical reasoning by exploring non-actual lines of situations. Pursuing this idea is left for future research.

Another interesting aspect of our presentation is that we can model some simple notions of causality between events. That is, causal relations in which one event is the *cause* of another. For example, "shooting a loaded gun causes a noise to occur":

$$holds(Loaded, s) \wedge occurs(Shoot, s) \supset occurs(Noise, do(Shoot, s)).$$

Also, we may state that if the event of a ball colliding with the floor occurs, it must be the case that some event occurred prior to the collision. Furthermore, this event must have started the motion of the ball.

Chapter 5

Enriching the Ontology of Actions.

In this chapter we investigate how richer ontologies of action can be integrated with the notion of occurrences introduced in the previous chapter. We study two different ontological extensions. The first one is an extra-logical extension in which *complex* actions are introduced as elements that are built from simple primitive actions. The approach to complex events studied here is the result of research conducted in the *Cognitive Robotics* group in the University of Toronto's Department of Computer Science. The second extension we explore is the introduction of concurrent actions. This time, we propose to extend the language of the situation calculus with operators among actions that denote new concurrent actions.

5.1 Complex Actions.

Until recently, the problem of using *complex actions* within the situation calculus had not been addressed. Gelfond, Lifschitz and Rabinov [16], proposed an approach for a situation calculus in which situations form a *continuum.* In that context, they proposed a way to deal with complex actions. Unfortunately, their work is not directly applicable to a discrete situation calculus. On the other hand, Lin, Lespérance, Levesque, Reiter and Scherl propose a mechanism to integrate complex actions into the more standard discrete situation calculus [30] (in the future we refer to this as CR's approach, for *Cognitive Robotics*). Their work is based on Reiter's solution to the frame problem in the situation calculus. In this section, we study how to integrate the notion of an *actual line* in the situation calculus, with the notion of *complex events* based on CR's work. The main objective is to be able to write sentences referring to complex action occurrences. In what follows, we show how to integrate CR's approach with the notion of an actual line of situations.

5.1.1 Definition of Complex Actions.

So far, we have studied the situation calculus in which all actions are considered *primitive.* Thus, actions cannot be decomposed nor defined in terms of others. It is clear, however, that the notion of *complex action* is necessary to describe the activity of agents. For example, we may state complex actions such as: *clear the table*, which can be expressed as *while there is some item on the table, remove an item from the table.*

An interesting feature of CR's approach is that the proposal does not involve an extension of the language of the situation calculus. Instead, they propose a mechanism in which a set of extra-logical symbols, along with a set of translation rules are introduced. Therefore, the notion of complex action is not incorporated into the ontology of the situation calculus. An important advantage is that the solution to the frame problem for complex actions is obtained directly from the solution for the primitive actions. Clearly, the main disadvantage of this approach is that complex actions are not objects in the language. Therefore, we cannot prove any properties of the complex actions within the logic.

In what follows, we summarize CR's approach to complex actions. We have made certain changes to their presentation to accommodate reified fluents.

First, there is a number of extra-logical symbols: *Do*, and the complex action constructors: ?, ;, |, π, **if then else**, and **while**. These symbols are used in expressions which have to be translated into the logical language of the situation calculus. Complex actions are composed from other primitive or complex actions using the complex action constructors. The basic extra-logical expression is formed by using the *Do* symbol. In particular, if a is an action (complex or primitive), then the expression $Do(a, s, s')$ should be understood as saying that if a is done in situation s, then s' is one of the possible situations reached. Complex actions appear only as first argument to the *Do* expression. Therefore, we need translation rules for $Do(a, s, s')$ when a is either a primitive action or a complex action. The translation of *Do* is as follows:

1. If a is a primitive action:

$$Do(a, s, s') =_{def} Poss(a, s) \wedge s' = do(a, s).$$

2. Tests:

$$Do(f?, s, s') =_{def} holds(f, s) \wedge s = s',$$

   where f is a fluent term.

3. Sequence:

$$Do([a_1; a_2], s, s') =_{def} (\exists\, s'').Do(a_1, s, s'') \wedge Do(a_2, s'', s').$$

4. Non-deterministic choice between two actions:

$$Do([a_1|a_2], s, s') =_{def} Do(a_1, s, s') \vee Do(a_2, s, s').$$

5. Non-deterministic choice of action parameters:

$$Do((\pi x)a, s, s') =_{def} (\exists\, x) Do(a, s, s').$$

6. Conditional actions:

$$\textbf{if } f \textbf{ then } a_1 \textbf{ else } a_2 =_{def} [f?; a_1] | [\neg f?; a_2].$$

7. Non-deterministic iteration, execute a zero or more times:

$$\begin{aligned} Do(a^*, s, s') =_{def} & \\ & (\forall\, P).[(\forall\, s_1) P(s_1, s_1)] \wedge \\ & [(\forall\, s_1, s_2, s_3).P(s_1, s_2) \wedge Do(a, s_2, s_3) \supset P(s_1, s_3)] \\ & \qquad \supset P(s, s'). \end{aligned}$$

   Not surprisingly, a second order definition is necessary to introduce non-deterministic iteration. This definition establishes that the set of states that can be reached by performing the action a^* is the reflexive and transitive closure of the performance of a.

8. While loops:

$$\textbf{while } f\ a =_{def} [f?; a]^*; \neg f?.$$

5.1.2 Occurrences and Complex Actions.

In order to integrate complex events with the situation calculus extended with an actual line, we introduce a new element to the set of *extra-logical* symbols: *Occurs*. The definition of *Occurs* relies on the definition of *Do* given above. *Occurs* is defined as:

$$Occurs(a,s) =_{def} (\exists\, s').actual(s') \wedge Do(a,s,s').$$

From this definition, we obtain:

1. If a is a primitive action:

$$\begin{aligned} Occurs(a,s) &\equiv (\exists\, s').actual(s') \wedge Poss(a,s) \wedge s' = do(a,s) \\ &\equiv Poss(a,s) \wedge actual(do(a,s)). \end{aligned}$$

   which, from (4.1) and (4.7), leads to:

$$\begin{aligned} Occurs(a,s) &\equiv actual(do(a,s)), \\ Occurs(a,s) &\equiv occurs(a,s). \end{aligned}$$

   As should have been expected, the expression $Occurs(a,s)$ gets translated into $occurs(a,s)$ when a is a primitive action.

2. Tests:

$$Occurs(f?,s) \equiv actual(s) \wedge holds(f,s).$$

3. Sequence:

$$Occurs([a_1;a_2],s) \equiv (\exists\, s').actual(s') \wedge (\exists\, s'').Do(a_1,s,s'') \wedge Do(a_2,s'',s').$$

   By simple logical manipulation we can derive:

$$Occurs([a_1;a_2],s) \equiv (\exists\, s')\ Occurs(a_1,s) \wedge Do(a_1,s,s') \wedge Occurs(a_2,s').$$

4. Non-deterministic choice between two actions:

$$Occurs([a_1|a_2],s) \equiv (\exists\, s').actual(s') \wedge [Do(a_1,s,s') \vee Do(a_2,s,s')].$$

   which, by simple manipulation, leads to:

$$Occurs([a_1|a_2],s) \equiv Occurs(a_1,s) \vee Occurs(a_2,s).$$

5. Non-deterministic choice of action parameters:

$$Occurs((\pi x)a,s) \equiv (\exists\, s').actual(s') \wedge (\exists\, x)Do(a,s,s'),$$

   which is equivalent to:

$$Occurs((\pi x)a,s) \equiv (\exists\, x)Occurs(a,s).$$

6. Conditional actions: As before, simple logical manipulation leads to:

$$Occurs(\mathbf{if}\ f\ \mathbf{then}\ a_1\ \mathbf{else}\ a_2, s) \equiv holds(f,s) \wedge Occurs(a_1,s) \vee \neg holds(f,s) \wedge Occurs(a_2,s).$$

   It is illustrative to rewrite this expression as:

$$\begin{aligned} &Occurs(\mathbf{if}\ f\ \mathbf{then}\ a_1\ \mathbf{else}\ a_2, s) \equiv \\ &\qquad [holds(f,s) \supset Occurs(a_1,s)] \wedge [\neg holds(f,s) \supset Occurs(a_2,s)]. \end{aligned}$$

   Also, notice that in the derivation of the expression for occurrences of conditional actions we make use of the abbreviation $\neg f$, which is defined by (2.23).

7. Non-deterministic iteration.

$$Occurs(a^*, s) \equiv (\exists\, s').actual(s') \wedge Do(a^*, s, s').$$

8. While loops.

$$Occurs(\mathbf{while}\ f\ a, s) \equiv (\exists\, s').actual(s') \wedge Do([f?; a]^*; \neg f?, s, s').$$

   That is:

$$\begin{aligned} Occurs(\mathbf{while}\ f\ a, s) &\equiv (\exists\, s').actual(s') \wedge \neg holds(f,s') \wedge Do([f?;a]^*, s, s') \\ &\equiv (\exists\, s').actual(s') \wedge \neg holds(f,s') \wedge \\ &\quad (\forall P).[(\forall\, s_1)P(s_1,s_1)] \wedge \\ &\quad [(\forall\, s_1,s_2,s_3).P(s_1,s_2) \wedge holds(f,s_2) \wedge Do(a,s_2,s_3) \supset P(s_1,s_3)] \\ &\qquad \supset P(s,s'). \end{aligned}$$

   It follows that s' is a state that is obtained by performing a 0 or more times from the situation s. As long as f holds, a is performed again.

As pointed out before, a difference between the approach presented here and that of CR is that the latter approach uses a situation calculus with non-reified fluents. A disadvantage of such an approach is that they need to introduce *pseudo-fluents* in addition to the extra-logical symbols we mentioned before. These *pseudo-fluents* are necessary when writing *test* actions, in fact, we simply write:

$$Do(f?, s, s') =_{def} holds(f,s) \wedge s = s'.$$

and f is a situation term in our language. In contrast, CR write

$$Do(p?, s, s') =_{def} p[s] \wedge s = s'.$$

where p is a *pseudo-fluent*, which stands for a formula in the situation calculus, but with all state arguments suppressed. $p[s]$ denotes the formula obtained from p by restoring the state arguments with the term s. An advantage of CR's approach is that they may write expressions such as:

$$\mathbf{while}[(\exists\, block)\ ontable(block)]remove\_a\_block,$$

which is not possible for us, since $(\exists\, block)\ ontable(block)$ is not a fluent. However, we can achieve the same effect by extending the language with a new fluent that holds in a situation exactly when the existential sentence is true. Thus, we add a fluent constant $Exisf$, such that

$$holds(Exisf, s) \equiv (\exists\, x) holds(f(x), s).$$

Therefore, the same meaning can be achieved by adding new fluent constants to the language (for another example, see below).

Notice that in the same way as complex events are treated as meta-logical constructs in the standard situation calculus, we treat occurrences of these complex events as meta-logical constructs in the situation calculus extended with a notion of occurrences. Furthermore, all the advantages gained by introducing the notion of simple action occurrences are inherited by the language with the meta-logical constructs *Do* and *Occurs*. For example, we can write expressions involving the abbreviation *Occurs* that would express constraints between complex actions. For example, to represent the fact that "John removed all the items from the table one by one and then served Mary a cup of tea," we can write:

$$\begin{aligned}&holds(Empty, s) \equiv \neg(\exists\, x)\ holds(ontable(x), s),\\&(\exists\, s)\ Occurs([\textbf{while}\ \neg Empty\ (\pi x)\ pickup(John, x); serves(Tea, John, Mary)], s).\end{aligned}$$

where we are assuming a suitable situation calculus language with the appropriate function symbols and constants. Another interesting advantage is that if we want to deal with the occurrence of complex events defined in terms of primitive events only, we need not appeal to the meta-logical constructs. In fact, if we want to state that "Mary took a sip of tea and smiled," we can simply write:

$$(\exists\, s)\ occurs(TakeSip(Mary), s) \wedge occurs(Smile(Mary), do(TakeSip(Mary), s)).$$

Thus, with respect to the *actual* line of situations, some of the expressive gains of the complex meta-logical actions are directly available within the situation calculus with occurrences.

5.2 Concurrency.

5.2.1 A Richer Ontology of Actions.

In the previous chapters, we considered axiomatizations based on the assumption that actions were not performed concurrently. Therefore, we provided necessary conditions for the execution of actions in isolation. Similarly, the effect axioms are written for non-concurrent actions. In this section, we discuss a simple proposal to integrate concurrency in theories of action based on the situation calculus. Also, we analyze the problems that arise when combining concurrent actions with an actual line of situations.

In general, concurrency does not offer significant problems if actions do not interact. The difficulties arise when actions performed concurrently interact with one another. In the same manner as CR's approach to complex actions, we want a way of dealing with concurrent actions that inherits the solution to the frame problem.

In order to introduce concurrency, we extend the language in a manner similar to [16]. Thus, we introduce the function $+ : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$, so that $a_1 + a_2$ denotes the action of performing a_1 and a_2 concurrently. We also introduce the predicate $\in \subseteq \mathcal{A} \times \mathcal{A}$, such that $a_1 \in a_2$ iff a_1 is part of action a_2. The axioms that characterize these new elements are:

$$a = a + a, \tag{5.1}$$

$$a_1 + a_2 = a_2 + a_1, \tag{5.2}$$

$$a_1 + (a_2 + a_3) = (a_1 + a_2) + a_3, \tag{5.3}$$

$$a_1 \in a_2 \equiv (\exists\, a)\ a_2 = a_1 + a. \tag{5.4}$$

We introduce the predicate *primitive* $\subseteq \mathcal{A}$ to distinguish those actions that are not the result of other actions being performed concurrently:

$$primitive(a) \equiv (\forall a').a' \in a \supset a = a'. \tag{5.5}$$

A consequence of (5.1)-(5.5) is that we cannot assume unique names for actions. In fact for any action name A, $A+A$ is another name for it. This drawback can be dealt with by providing unique names axioms for all action terms that do not mention the function symbol $+$. Furthermore, in order to derive inequalities of the kind $A_1 \neq A_2$, the following necessary condition for the equality of action terms needs to be introduced:

$$[(\forall a')(a' \in a_1 \equiv a' \in a_2)] \supset a_1 = a_2. \tag{5.6}$$

Thus, two actions are equal if they contain exactly the same sub-actions.

There are two problems that arise when considering concurrency, namely the *precondition interaction* problem, and the *effect interaction* problem. The *precondition interaction* problem arises when a pair of actions cannot be performed concurrently. This is normally due to incompatible requirements for the physical performance of the actions. We adopt the point of view that in general this incompatibility can be modeled in terms of the resources that each action uses. For example, if we need some resource for the execution of some action A_1, and some other action A_2 needs the same resource, then it follows that the action $A_1 + A_2$ is not possible unless the said resource can be shared.

On the other hand, the *effect interaction* problem has to do with the effects of actions. Thus, assuming that a pair of actions can be executed concurrently, we need to determine how they affect the world. We take the point of view that each action has some *direct* or *primary* effects. These effects arise as a result of performing the action given specific conditions on the situation in which they are performed. The direct effects will be present whether or not the action is performed concurrently with others. Thus, direct effects are never cancelled by the concurrent execution of other actions. For example, the action of pouring a liquid out of a jug results in liquid coming out of the jug. The only way to prevent this effect, given the conditions, is by impeding the execution of the action. However, if the action is performed in the right situation the primary effects *must* arise. In addition to primary effects, we consider that actions have *indirect* or *secondary* effects. These effects arise depending on whether or not other actions are performed concurrently. For example, if the actions of lifting one side of a table and lifting the other side of the table are performed concurrently, then the table will be lifted in the air. In our approach to deal with indirect effects, we adopt the point of view that all the secondary effects can be derived from state constraints. Thus, in the example, we will have a constraint stating that whenever both sides of the table are lifted, the table is lifted.

5.2.2 The Precondition Interaction Problem.

If A_1 and A_2 are two distinct actions for which we have an adequate characterization of $Poss(A_1, s)$ and $Poss(A_2, s)$, we would like to have a systematic way to obtain a characterization for $Poss(A_1 + A_2, s)$. For example, consider the action term $paint(agent, wall, colour)$ used to denote the action of some *agent* painting some *wall* with some *colour*. Also, assume that we have established that $Poss(paint(G_1, W_1, C_1), S_0)$ and $Poss(paint(G_2, W_2, C_2), S_0)$ are true. Of course, we do not want to infer that:

$$Poss(paint(G_1, W_1, C_1) + paint(G_2, W_2, C_2), S_0)$$

is also true. In particular, if $G_1 = G_2$, and $W_1 \neq W_2$, then the concurrent execution of both actions should not be possible (assuming that the agent cannot paint two walls simultaneously). In order to provide an appropriate axiomatization, we propose to write the following general axiom about concurrent actions:

$$Poss(a_1 + a_2, s) \equiv Poss(a_1, s) \wedge Poss(a_2, s) \wedge \neg precInt(a_1, a_2). \tag{5.7}$$

Here, we have introduced the predicate $precInt(a_1, a_2)$ to say that there is interaction between the preconditions of a_1 and a_2, making the concurrent execution of a_1 and a_2 impossible. From (5.1)-(5.1) and (5.7) it follows that:

$$precInt(a_1, a_2) \wedge a_1 + a_2 \in a \supset \neg Poss(a, s). \tag{5.8}$$

The axiomatization of *precInt* seems to be domain dependent. However, at the very least we need the following:

$$\neg precInt(a, a).$$

That is, an action never interacts with itself. Depending on the problem at hand, we would need to provide a different level of granularity in the formalization of *precInt*. At one end of the spectrum, we could incorporate the laws governing the phenomena that determines whether or not *precInt* holds for a given set of actions. For example, in the physical world, we might incorporate axioms detailing notions of space. Thus, from these axioms we would infer that actions performed by different agents requiring the same space are not possible. A complete axiomatization of *precInt* is in general not possible. There are several ways to deal with an incomplete axiomatization. For instance, we could list necessary conditions for *precInt* to be true and *non-monotonically* assume that these conditions are also sufficient. Alternatively, we choose to abstract away from a fine level of granularity by appealing to the notion of *resource*. For example, to paint, a painter needs a brush, paint, light, and access to the object to be painted (e.g., a wall). In order to talk about resources, we introduce two predicates, *xres* and *sres* which take an action and an object as arguments. $xres(a, r)$ means that action a requires the exclusive use of some resource r. On the other hand, $sres(a, r)$ means that action a requires the use of r for its execution, but that the resource can be shared. For example, we may write:

$$xres(Paint, Brush) \wedge xres(Paint, Wall) \wedge xres(Paint, PaintCanister), \tag{5.9}$$
$$sres(Paint, Light). \tag{5.10}$$

Here, (5.9) states that to *Paint*, the painter needs exclusive access to a brush, wall and paint. Also, (5.9) states that in order to paint, the painter needs light, which can be shared. Given the information about the exclusive and shared resources we may determine whether actions interact on that basis:

$$\begin{aligned} precInt(a_1, a_2) \equiv\ & a_1 \not\in a_2 \wedge a_2 \not\in a_1 \wedge \\ & (\exists r)\ [xres(a_1, r) \wedge xres(a_2, r) \vee \\ & \quad sres(a_1, r) \wedge xres(a_2, r) \vee \\ & \quad xres(a_1, r) \wedge sres(a_2, r)]. \end{aligned}$$

Our separation of resources into exclusive and shared is rather simplistic. It is not difficult to come up with situations in which this distinction will be inadequate. For example, I can share my office with one office-mate and we can both work at the same time, but if you try to put an extra person in our office we will both be unable to work there. A more general solution to this problem is obtained by providing a more detailed axiomatization. For example, we can state explicitly how much space is necessary in order for each of us to work. Also, we would need a sentence stating that if there is not enough space available, then work becomes impossible.

5.2.3 Effect Interaction Problem.

Once we have established that $Poss(A_1 + A_2, S)$ is true for some situation S, we have to determine what the effects of the combined execution of both actions are. There are two phenomena associated with this interaction:

1. **Cancellation:** Actions may cancel each other's effects. For instance, if an agent pushes a door to open it, the door will become open. This effect is cancelled, however, if at the same time another agent pushes the door in the opposite direction with an equal or greater force.
2. **Synergy:** Actions may have synergistic effects. I.e., two actions, when performed in conjunction, will provoke changes in the world that would not be provoked by any of the actions performed in isolation.

We view these two phenomena as *ramifications* of the individual actions, which are derived from state constraints. To illustrate this, we use Lifschitz' bowl of soup example [16]. In this example we have a bowl of soup and the actions $LiftLeft$ and $LiftRight$, which represent the actions of lifting the left and right side of the bowl respectively. Also, we have the fluents $Lifted$, which holds if the bowl is lifted, and $Llifted$ and $Rlifted$ that hold if the left and right sides of the bowl are lifted respectively. Also, we have the following[1]:

$$holds(Lifted, s) \equiv holds(Llifted, s) \wedge holds(Rlifted, s), \tag{5.11}$$

$$holds(Spilled, s) \equiv holds(Llifted, s) \oplus holds(Rlifted, s). \tag{5.12}$$

To simplify the discussion, we assume that there is no interaction among action preconditions:

$$\neg precInt(a, a'). \tag{5.13}$$

The *effect interaction problem* is solved by deriving effect axioms for fluents. These axioms describe the consequences of executing a concurrent action. In our view, effect axioms represent *direct* effects of certain actions. That is, they are effects that must be present given the right environmental conditions (i.e., qualifications) regardless of whether there are other actions occurring at the same time. For example, the action *Close-door* is such that the door being acted upon *must* be closed immediately after the action is performed. Otherwise, it would be difficult to argue that the action took place. If this is unsatisfactory, we would be required to provide an extra level of granularity. In the *Close-door* example, we may have an action *attempt to close the door* whose direct effect is to apply a force of certain magnitude on the door. If all other forces are such that the net force is not enough to close the door, then the door will remain open.

Notice that the consequences of taking this point of view are technical as well as conceptual. Conceptually, we need to identify those consequences that result *directly* from the execution of an action, independently of what else occurs simultaneously. Technically, we express the relation between an action and its direct consequences as *effect axioms*. On the other hand, if some consequences of an action are subject to whether or not the action occurs in conjunction with other actions, then these consequences are ramifications of some intermediate *primary* consequences.

[1]We use $\oplus$ to denote *exclusive or.*

Thus, the *primary* effects of an action are specified with effect axioms. For example:

$$Poss(LiftLeft, s) \supset holds(Llifted, do(LiftLeft, s)), \tag{5.14}$$

$$Poss(LiftRight, s) \supset holds(Rlifted, do(LiftRight, s)). \tag{5.15}$$

The *secondary* effects are inferred from primary effects and state constraints. These constraints establish relationships between primary and secondary effects. Thus, the state constraints (5.11) and (5.11) determine that *Lifted* and *Spilled* are secondary effects of the actions *LiftLeft* and *LiftRight*.

Finally, if before we wrote effect axioms as:

$$Poss(a, s) \wedge \gamma_R^+(a, s) \supset holds(R, do(a, s)), \tag{5.16}$$

we now write them as:

$$Poss(\alpha, s) \wedge \gamma_R^+(a, s) \wedge a \in \alpha \supset holds(R, do(\alpha, s)). \tag{5.17}$$

Here, R is a primary consequence of action a. Thus, instead of (5.14) and (5.14), in the bowl of soup example, we have:

$$Poss(\alpha, s) \wedge LiftLeft \in \alpha \supset holds(Llifted, do(\alpha, s)),$$
$$Poss(\alpha, s) \wedge LiftRight \in \alpha \supset holds(Rlifted, do(\alpha, s)).$$

That is, no matter what other actions are performed, if *LiftLeft* is one of the sub-actions executed, then *Llifted* will hold, and similarly for the right side.

Now, in our example, there are two definitions (for *Lifted* and *Spilled*). Therefore, we can use the approach described in chapter 3 and generate a theory of action with a set of successor state axioms. In particular, we would obtain:

$$Poss(a, s) \supset holds(Llifted, do(a, s)) \equiv LiftLeft \in a \vee holds(Llifted, s). \tag{5.18}$$

Along with an analogous axiom for *Rlifted*. Obviously, to determine the values of the defined fluents it suffices to use the definitions (5.11) and (5.11).

In summary, we build a theory in which effect axioms describe direct effects of actions, and constraints determine the secondary effects (which are ramifications of the primary effects). In the same manner as with theories with non-concurrent actions, we can solve the frame and ramification problems for theories that contain stratified definitions or binary state constraints. This is illustrated in the example that follows.

5.2.4 An Example.

Here we consider the traditional *Producer/Consumer* problem, which is the example of choice in Computer Science to illustrate the problem of concurrent execution of actions [44]. Specifically, it is said that two processes are in a *Producer/Consumer* relationship when one of these processes generates output that is the input to the other process. The problem consists of correctly modelling the behavior of the two processes under any possible interleaving or concurrent execution of the basic operations of production and consumption.

To make the example more concrete, we consider that there are only two primitive actions *Produce* and *Consume*. Thus:

$$(\forall a).a = Produce \vee a = Consume,$$
$$Consume \neq Produce.$$

The *Produce* action generates an item. A *Consume* action, takes an unused item if one is available and consumes it. Thus, the producer and consumer are in a pipeline. As an example, a producer may generate ASCII characters to be used by a consumer that counts words. To model the state of the process we use three counters: *inBin*, *produced* and *consumed*. Each one of these counters is modelled as a fluent function symbol that takes an integer as an argument. For example, $holds(inBin(N), S)$ would be true if in situation S there are N items produced and not yet consumed. $holds(produced(N), S)$ and $holds(consumed(M), S)$ are true if in S there have been N items produced and M items consumed. *Produce* increases the items produced by one and *Consume* increases the items consumed by one. We have the following constraints:

$$n \neq m \supset \neg holds(produced(n), s) \vee \neg holds(produced(m), s), \tag{5.19}$$

$$n \neq m \supset \neg holds(consumed(n), s) \vee \neg holds(consumed(m), s). \tag{5.20}$$

We introduce a defined fluent *inBin*, which also takes a non-negative integer as an argument. We define it as:

$$holds(inBin(n), s) \equiv (\exists\, n_p, n_c).holds(consumed(n_c), s) \wedge holds(produced(n_p), s) \wedge n = n_p - n_c,$$

where we assume the usual interpretation for the operation minus $(-)$. The constraint:

$$n \neq m \supset \neg holds(inBin(n), s) \vee \neg holds(inBin(m), s), \tag{5.21}$$

can be derived from the the definition of *inBin* and the state constraints for *produced* and *consumed*.

Consumption is not possible unless there is something in the bin. On the other hand, production is always possible:

$$Poss(Consume, s) \equiv (\exists\, n).holds(inBin(n), s) \wedge n \neq 0,$$
$$Poss(Produce, s).$$

The effect axioms for these actions are as follows:

$$Poss(\alpha, s) \wedge Produce \in \alpha \wedge holds(produced(n_p - 1), s) \supset holds(produced(n_p), do(\alpha, s)), \tag{5.22}$$

$$Poss(\alpha, s) \wedge Consume \in \alpha \wedge holds(consumed(n_c - 1), s) \supset holds(consumed(n_c), do(\alpha, s)). \tag{5.23}$$

Notice that it is tempting to state that a consumption would decrease the *inBin* number and that production would increase it. However, this is an incorrect approach, since in general such a statement is not true. Specifically, when a production and a consumption are performed simultaneously, the statement does not hold. In fact, a reduction of the elements in the bin is a ramification of consumption when nothing else happens concurrently.

To derive the successor state axioms we follow the same procedure utilized for non-concurrent actions. Notice that the constraints (5.19)-(5.21) are of the form (3.1). Therefore, we can replace them with a suitable set of new effect axioms derived from the original effect axioms and the state constraints. After this is accomplished, we follow Reiter's strategy to generate the successor state axioms. For instance, in order to derive the successor state axioms for *produced*, we first need to derive the effect axioms that result from (5.19) and (5.22). We obtain the following:

$$\begin{aligned} Poss(\alpha, s) \wedge Produce \in \alpha \wedge m \neq n_p \wedge holds(produced(n_p - 1), s) \\ \supset \neg holds(produced(m), do(\alpha, s)). \end{aligned} \tag{5.24}$$

Therefore, to derive the successor state axiom for *produced* we use effect axioms (5.22) and (5.24) and obtain:

$$\begin{aligned}&Poss(\alpha, s) \supset holds(produced(n), do(\alpha, s)) \equiv\\ &\qquad [Produce \in \alpha \wedge holds(produced(n-1), s)] \vee\\ &\qquad holds(produced(n), s) \wedge \neg[(\exists\, n_p)\ Produce \in \alpha \wedge holds(produced(n_p - 1), s) \wedge n \neq n_p].\end{aligned}$$

Utilizing the state constraint (5.19) we can simplify this expression to obtain:

$$\begin{aligned}&Poss(\alpha, s) \supset holds(produced(n), do(\alpha, s)) \equiv\\ &\qquad [Produce \in \alpha \wedge holds(produced(n-1), s)] \vee\\ &\qquad\quad \neg Produce \in \alpha \wedge holds(produced(n), s).\end{aligned}$$

Following a similar procedure, we also derive:

$$\begin{aligned}&Poss(\alpha, s) \supset holds(consumed(n), do(\alpha, s)) \equiv\\ &\qquad [Consume \in \alpha \wedge holds(consumed(n-1), s)] \vee\\ &\qquad \neg Consume \in \alpha \wedge holds(consumed(n), s).\end{aligned}$$

We assume that there is no precondition interaction between *Consume* and *Produce*, so:

$$\neg precInt(Consume, Produce).$$

Therefore, it follows that the action *Consume* + *Produce* is possible in those situations in which each sub-action is individually possible.

As initial conditions, we require:

$$holds(consumed(0), S_0) \wedge holds(produced(0), S_0) \wedge holds(inBin(0), S_0).$$

From the above axiomatization, and with the induction axiom, we can infer that the *produced* number is always greater or equal to the *consumed* number[2].

$$S_0 \preceq s \wedge holds(consumed(n_c), s) \wedge holds(produced(n_p), s) \supset n_c \leq n_p.$$

This problem serves to illustrate how the interaction of effects can be handled. Particularly, we have shown how possible conflicts between actions may be avoided. For example, in order to determine the number of items that are available for consumption we have a fluent function *inBin*. Changes to this fluent are indirect effects of the *produce* and *consume* actions.

In general, in many areas of computer science, e.g. Operating Systems, the approach to model *concurrency* is to avoid situations in which processes may concurrently access the same pieces of information. Thus, they create tools and concepts like *mutual exclusion*, *critical sections*, and others to specify systems in which interaction between processes is avoided. With respect to our approach to concurrency, this avoidance can be modeled with the *Poss* predicate. Thus, if two operations access mutually exclusive resources, then their concurrent execution is deemed not possible.

[2] Here we utilize the symbol $<$ to denote the usual order among the integers. This overloading of the symbol $<$ should cause no difficulty to the reader since the context identifies which operator we refer to.

5.2.5 Concurrency and Occurrences.

Here we analyze the integration of the notion of occurrences and an actual line of situations with concurrent actions. First, remember that according to our definition, an action a occurs in a situation s if and only if the situation $do(a, s)$ falls in the actual line of situations. On the other hand, notice that if we have an action $A_1 + A_2$ such that $A_1 \neq A_2$, then the occurrence of $A_1 + A_2$ in a situation S is identified with the fact that $do(A_1 + A_2, S)$ belongs to the actual line. Given that $A_1 \neq A_1 + A_2$, this latter situation is not the same as $do(A_1, S)$. So, it follows that the concurrent occurrence of A_1 and A_2 in S implies that neither A_1 nor A_2 occur in S. Thus, in the presence of concurrency, the statement $occurs(a, s)$ should be understood as: "the global action a occurred in s", therefore nothing else occurred. To formalize this, we have:

Observation 5.1

$$occurs(a, s) \wedge a' \neq a \supset \neg occurs(a', s),$$

which follows trivially from (4.1) and (4.7).

This seems counterintuitive and inflexible. In particular, we may want to state that some action A_1 occurred in some situation without necessarily implying that A_1 is all that occurred. With this purpose in mind we introduce a new predicate $occurs^c \subseteq \mathcal{A} \times \mathcal{S}$ defined as:

$$occurs^c(a, s) \equiv (\exists\, a')occurs(a', s) \wedge a \in a'.$$

An analogous problem arises when looking at occurrences in the time line using the predicate $occurs_{\mathcal{T}}$. That is, $occurs_{\mathcal{T}}(a, t)$ means that the global action a has occurred at time t. Hence, we also need to introduce the predicate $occurs^c_{\mathcal{T}} \subseteq \mathcal{A} \times \mathcal{T}$ as:

$$occurs^c_{\mathcal{T}}(a, t) \equiv (\exists\, a', s).occurs(a', s) \wedge a \in a' \wedge start(do(a', s)) = t. \tag{5.25}$$

As a simple example, consider the Producer/Consumer axiomatization presented before. Furthermore, assume that we are told that:

$$(\exists\, s_c).actual(s_c) \wedge holds(inBin(0), s_c).$$

If this is all the domain information available, the (non-circumscriptive) models of this domain will contain an infinite number of models. In one model, the situation denoted by s_c will be equal to the initial situation. Also, there is an infinite number of models in which s_c equals a situation obtained by performing some arbitrary legal sequence of actions such that all elements produced have been consumed.

As is the case with non-concurrent actions, we may be interested in finding out what in all likelihood occurs. Unfortunately, the circumscription policy (4.23) does not do the job. To illustrate why, assume that we have a theory in which all that is known is that $occurs^c(A, S_0)$. Not having other information, we would like to infer that A is all that ever occurred. However, using our previous circumscription policy, a model in which $occurs(A, S_0)$ is true and one in which $occurs(A + B, S_0)$ is true are not comparable. Thus, there will be a minimal model in which $occurs(A + B, S_0)$ is true. Therefore, we need to slightly modify the circumscription policy in worlds in which concurrency is possible. The policy we use is:

$$Circ(\Sigma_{conc}; occurs^c_{\mathcal{T}}; actual, start, \mathbf{S_k}), \tag{5.26}$$

where the varying terms of the circumscription are the same as in (4.23).

Now, assume that using the same background theory of the Producer/Consumer problem, we have that:

$$\begin{aligned} &actual(S_c) \wedge holds(inBin(1), S_c),\\ &holds(consumed(N), S_c) \wedge N \geq 2. \end{aligned}$$

Proposition 5.1 *The circumscription policy leads to:*

$$holds(consumed(2), S_c), \tag{5.27}$$
$$holds(produced(3), S_c). \tag{5.28}$$

Proof: First, we show that there is model in which (5.27)-(5.27) hold. Here, we introduce new constants S_1, S_2, S_3, and S_4, as abbreviations for the following situations:

$$\begin{aligned} S_1 &= do(Produce, S_0),\\ S_2 &= do([Produce, Consume], S_0),\\ S_3 &= do([Produce, Consume, Produce], S_0),\\ S_4 &= do([Produce, Consume, Produce, Produce + Consume], S_0). \end{aligned}$$

There are several models that satisfy (5.27) and (5.27). In particular, consider a model in which:

$$actual(s) \equiv s \leq do([Produce, Consume, Produce, Produce + Consume], S_0).$$

This model satisfies:

$$\begin{aligned} occurs_{\mathcal{T}}(a,t) \equiv\ & a = Produce \wedge t = start(S_1) \vee\\ & a = Consume \wedge t = start(S_2) \vee\\ & a = Produce \wedge t = start(S_3) \vee\\ & a = Produce + Consume \wedge t = start(S_4). \end{aligned}$$

Clearly, this is a minimal model since no elimination of action occurrences would yield another model. Also, there are other minimal models, all of which have the same number of items produced and consumed.

Finally, we need to show that any model that does not satisfy (5.27)-(5.27) is not minimal. We do so by contradiction. Assume that there is a minimal model in which there are n items consumed. This model has to satisfy:

$$holds(produced(n+1), S_c) \wedge holds(consumed(n), S_c) \wedge actual(S_c) \wedge n > 2.$$

Now, there are many models that can be built satisfying this sentence. Let us pick an arbitrary such model (call it $\mathcal{M}$). Thus, given a model that satisfies the above sentence we can always build a model that is preferred to it and that satisfies (5.27)-(5.27). To do so, we take the original model $\mathcal{M}$ and build a new model $\mathcal{M}'$ that shares the exact same situation structure. In the new model the branch that is considered actual in $\mathcal{M}$ will be considered a non-actual branch. Furthermore, from this branch, we take the set of actions that have occurred and eliminate all but two consumptions and all but three productions (maintaining a legal sequence of productions and consumptions). For example, from the occurrences:

$$\{produce, produce + consume, produce, produce, produce + consume\},$$

we would eliminate the last two productions and leave:

$$\{produce, produce + consume, produce, consume\}.$$

Finally, the reduced set of *Produce* and *Consume* operations names a branch in the tree which we deem *actual*. Obviously, the model $\mathcal{M}'$ is preferred to $\mathcal{M}$.
□

Chapter 6

Continuity and Natural Events.

6.1 Introduction and Motivation.

An important drawback of the original situation calculus and some of its extensions is the impossibility of talking about properties that change continuously with time. This is the case with the discrete situation calculus, in which an induction axiom (e.g., (2.1)) is used to characterize the set of existing situations. For example, consider the problem of modeling the dynamic behavior of physical objects. To do so, we need a language to describe different properties of these objects. For instance, we may want to describe situations in which an object is falling. Thus, we may introduce a fluent term $falling(A)$ to denote the property *object A is falling.* Therefore, we may write $holds(falling(A), s)$ to mean that $falling(A)$ is true in some situation s. Also, assume that we want to describe the position of the object. Specifically, we may want to consider the property *object A is at height Y.* We might be tempted to write an expression like:

$$holds(height(A, Y), S),$$

as a sentence intended to mean that S is a situation in which A is at height Y. However, if A is falling, and we take height to be a real number, such a formula would be true at an instant of time only. For this to be logically correct, we would need S to span a single time point. Therefore we would need a different situation for each time point in which the object is falling. Since we take time to correspond to the non-negative real numbers, we would need a continuum of situations. This, of course, violates the induction axiom. Hence, the property of an object being in a certain position in space cannot be denoted by a fluent.

The previous problem does not arise if one gives up the notion that space-time has a continuous nature. For example, we may consider that coordinates are isomorphic to the integers, and that objects stay a positive and finite length of time at each location. In this approach, if an object falls it would pass through a finite set of points during any interval of time. However, this is unsatisfactory for several reasons. The most important one is that it introduces severe representational constraints. In fact, it does not adequately describe the physical phenomena we want to reason about. Now, assuming that we are willing to accept this representational limitation, the resulting framework is very artificial. In particular, since the ball would be changing position from one situation to the next, we would have to introduce actions to provoke these changes at every step.

Another possibility is to consider actions that have durations. For example, the falling of a ball can be considered to be an event that results in the ball hitting the ground. Such an approach greatly complicates the structure of the tree of situations. Furthermore, in this approach it is unclear how to consider occurrences of events while other events are in progress.

Instead, we propose a solution that involves dividing the properties of the world into two different classes. First, we have the properties modeled using standard fluents, which we also call *discrete fluents*. Second, we have the properties that vary continuously with time (e.g., the position of an object), which we call *continuous parameters*. Then, the property *object A is falling* will be considered a discrete fluent. As before, actions may be performed that would make discrete fluents change from one situation to another. For example, some action or event might produce a change that would make the discrete fluent *object A is at rest* true, and the property *object A is falling* false. On the other hand, continuous parameters, such as the *position of object A is x* do not only change due to actions being performed by agents, but they also change due to the object's natural *inertia*. For example, if the ball in our example is falling, its position will be changing until some action that stops its motion is performed.

A further distinction between discrete fluents and continuous parameters is that the former hold or do not hold throughout situations, whereas continuous parameters may change their value continuously within a situation. This taxonomic division of properties of the world is not new. In fact, Galton [14] proposes a similar distinction within the framework of Allen's theory of time [3]. Galton uses the terminology *state of motion* and *state of position* for properties that correspond to our discrete fluents and continuous parameters respectively. Also, related ideas are explored by Shanahan in the framework of the event calculus in logic programming [55].

The majority of the problems that deal with properties that vary continuously with time are problems in the domain of physics. Since physics is concerned with building mathematical models for natural phenomena, we have to model actions or events that are considered *natural*. We use the term *natural event* to refer to events whose occurrence is determined by the laws of nature (physics). As an example, imagine that we start pumping a continuous flow of air into a rubber balloon. The inflating of the balloon will cause it to explode (a natural event), unless something happens to stop the flow of air. In other words, the laws of physics prescribe that when the pressure inside the balloon reaches a certain point, the balloon ought to explode. Also, we use the term *natural process* to refer to a totally ordered set of natural events that occur in sequence. For example, if a ball is released, the set of bounces can be considered a natural process.

In contrast to what we call natural events, the actions that have been considered in the previous chapter are *performed* by some *agent*. An important difference between agent-driven actions and natural events is that the former can be performed at any time (given that the preconditions for their execution are true). On the other hand, natural events cannot happen at an arbitrary time. Rather, they occur if and only if the environmental conditions dictate that their occurrence must arise. In this chapter we present a characterization of natural events. Clearly, a characterization of natural phenomena is essential to build systems for reasoning about physical processes.

Our interest is to provide a formal foundation for specifying dynamic systems. To do so, we present a mechanism to integrate continuous change into the situation calculus. An important advantage of our approach is that the notions of *fluent* and *situation* are maintained. Thus, we maintain the view that fluents are properties that change from situation to situation. Also, situations are *discrete*. Therefore, the same approach to deal with the frame and ramification problems is applicable in the extended language presented here.

6.2 Extensions to the language.

6.2.1 An Oracle.

We extend the language with a sort $\mathcal{R}$ for the reals. We consider the sort $\mathcal{T}$ to be a subset of $\mathcal{R}$. Again, we work with an interpreted theory in which $\mathcal{R}$ is fixed. Thus, we only consider standard interpretations for the sort $\mathcal{R}$. Furthermore, we add some standard functions for the reals, (i.e., $+$, $-$, $\times$, $/$, etc.) along with an infinite set of *numerical* constants (e.g. 0, -1, 3.14, etc.). We also assume that we work with standard interpretations for these functions and constants.

Another important assumption we make here, is that we have an Oracle capable of answering questions about the truth or falsity of some sentences. For example, if in the course of a proof we need to show that:

$$(\exists\, k, t).t > 0 \land k > 0 \land k \times t = 12,$$

we assume that our Oracle will correctly respond *true*.

It may be argued that the Oracle introduced here is not really necessary. This is due to the fact that we are working with an interpreted theory in which the sort of real numbers has a fixed interpretation. Similarly, we assume that the real function symbols (e.g., $+, \times$, etc.) are also interpreted in the standard way. The reason we introduce this Oracle is that it allows us to cleanly separate the tasks of reasoning within the logic of the situation calculus and the task of reasoning about the reals.

We appeal to the Oracle to respond to questions regarding relationships between real numbers. For instance, solving systems of equations, and determining the existence of solutions to equations or inequalities. By using such an Oracle, we deliberately gloss over many issues regarding reasoning about equations and real numbers. This simplification allows us to concentrate on the extensions that are necessary for the situation calculus to deal with continuity.

In order to characterize the Oracle, we consider that we have:

- A language $\mathcal{L}_{\mathcal{R}}$, which is a sublanguage of the situation calculus that contains the single sort $\mathcal{R}$ of real numbers. A set of real variables and constants, the binary function symbols $+, -, \times, / : \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R}$, the predicate symbol $< \subseteq \mathcal{R} \times \mathcal{R}$ and the equality symbol.

- A theory $\Sigma_{\mathcal{R}}$ in the language $\mathcal{L}_{\mathcal{R}}$ that characterizes the sort $\mathcal{R}$ of real numbers, along with the real operators $+, -, \times$, and $/$. Thus, we assume that the models of $\Sigma_{\mathcal{R}}$ correspond to the standard interpretations for the real numbers and the real operators mentioned above. Furthermore, let $\Phi(\mathbf{x})$ be an arbitrary quantifier free formula, in which $\mathbf{x}$ represents a tuple of real variables. All the free variables mentioned in the formula $\Phi(\mathbf{x})$ are in $\mathbf{x}$. We require that $\Sigma_{\mathcal{R}}$ be such that either:

$$\Sigma_{\mathcal{R}} \models (\exists\, \mathbf{x})\ \Phi(\mathbf{x}), \tag{6.1}$$

or

$$\Sigma_{\mathcal{R}} \models \neg(\exists\, \mathbf{x})\ \Phi(\mathbf{x}). \tag{6.2}$$

In this work we do not address the issue of writing down the theory $\Sigma_{\mathcal{R}}$. Rather, we assume that the theory is given. Finally, we assume that the Oracle is capable of answering whether (6.1) is true or (6.2) is true for any given formula $\Phi(\mathbf{x})$ that obeys the constraints mentioned earlier. Furthermore, we assume that if (6.1) is true, then the Oracle provides us with an assignment of elements of $\mathcal{R}$ to the variables in $\mathbf{x}$ that makes $\Phi(\mathbf{x})$ true.

Obviously, our entire approach to deal with continuity hinges on the expressive capabilities of the language $\mathcal{L}_{\mathcal{R}}$, as well as on the existence of this Oracle. We have taken the simplest possible language $\mathcal{L}_{\mathcal{R}}$ using only the basic real operators. On the other hand, Oracles exist that are capable of answering our questions for a wide variety of classes of existential queries of the kind (6.1), for example, the Maple and Mathematica software systems.

As illustrated in the examples discussed later, we need to appeal to the Oracle in order to prove or disprove that certain sentences of the form $(\exists\, t)\ \phi(t)$ are true.

6.2.2 Ontological Extensions.

Some properties of objects in the world vary continuously with time. For example, the temperature or position of an object. In order to refer to these properties, we introduce a class of objects called *parameters*. These parameters are used to name continuously varying properties. For example, *Pos* may be a parameter that names the position of some object along the X-axis in a cartesian space. The actual real value of the property during a situation is described using a real function of time (e.g., $y(t) = 5 + 6 \times t$). Each such function will have a name (e.g., y). If during a situation s, a property named by a parameter p is behaving according to some function of time named by y we will write an expression stating that p is evaluated through y during situation s.

Formally, we introduce two new sorts in the language. First, we add a sort $\mathcal{P}$ for parameters, and the sort $\mathcal{Y}$ for names of real functions of time. We use the letters p and P for sort $\mathcal{P}$, and y and Y for sort $\mathcal{Y}$. These letters are used with or without subscripts or other markers. Furthermore, we introduce two new functions: First, the function $q : \mathcal{Y} \times \mathcal{T} \to \mathcal{R}$; such that $q(y, t)$ denotes the real number that results from evaluating the function y at time t[1]. Second, the function $\doteq: \mathcal{P} \times \mathcal{Y} \to \mathcal{F}$; such that $\doteq (p, y)$, which we use in infix notation as $p \doteq y$, denotes a fluent that holds when the parameter p behaves as described by the function named by y.

As a simple example, assume that we want to state that the height of some object is given by the function of time $5 + 6 \times t$ in some state S. Then, we may introduce the constant Y to name the function of time $5 + 6 \times t$, and the constant P_h to denote the parameter height. This is stated in the language as follows:

$$(\forall\, t)\ q(Y, t) = 5 + 6 \times t,$$
$$holds(P_h \doteq Y, S).$$

The first sentence states that the value of the function denoted by Y is $5 + 6 \times t$. The second sentence states that the object's height behaves according to Y during the situation S.

An important axiom for the function $\doteq$ is:

$$holds(p \doteq y_1, s) \wedge holds(p \doteq y_2, s) \supset (\forall\, t)\ q(y_1, t) = q(y_2, t). \tag{6.3}$$

Thus, if during a situation s a parameter's value varies according to two functions of time, then these functions must agree on their values at all times.

Since the function symbol $\doteq$ denotes a fluent function, a fluent $p \doteq y$ may change from situation to situation. For example, a golf ball may be following a trajectory whose height is given by a function $2 \times t - t^2$ during S_0. If it falls in a sand trap, its height will be described by the constant 0. Therefore, the event of the ball falling into the trap has had the effect of changing the way in which the ball's height is described (e.g. from a parabolic function of time to a constant). On the other hand, if some other event occurs, the behavior of the ball may be unaffected. Thus, in the

[1]In [50], Sandewall proposes a similar approach to deal with functions of time.

same manner as with traditional fluents, we need to describe how actions affect the behavior of parameters. This is addressed in the following section.

Also, we need to introduce the functions $+$, $-$, $\times$ and $/ : \mathcal{Y} \times \mathcal{Y} \to \mathcal{Y}$. Once again, we overload symbols in the language and rely on the context to disambiguate them. These symbols are introduced to apply real operators to function names, with the meaning described by the following axioms:

$$q(y_1 + y_2, t) = q(y_1, t) + q(y_2, t), \tag{6.4}$$
$$q(y_1 - y_2, t) = q(y_1, t) - q(y_2, t), \tag{6.5}$$
$$q(y_1 \times y_2, t) = q(y_1, t) \times q(y_2, t), \tag{6.6}$$
$$q(y_1 / y_2, t) = q(y_1, t) / q(y_2, t). \tag{6.7}$$

Furthermore, we introduce the functions *constant*, *linear* and *quadratic* of sort $\mathcal{Y}$, with one, two and three real arguments respectively. We utilize these parameters to name real functions of time as follows:

$$q(constant(r_k), t) = r_k, \tag{6.8}$$
$$q(linear(r_k, r_r), t) = r_r + r_k \times t, \tag{6.9}$$
$$q(quadratic(r_k, r_r, r_a), t) = r_a + r_r \times t + 0.5 \times r_k \times t^2. \tag{6.10}$$

We also introduce *derivatives* with respect to time. We use the function $d : \mathcal{Y} \to \mathcal{Y}$, some of whose properties are:

$$d(y) = constant(0) \equiv (\exists r_k)\ y = constant(r_k), \tag{6.11}$$
$$d(y) = constant(r_k) \equiv (\exists r_r)\ y = linear(r_k, r_r), \tag{6.12}$$
$$d(y) = linear(r_k, r_r) \equiv (\exists r_a)\ y = quadratic(r_k, r_r, r_a). \tag{6.13}$$

These properties of the derivatives of polynomial functions could have been obtained from a set of general axioms about derivatives of real functions. However, this approach would take us too far afield. We prefer not to deal with these complications since they fall beyond the scope of this work. Other well known properties of the derivatives of functions could be added to the list (6.11)-(6.11). However, in our examples we only make use of these ones.

In the next section, we utilize the extension of the situation calculus that was just introduced to model natural processes. In particular, we show how to write effect axioms to describe the effects of natural events. Furthermore, we discuss how to derive successor state axioms for fluent terms of the form $p \doteq y$.

6.2.3 Functional Fluents.

An alternative to the approach presented in the previous subsection is to eliminate the use of the sorts $\mathcal{P}$ and $\mathcal{Y}$ and use functional fluents instead. Functional fluents are functions whose denotations vary from situation to situation. Thus, if $under(x)$ is a function denoting the object that is under x, then the denotation of $under(x)$ will vary from situation to situation. In order to accommodate functional fluents in the language of the situation calculus with reified fluents, we need to *reify* equality. Thus:

$$holds(under(D_W) = D_M, s)$$

would be true if in situation s the object under D_W is D_M. Thus, if we want to state that the height of some object is given by the function of time $5 + 6 \times t$ in some state S, then we write:

$$(\forall t)\ holds(height(t) = 6 \times t + 5, S). \tag{6.14}$$

A clear advantage of this approach is that its syntax is much more appealing. However, an important objection to the use of functional fluents is that we want to treat functions of time as domain objects. In our approach, we do so using the names of the functions, rather than the functions themselves. The need to treat functions of time as objects is common when reasoning about physical processes. For instance, in later examples we describe the effects of certain actions by appealing to the derivatives of the real functions of time. This is only possible in a language that allows to treat these functions of time as objects.

Another problem with functional fluents is that they serve to write sentences that describe instantaneous values for physical parameters. In contrast, the approach that utilizes parameters and function names serves to describe behaviors over situations. Thus, if we were to obtain successor state axioms for the approach based on functional fluents this would describe change or non-change of values of parameters at individual time points. Using our approach, successor state axioms state whether or not the overall behavior of some physical parameter changes or not.

6.3 Representation of Natural Processes.

6.3.1 Preliminaries.

In the previous section, we extended the ontology of the situation calculus to include the notions of parameter and names for real function of time. With these new sorts of objects we can model simple physical phenomena using equations on real quantities. We would like to incorporate those models into a logical theory based on the situation calculus extended with occurrences.

We consider only phenomena which can be modeled using a deterministic mathematical model. These deterministic systems are such that if we know the initial conditions, the complete future evolution of the system is described using the model. These mathematical models describe the behavior of *closed* systems. That is, systems in which there are no external perturbations. By integrating these models with the situation calculus we obtain a language in which we may describe the interactions between active agents and physical systems.

As an example, consider a billiards table scenario in which there are two balls. Furthermore, assume that we know the initial position and parameters of each ball (i.e., their position, velocity, acceleration, etc.), an accurate mathematical model will tell us the behavior of the system at all future times. If the balls are moving towards each other, then a collision will occur and such a collision will change the way in which the balls move. When studying these systems, physicists perform "piece-wise" analyses. In the example, the trajectories of each ball would be described mathematically before and after the collision. Thus, the collision would be seen as an event that changes the parameters of the system.

Within the situation calculus framework, we follow the same approach. For example, assume that we place a cartesian coordinate system in the billiards table. Also, assume that the X and Y axes coincide with two perpendicular sides of the table and that $X = 1$ and $Y = 1$ represent the other two sides of the table. Let X^i and Y^i denote the parameters *position of the ball i with respect to the X and Y axes* respectively. Furthermore, assume that the following are the initial conditions:

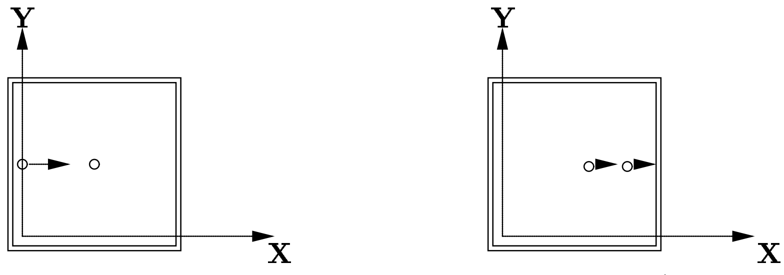
$$holds(Y^1 \doteq constant(0.5), S_0) \wedge holds(Y^2 \doteq constant(0.5), S_0),$$

$$holds(X^1 \doteq linear(1,0), S_0) \wedge holds(X^2 \doteq constant(0.5), S_0).$$

Thus, ball 1 is moving horizontally (i.e., parallel to X-axis) from position $X = 0$, $Y = 0.5$ towards ball 2, which is motionless in the middle of the table (see fig. 6.1a). Therefore, we would like to infer that if no agents are present in the environment, the balls will collide when ball 1 reaches the initial position of ball 2.

Furthermore, if we introduce the action constant *Collision*, then a correct axiomatization of this problem within the framework of the situation calculus would let us infer the following:

- The collision is possible in S_0 if and only if $start(do(Collision, S_0)) = 0.5$[2]. Thus, the time in which the *Collision* may occur in S_0 is fixed.
- After the collision in the initial situation, i.e. in situation $do(Collision, S_0)$, the motion of both balls is affected (see fig. 6.1b). Thus, ball 1 will no longer move according to its original trajectory, and ball 2 will start moving.



(a) Situation S_0. (b) Situation $do(Collision, S_0)$.

Figure 6.1: The Billiards Table Scenario.

Let us consider a second scenario in the billiards table. Assume now that the initial situation changes so that:

$$holds(Y^1 \doteq constant(0.4), S_0) \wedge holds(Y^2 \doteq constant(0.6), S_0),$$
$$holds(X^1 \doteq linear(1,0), S_0) \wedge holds(X^2 \doteq linear(2,0), S_0).$$

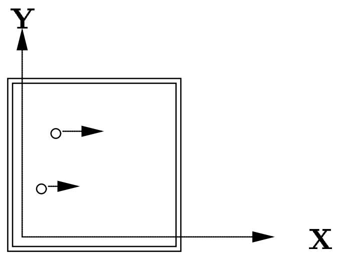


Figure 6.2: The Second Billiards Table Scenario.

To describe this scenario, we introduce new action constants $ReachEdge_1$ and $ReachEdge_2$.

Initially, in S_0, both balls are moving parallel to the X-axis (see figure 6.2). The second ball moves faster than the first one. Given that there are no agents that could introduce perturbations,

[2]We ignore the dimension of the balls.

we would like to infer that ball 1 reaches the edge of the table at time $t = 1$, and that ball 2 reaches it at time $t = 0.5$. Here, there are two independent events, $ReachEdge_1$ (ball 1 reaches the edge) and $ReachEdge_2$ (ball 2 reaches the edge). Let us assume that the edge absorbs all the energy of the balls and that they stop at the point of collision. In these circumstances, the evolution of the billiards world is completely determined. There is no uncertainty. Thus, what we should conclude is that *the only* event that is possible in S_0 is $ReachEdge_2$, and that the only event possible in $do(ReachEdge_2, S_0)$ is $ReachEdge_1$. Thus, $ReachEdge_1$ is impossible in S_0. Therefore, if we eliminate from the tree of situations all those situations that are reached by performing impossible events, we end up with a single line of possible situations. This should not be surprising, since it is a reflection of the fact that the laws of physics along with the initial conditions completely determine the evolution of the world (at least for the kind of models we are working with). Hence, the only way in which we could obtain a branching future is by introducing a form of incompleteness with regards to the actions that may occur. This incompleteness arises when there are *agents* that can choose from a gamut of actions. Each choice is represented by a branch in the situation tree.

Thus, in regards to this chapter, we understand a *natural process* to be a process whose behavior is completely determined from:

- A description of the initial conditions of the system.
- The physical laws governing the phenomenon.

All events whose occurrences are dictated by these physical laws will be called *natural*. In this section, we present an axiomatization of natural events to characterize the properties we just described. Basically, our view of natural processes is reflected on the structure of the situation calculus trees in the following way: In each situation there is at most one natural action or event possible. This follows from the axiomatization presented below.

6.3.2 Axiomatization.

In this section we present an approach to axiomatize problems involving natural phenomena. We make several assumptions that greatly simplify the presentation. In particular, we assume that no natural events occur concurrently. In general, this assumption is too strong, and in section 6.4 we discuss how to extend the approach to allow for concurrency.

In this axiomatization we distinguish natural events from the other events (or actions) introducing the predicate $natural \subseteq \mathcal{A}$. The most important characteristic of natural events is that when the conditions for their occurrence arise, they must occur. In this framework, a complete specification of the predicate *natural* has to be provided by the axiom writer.

To describe some important properties of natural actions we introduce a new predicate $\Pi_t \subseteq \mathcal{A} \times \mathcal{S} \times \mathcal{T}$ which is domain dependent. $\Pi_t(a, s, t)$ is true whenever the conditions at the time $start(s)$ are such that at time t action a would occur. For example, given the initial situation in figure 6.1a, the literal $\Pi_t(Collision, S_0, 0.5)$ should be true. That is, in S_0 the conditions are such that if things follow the course predicted at the start of S_0, then a collision is possible at time 0.5. Furthermore, the collision can be guaranteed to occur if no other events occur before time 0.5.

In the same manner as with *natural*, we assume that we have a complete axiomatization for Π_t. Based on the predicates just introduced, we now describe some essential properties of natural events.

First, if a situation s starts at time t_s, then the situation s would be ended by some natural action a at the first time greater than t_s for which $\Pi_t(a, s, t)$ was true. This is written as follows:

$$natural(a) \wedge \Pi_t(a, s, t) \wedge \neg(\exists\, t')[start(s) < t' < t \wedge \Pi_t(a, s, t')] \supset end(s, a) = t. \tag{6.15}$$

Also, we use the predicate Π_t to completely characterize the predicate *Poss*. Thus, unlike the case with discrete non-*natural* actions, we may write a general form for a definition of *Poss* for natural events:

$$\begin{aligned} natural(a) \supset Poss(a,s) \equiv [&\Pi_t(a,s,end(s,a)) \wedge \\ &\neg(\exists\, a',t').natural(a') \wedge \Pi_t(a',s,t') \wedge start(s) < t' \leq end(s,a)]. \end{aligned} \tag{6.16}$$

That is, natural action a is possible in state s if, given the conditions at the start of s, the action is possible[3] at the time $end(s,a)$, and no other natural action could have occurred between the start of s and $end(s,a)$. For instance, in the example of figure 6.2, $\Pi_t(ReachEdge_1, S_0, 1)$ should be true but $Poss(ReachEdge_1, S_0)$ should not. On the other hand, $\Pi_t(ReachEdge_2, S_0, 0.5)$ and $Poss(ReachEdge_2, S_0)$ should be true.

Another essential aspect of any *natural* action a is described by the following axiom:

$$[natural(a) \wedge Poss(a,s) \wedge Poss(a',s) \wedge \neg natural(a')] \supset end(s,a') < end(s,a). \tag{6.17}$$

Thus, if a is a *natural* action that is possible in situation s, then no other action is possible after the time at which a would occur (i.e., $end(s,a)$). For instance, in the example of figure 6.2, if an action is possible in S_0, it must bring S_0 to an end at a time before 0.5 (i.e., $(\forall\, a).Poss(a,S_0) \supset end(S_0,a) \leq 0.5$).

An immediate consequence of the above axioms is:

Observation 6.1

$$natural(a) \wedge natural(a') \wedge Poss(a,s) \wedge Poss(a',s) \supset a = a'.$$

Thus, at most one natural action is possible in a given situation.

Finally, assume that some natural event a is possible in a situation s. Furthermore, assume that nothing occurs before $end(s,a)$. Under these assumptions, the event a must occur. This is formalized as follows:

$$natural(a) \wedge Poss(a,s) \wedge \neg(\exists\, a')\ [occurs(a',s) \wedge end(s,a') < end(s,a)] \supset occurs(a,s). \tag{6.18}$$

For example, in the billiards ball scenario of figure 6.1, if nothing occurs prior to time 0.5, we infer that the *Collision* occurs.

The structure of logical theories for natural processes is similar to the structure for theories with occurrences described in chapter 4. In fact, let Σ_{nat} denote one of these theories. Σ_{nat} will contain:

- Σ_g. General axioms for the situation calculus. These are: basic axioms of section 2.1. Axioms (4.1)-(4.11) and (4.23) which describe the structure of occurrences. Axiom (6.3), which establishes the properties of the function $\doteq$. Situation independent axioms (6.4)-(6.11). Axioms (6.15)-(6.18) that constrain the structure of the situation tree and the assignment of starting times to situations.

- Σ_d, state independent axioms.

- T_{pos}. Same as the set T_{pos} for the theories studied in earlier chapters, except that no *Poss* axioms are provided for natural events.

[3]As described by Π_t.

- T_{occ}. Same as the set T_{occ} for the theories studied in chapter (4).
- T_{Π_t}. In order to describe the conditions under which natural actions are possible, we appeal to the predicate *Poss*. Which is true of a situation s and an action a whenever the preconditions for the execution of a in s hold. However, as discussed before, *Poss* may be completely characterized for natural actions after Π_t is given. Therefore, we include in T_{Π_t} a set of sentences that completely specifies Π_t.
- T_{ef}, effect axioms.
- T_{S_0}, specification of the initial conditions.
- T_{sc}, state constraints.

To illustrate the approach, we make use of the ball in the shaft example, originally proposed by Erik Sandewall [50]. In this example, a ball is moving along a horizontal surface at a positive speed towards a shaft (see figure 6.3). If the ball reaches the edge of the shaft, it will fall into it with acceleration G. In this scenario, there is only one object: the ball, which is considered point-like. The vertical axis (V) is taken as pointing downwards, whereas the horizontal axis (H) points to the right. The coordinates of the relevant points are indicated in the figure as pairs (H, V). In order to write a theory to describe this scenario, we introduce the action constants *Fall*, *HitWall* and *HitBottom*, and the parameter constants P_h and P_v.

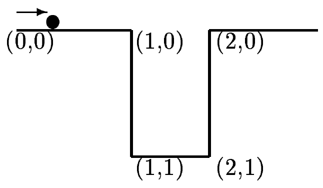


Figure 6.3: The ball in the shaft.

We axiomatize Sandewall's example as follows:

- Σ_d: There are only three possible events, thus[4]:

$$(\forall a).a = \mathit{Fall} \vee a = \mathit{HitWall} \vee a = \mathit{HitBottom}. \tag{6.19}$$

  Also, all the events in this problem are natural:

$$(\forall a)\ natural(a). \tag{6.20}$$

  Furthermore, the only parameters in this problem are P_h, P_v for the position of the ball with respect to the H and V axes. Also, the parameters are distinct:

$$P_h \neq P_v.$$

- T_{pos}. In this problem, all actions are natural. Thus, no domain specific axioms about *Poss* are given.

[4]Remember that we assume unique names axioms for actions.

- T_{Π_t}.

  The conditions for *Fall* to occur are that the vertical position of the ball be 0, and that the horizontal motion of the ball be such that an edge of the shaft is reached while moving in the direction of the shaft:

$$\begin{aligned}\Pi_t(Fall, s, t) \equiv (\exists\, y_h).holds(P_h \doteq y_h, s) \wedge holds(P_v \doteq constant(0), s) \wedge \\ [q(y_h, t) = 1 \wedge q(d(y_h), t) > 0 \vee q(y_h, t) = 2 \wedge q(d(y_h), t) < 0].\end{aligned} \tag{6.21}$$

  We must emphasize that this axiomatization is very simplified. A better axiomatization would consider the sum of the forces on the ball as the determining factor for the ball's motion. In future research we will attempt to provide a more general theory that incorporates these views.

  In (6.21), we appeal to the fact that if y names the function that describes the position of an object, then $d(y)$ would denote its speed.

  The ball hits the wall if its vertical position is inside the shaft and its horizontal position coincides with one of the walls of the shaft and its motion is towards that wall.

$$\begin{aligned}&\Pi_t(HitWall, s, t) \equiv \\ &\quad (\exists\, y_h, y_v).holds(P_h \doteq y_h, s) \wedge holds(P_v \doteq y_v, s) \wedge q(y_v, t) > 0 \wedge \\ &\quad [q(y_h, t) = 1 \wedge q(d(y_h), t) < 0 \vee q(y_h, t) = 2 \wedge q(d(y_h), t) > 0].\end{aligned} \tag{6.22}$$

  Finally, a *HitBottom* event occurs when its coordinates coincide with some point along the bottom of the shaft and it is moving towards it.

$$\begin{aligned}\Pi_t(HitBottom, s, t) \equiv (\exists\, y_h, y_v).holds(P_h \doteq y_h, s) \wedge holds(P_v \doteq y_v, s) \wedge \\ [q(y_h, t) \geq 1 \wedge q(y_h, t) \leq 2 \wedge q(d(y_v), t) > 0 \wedge q(y_v, t) = 1].\end{aligned} \tag{6.23}$$

- T_{ef}. When the ball hits a wall, it bounces without losing any energy. Thus, its vertical motion is unaffected, while its horizontal motion changes direction. This effect is modeled by changing the sign of the ball's speed, while maintaining the other parameters:

$$\begin{aligned}&Poss(HitWall, s) \wedge (\exists\, y_h)\ [holds(P_h \doteq y_h, s) \wedge d(y_h') = constant(-1) \times d(y_h) \wedge \\ &\quad q(y_h', end(s, HitWall)) = q(y_h, end(s, HitWall))] \supset holds(P_h \doteq y_h', do(HitWall, s)).\end{aligned}$$

  The *Fall* does not affect the horizontal motion of the ball. However, it starts a free fall, as reflected by a change in its vertical motion:

$$Poss(Fall, s) \supset holds(P_v \doteq quadratic(0, 0, G), do(Fall, s)).$$

  In this effect axiom, we set the arguments for y_v assuming that the initial conditions for the fall are that the height and vertical velocity are zero. A more general effect axiom would state that these initial conditions are determined by the value of the parameters in the previous situation.

  When the ball hits the bottom of the shaft (*HitBottom*), all its energy is absorbed. Thus, as a result, motion in both directions stops.

$$\begin{aligned}&Poss(HitBottom, s) \wedge (\exists\, y_h)\ [holds(P_h \doteq y_h, s) \wedge \\ &\quad y_h' = constant(q(y_h, end(s, HitBottom)))] \supset holds(P_h \doteq y_h', do(HitBottom, s)). \\ &Poss(HitBottom, s) \wedge (\exists\, y_v)\ [holds(P_v \doteq y_v, s) \wedge \\ &\quad y_v' = constant(q(y_v, end(s, HitBottom)))] \supset holds(P_v \doteq y_v', do(HitBottom, s)).\end{aligned}$$

These effect axioms can be easily rewritten to conform to the standard form of positive (2.27) and negative effect axioms (2.28). In fact, we can combine them to obtain a single positive effect axiom of the form (2.27) as described later.

- T_{S_0}:

$$holds(P_h \doteq linear(1,0), S_0) \wedge holds(P_v \doteq constant(0), S_0). \tag{6.24}$$

- T_{sc} In the example, we do not require domain specific state constraints. However, to illustrate their use, assume that we introduce parameters to denote the speed of the ball with respect to each axis (e.g. $\dot{P}_v$). In this case, the following would be a state constraint:

$$holds(\dot{P}_v \doteq d(y), s) \equiv holds(P_v \doteq y, s).$$

6.3.3 Natural Processes and the Frame and Ramification Problems.

So far, we have presented an extension of the situation calculus that allows us to write theories about natural processes. A question that arises is whether or not this extension can be adequately integrated with the solution to the frame and ramification problems studied in chapters 2 and 3. To argue that this is indeed the case, we simply show that axiomatizations of natural processes are a special case of the theories of action with occurrences discussed in 4.

First, the set Σ_d of state independent domain axioms are present in both types of theories. The general axioms (6.4)-(6.11), can be considered part of Σ_d.

In Σ_{nat}, the axioms in $T_{pos} \cup T_{\Pi_t}$ along with general axiom (6.16) play the role of T_{pos} in the theories of action with occurrences. That is, these axioms specify the predicate *Poss*.

The axioms in the sets T_{occ} of occurrence axioms, T_{ef} of effect axioms for natural events, and T_{S_0} of initial conditions have the same form as in previously studied theories of action. The only difference, is that we include general axiom (6.18) as an occurrence axiom.

The axioms in T_{sc} are as before. Therefore, if the state constraints are of the kind discussed in chapter 3, we will be able to derive successor state axioms from T_{sc} and T_{ef}. Also, axiom (6.3) is included in the set T_{sc}. Fortunately, the contrapositive form of this axiom corresponds to a binary state constraint:

$$[\neg(\forall t)\ q(y_1,t) = q(y_2,t)] \supset \neg holds(p \doteq y_1, s) \vee \neg holds(p \doteq y_2, s). \tag{6.25}$$

Clearly, this axiom is a constraint of the form (3.1). Therefore, it can be used to obtain new effect axioms following the procedure of chapter 3.

Finally, axioms (6.15) and (6.17) establish constraints on the temporal relationships between situations in the situation calculus tree, and can be considered part of the basic axioms of the theories of action.

The Ball in the Shaft.

Here, we show how the solution to the frame and ramification problems is applied to the ball in the shaft example.

First, we rewrite the effect axioms as a single positive effect axiom like (2.27):

$$Poss(a,s) \wedge \gamma^{+}_{p \doteq y}(a,s) \supset holds(p \doteq y, do(a,s)), \tag{6.26}$$

where $\gamma^{+}_{p \doteq y}(a,s)$ denotes the formula:

$$\begin{aligned}
&a = \mathit{HitWall} \wedge (\exists\, y_h)\ [\mathit{holds}(P_h \doteq y_h, s) \wedge d(y) = \mathit{constant}(-1) \times d(y_h) \wedge \\
&\qquad q(y, \mathit{end}(s, \mathit{HitWall})) = q(y_h, \mathit{end}(s, \mathit{HitWall}))] \vee \\
&a = \mathit{Fall} \wedge p = P_v \wedge y = \mathit{quadratic}(0,0,G) \vee \\
&a = \mathit{HitBottom} \wedge (\exists\, y_h)\ [p = P_h \wedge \mathit{holds}(P_h \doteq y_h, s) \wedge \\
&\qquad\qquad y = \mathit{constant}(q(y_h, \mathit{end}(s, \mathit{HitBottom})))] \vee \\
&a = \mathit{HitBottom} \wedge (\exists\, y_v)\ [p = P_v \wedge \mathit{holds}(P_v \doteq y_v, s) \wedge \\
&\qquad\qquad y = \mathit{constant}(q(y_v, \mathit{end}(s, \mathit{HitBottom})))].
\end{aligned}$$

Also, from (6.25) and (6.26), we obtain the effect axiom:

$$\mathit{Poss}(a,s) \wedge \gamma^{+}_{p \doteq y}(a,s) \wedge \neg(\forall\, t)\ q(y,t) = q(y',t) \supset \neg \mathit{holds}(p \doteq y, \mathit{do}(a,s)).$$

With respect to the ball in the shaft example, this means that the ball may not follow two different trajectories. To obtain the set of successor state axioms for each of the fluents, we follow the same procedure outlined in chapter 3. We obtain:

$$\begin{aligned}
\mathit{Poss}(a,s) \supset \mathit{holds}(p \doteq y, \mathit{do}(a,s)) \equiv \gamma^{+}_{p \doteq y}(a,s) \vee \mathit{holds}(p \doteq y, s) \wedge \\
\neg[\gamma^{+}_{p \doteq y}(a,s) \wedge \neg(\forall\, t)\ q(y,t) = q(y',t)].
\end{aligned} \tag{6.27}$$

Given that in the ball in the shaft example we assume that there are no external agents, we should be able to determine what happens with the ball at all times. In fact, we can show that:

Proposition 6.1 *In the theory Σ_{nat} for the ball in the shaft example, it follows that:*

$$\mathit{occurs}(\mathit{Fall}, S_0), \tag{6.28}$$

$$\mathit{occurs}(\mathit{HitBottom}, s) \supset \neg(\exists\, a)\ \mathit{occurs}(a, \mathit{do}(\mathit{HitBottom}, s)), \tag{6.29}$$

$$(\forall\, s).\mathit{occurs}(a,s) \wedge s > S_0 \supset a = \mathit{HitWall} \vee a = \mathit{HitBottom}. \tag{6.30}$$

Proof: (6.28) states that the first occurrence is a *Fall* event. The proof is as follows: From (6.21) and (6.24) we obtain:

$$\begin{aligned}
\Pi_t(\mathit{Fall}, S_0, t) \equiv [q(\mathit{linear}(1,0), t) = 1 \wedge q(d(\mathit{linear}(1,0)), t) > 0 \vee \\
q(\mathit{linear}(1,0), t) = 2 \wedge q(d(\mathit{linear}(1,0)), t) < 0].
\end{aligned} \tag{6.31}$$

From (6.8) we know that $\mathit{linear}(1,0) = t$, and from (6.11) $d(\mathit{linear}(1,0)) = \mathit{constant}(1)$. Thus, it follows that we can rewrite (6.31) as:

$$\Pi_t(\mathit{Fall}, S_0, t) \equiv [t = 1 \wedge q(\mathit{constant}(1), t) > 0 \vee t = 2 \wedge q(\mathit{constant}(1), t) < 0].$$

Which can be rewritten as:

$$\Pi_t(\mathit{Fall}, S_0, t) \equiv [t = 1 \wedge 1 > 0 \vee t = 2 \wedge 1 < 0].$$

At this point in the proof, we need to ask our Oracle whether:

$$(\exists\, t).y = 1 \wedge 1 > 0 \vee t = 2 \wedge 1 < 0.$$

To which the Oracle will reply *true* and give an assignment $t = 1$. Then we may ask whether:

$$(\exists\, t).t \neq 1 \wedge [y = 1 \wedge 1 > 0 \vee t = 2 \wedge 1 < 0].$$

To which the Oracle would reply *false*. Thus, we conclude that:

$$\Pi_t(Fall, S_0, t) \equiv t = 1.$$

Also, from (6.15), we conclude that:

$$end(S_0, Fall) = 1.$$

Now, from (6.22):

$$\Pi_t(HitWall, s, t) \equiv q(constant(0), t) > 0 \wedge [t = 1 \wedge 1 < 0 \vee t = 2 \wedge 1 > 0].$$

By consulting the Oracle we know that $0 > 0$ is false, therefore $\Pi_t(HitWall, S_0, t)$ is also false. A similar proof leads us to conclude that $\Pi_t(HitBottom, S_0, t)$ is false. Therefore, we arrive at the conclusion that:

$$\Pi_t(a, S_0, t) \equiv a = Fall \wedge t = 1.$$

From (6.16), (6.19) and (6.20), we infer that $Poss(a, S_0) \equiv a = Fall$. Furthermore, from (6.18), it follows that $occurs(Fall, S_0)$.

To prove (6.28), it is simple to show (from T_{Π_t}) that after a *HitBottom* occurs in s:

$$\Pi_t(a, do(HitBottom, s), t) \equiv false.$$

We omit the proof.

Finally, to prove (6.28) we use the induction axiom (2.1) with:

$$\begin{aligned}\varphi(s) = occurs(a, s) \wedge s > S_0 \supset a = HitWall \wedge \\ [holds(y_v, do(HitWall, s)) \equiv holds(y_v, do(Fall, S_0))] \vee a = HitBottom.\end{aligned}$$

$\varphi(S_0)$ is vacuously true. Assuming that for an arbitrary S, $\varphi(S)$ is true, we need to show that $\varphi(do(a', S))$ is also true, for an arbitrary a'. This is trivial to show by using the effect axioms and appealing to the Oracle.

□

6.3.4 Relationship to Sandewall's approach.

Conceptually, the work presented in this chapter is related to Sandewall's [50, 51]. Sandewall proposes to combine logic and differential equations for describing *real world* systems. In Sandewall's words:

> The method applies to *piecewise continuous* physical systems, i.e. systems whose possible histories over time may contain a number of significant time-points called *breakpoints*, and where all parameters of the system are assumed to be continuous in the intervals between breakpoints. [50, p.412]

Sandewall's significant time-points correspond to the times at which events occur. However, in our approach we do not require all parameters to be continuous in the intervals between breakpoints. Rather, we require that the parameters be fully described by some function of time.

Another essential difference between Sandewall's approach and ours is that he appeals to non-monotonic reasoning in order to study the relationship between parameters before and after the breakpoints. In fact, the problem of determining what paramters are discontinuous at a breakpoint is a special form of the frame problem. Sandewall's solution to the frame problem is a variant of Yoav Shoham's chronological minimization [58] which is called *Chronological Minimization of Discontinuities.*

Unfortunately, as shown by Rayner [43], Sandewall's approach fails to sanction the intuitively correct answers in some simple problems. Sandewall's approach is such that if two models coincide in the changes that are produced up to some time point t, then the model that is preferred is the one that has a minimal set of discontinuous parameters at t (regardless of what the models say about the parameters at times after t, hence the term chronological). The problem arises when there are multiple minimal models. Rayner's example is a variation of the ball in the shaft problem studied earlier. In his example, in the bottom half of the shaft there is a radiation field. The radiation field changes the pattern of heat exchange between the ball and its environment. Thus, if the ball enters the radiation field, then the parameter describing the temperature of the ball will suffer a discontinuous change. The problem is that when the ball reaches the field's boundary it may either bounce (changing its motion but not the nature of the heat exchange), or it may cross the field (changing the nature of the heat exchange but not its motion). Thus, the chronological minimization of discontinuities cannot distinguish between the two models. However, the intended conclusion corresponds to the model in which the ball enters the radiation field.

It is important to point out that the deficiency that Rayner found on Sandewall's approach is due to an inadequate solution to the frame problem and not, we believe, to the conceptual underpinnings of his approach. In fact, if we use Reiter's solution to the frame problem, Rayner's problem does not arise. Basically, each breakpoint is associated with some event, whose effects are described using *effect axioms.* Thus, in the previous example, the event of *reaching the boundaries of the radiation field* will have effects on the thermal parameters of the ball but not on its motion. Hence, the successor state axioms will not relate such an event to a change in the ball's motion.

6.4 Concurrency, Continuity and Natural Processes.

So far, we have discussed continuity and natural processes in the context of non-concurrent events. This has allowed us to simplify the presentation of our approach. However, in practice, it is important to be able to consider actions or events that occur concurrently. There are several minor complications that arise when considering natural concurrent events. Basically, we need to re-study and replace axioms (6.15)-(6.18) in the context of an extended language that incorporates the notion of concurrent actions. Thus, we extend the language as indicated in section (5.2) with the predicates $occurs^c$ and $\in$, along with the action function symbol $+$. With respect to concurrent natural actions, we need to add the following axiom:

$$natural(a + a') \equiv natural(a) \wedge natural(a'). \tag{6.32}$$

Thus, for any pair of natural actions a and a', the concurrent action $a + a'$ is also natural.

Since the axioms for concurrent actions we provided before were meant to describe concurrency between non-natural actions, we need to qualify axiom (5.7). Thus, we replace it with:

$$\neg natural(a_1) \vee \neg natural(a_2) \supset [Poss(a_1 + a_2, s) \equiv Poss(a_1, s) \wedge Poss(a_2, s) \wedge \neg precInt(a_1, a_2)]. \tag{6.33}$$

Now, axiom (6.15) has to be replaced with:

$$natural(\alpha) \wedge a \in \alpha \wedge \Pi_t(a, s, t) \wedge \neg(\exists\, t')[start(s) < t' < t \wedge \Pi_t(a, s, t')] \supset end(s, \alpha) = t. \tag{6.34}$$

On the other hand, axiom (6.16) defines *Poss* for non-concurrent natural actions. The corresponding axiom for concurrent natural actions is more complex. We still want to provide necessary and sufficient conditions for *Poss* to be true of some natural concurrent action. The axiom is as follows:

$$\begin{aligned} natural(\alpha_n) \supset Poss(\alpha_n, s) \equiv & \\ & (\forall\, a)[natural(a) \wedge primitive(a) \supset [a \in \alpha_n \equiv \Pi_t(a, s, end(s, \alpha_n))]] \wedge \\ & \neg(\exists\, a', t').natural(a') \wedge primitive(a') \wedge \Pi_t(a', s, t') \wedge start(s) < t' < end(s, \alpha_n). \end{aligned} \tag{6.35}$$

This axiom is similar to (6.16), it only adds the condition that all natural actions that may occur concurrently must be present in α_n for it to be possible. Thus, if α_n is a concurrent natural action, and α_n is possible in a situation s, then all primitive natural actions that are possible (according to Π_t) at time $end(s, \alpha_n)$ must be in α_n. Also, no other natural action could be possible before $end(s, \alpha_n)$.

Also, we replace (6.17) with:

$$[natural(a) \wedge Poss(\alpha, s) \wedge Poss(\alpha', s) \wedge a \in \alpha] \supset end(s, \alpha') \leq end(s, \alpha). \tag{6.36}$$

Thus, given that *Poss* is completely characterized, it establishes conditions on the temporal relationships of possible situations. This axiom can also be read as stating that for a concurrent action to be possible in a situation s, it must terminate s before any natural action would.

Finally, we replace (6.18) with:

$$natural(a) \wedge Poss(a, s) \wedge \neg(\exists\, a')\ [occurs(a', s) \wedge end(s, a') < end(s, a)] \supset occurs^c(a, s). \tag{6.37}$$

This axiom is a simple variation of (6.18). It states that if nothing happens before the time in which some natural action is possible, then the natural action must occur.

As a simple example, consider the scenario of figure 6.2, but with different initial conditions. The axiomatization is given below. We introduce the action constants *ReachEdge*1, *ReachEdge*2 and *Catch*1, the parameter constants X^1, Y^1, X^2, and Y^2,

- Σ_d, state independent axioms:

$$(\forall\, a).primitive(a) \equiv a = ReachEdge1 \vee a = ReachEdge2 \vee a = Catch1, \tag{6.38}$$

$$(\forall\, a).primitive(a) \supset natural(a) \equiv a = ReachEdge1 \vee a = ReachEdge2, \tag{6.39}$$

$$PrecInt(Catch1, ReachEdge1). \tag{6.40}$$

  Thus, *Catch*1 is the only non-natural action.

- T_{pos}:

$$Poss(Catch1, s).$$

  Thus, it is always possible to perform *Catch*1.

- T_{Π_t}:

$$\Pi_t(\mathit{ReachEdge1}, s, t) \equiv (\exists\, y_x)\ \mathit{holds}(X^1 \doteq y_x, s) \wedge q(y_x, t) = 2 \wedge q(d(y_x), t) > 0, \tag{6.41}$$
$$\Pi_t(\mathit{ReachEdge2}, s, t) \equiv (\exists\, y_x)\ \mathit{holds}(X^2 \doteq y_x, s) \wedge q(y_x, t) = 2 \wedge q(d(y_x), t) > 0. \tag{6.42}$$

- T_{ef}:

$$\begin{aligned}
&\mathit{Poss}(\alpha, s) \wedge \mathit{Catch1} \in \alpha \wedge \mathit{holds}(X^1 \doteq y_x, s) \wedge \mathit{holds}(Y^1 \doteq y_y, s) \supset \\
&\qquad \mathit{holds}(X^1 \doteq \mathit{constant}(q(y_x, \mathit{end}(s, \alpha))), \mathit{do}(\alpha, s)) \wedge \\
&\qquad \mathit{holds}(Y^1 \doteq \mathit{constant}(q(y_y, \mathit{end}(s, \alpha))), \mathit{do}(\alpha, s)), \\
&\mathit{Poss}(\alpha, s) \wedge \mathit{ReachEdge1} \in \alpha \wedge (\exists\, y_x)\ [\mathit{holds}(X^2 \doteq y_x, s) \wedge q(d(y_x), \mathit{end}(s, \alpha) > 0)] \supset \\
&\qquad \mathit{holds}(X^1 \doteq \mathit{constant}(1), \mathit{do}(\alpha, s)), \\
&\mathit{Poss}(\alpha, s) \wedge \mathit{ReachEdge2} \in \alpha \wedge (\exists\, y_x)\ [\mathit{holds}(X^2 \doteq y_x, s) \wedge q(d(y_x), \mathit{end}(s, \alpha) > 0)] \supset \\
&\qquad \mathit{holds}(X^2 \doteq \mathit{constant}(1), \mathit{do}(\alpha, s)).
\end{aligned}$$

  In the last two effect axioms, the geometry of the table is implicit in the constant utilized (i.e., using the term $\mathit{constant}(1)$).

- T_{S_0}:

$$\begin{aligned}
&\mathit{holds}(Y^1 \doteq \mathit{constant}(0.4), S_0) \wedge \mathit{holds}(Y^2 \doteq \mathit{constant}(0.6), S_0), \\
&\mathit{holds}(X^1 \doteq \mathit{linear}(0.5, 0), S_0) \wedge \mathit{holds}(X^2 \doteq \mathit{linear}(0.5, 0), S_0).
\end{aligned}$$

  Thus, both balls start at the same X coordinate and with the exact same motion towards the right end of the table. Given these conditions, the balls will arrive at the right edge at the same time, unless $\mathit{Catch1}$ is performed before time 2.

- T_{sc}: None.

For this example, and given the axiomatization for concurrent natural processes, we have:

Proposition 6.2

$$\neg \mathit{Poss}(\mathit{ReachEdge1}, S_0), \tag{6.43}$$
$$\mathit{Poss}(\mathit{ReachEdge1} + \mathit{ReachEdge2}, S_0), \tag{6.44}$$
$$\neg \mathit{Poss}(\mathit{ReachEdge1} + \mathit{ReachEdge2} + \mathit{Catch1}, S_0), \tag{6.45}$$
$$\mathit{end}(S_0, \mathit{Catch1}) < 1. \tag{6.46}$$

Proof: From T_{Π_t}, T_{S_0}, along with the Oracle, it follows that:

$$\Pi_t(\mathit{ReachEdge1}, S_0, t) \equiv t = 2, \tag{6.47}$$
$$\Pi_t(\mathit{ReachEdge2}, S_0, t) \equiv t = 2. \tag{6.48}$$

From (5.5), we infer that $\mathit{ReachEdge2} \notin \mathit{ReachEdge1}$. Also, (6.34) implies that

$$\mathit{end}(S_0, \mathit{ReachEdge1}) = 2.$$

These facts, along with (6.47), (6.47) and (6.35) lead us to conclude (6.43).

From (6.32) it follows that $ReachEdge1 + ReachEdge2$ is a natural action. Hence, a direct application of (6.35), leads us to conclude that (6.43) holds.

From (6.38) and (5.8), it follows that (6.43) holds as well. Conceptually, the reason for the impossibility of executing $ReachEdge1+ReachEdge2+Catch1$ is that $Catch1$ and $ReachEdge1$ are said to have interacting preconditions ($precInt$), making it impossible to execute them concurrently.

Finally, (6.43) follows from (6.36) and (6.43).

□

6.5 Discussion.

In this chapter we have presented a novel approach to deal with continuity within the situation calculus. Also, we presented an axiomatization to model natural phenomena utilizing the concept of occurrences introduced in an earlier chapter. The greatest advantage of the approach presented is that it builds upon a solid foundation based on the situation calculus. In particular, we make use of Reiter's solution to the frame problem, as well as of Lin and Reiter's approach to the ramification problem.

The enriched situation calculus is an attempt to provide a formal foundation for the specification of dynamic systems with continuous variables. Also, we believe that this language can be used to address many of the representational concerns in the area of qualitative reasoning. We have not addressed these issues directly and they are left for future investigation.

There are a number of issues that we have not addressed and which merit some further research. We briefly discuss some of these below.

First, an essential limitation of our approach is that the set of times at which events or actions occur have to be discrete. Otherwise, the induction axiom for situations would be violated. Furthermore, the approach presented is limited by the power of the Oracle and the expressive power of the interaction language. It would be clearly advantageous to study the power of existing systems for dealing with real equations and inequalities. For instance, until we properly characterize the behavior of the Oracle, it is not possible to describe the computational properties of the task of reasoning with continuous systems.

In particular, recall that when the Oracle is asked whether or not an expression of the type (6.1):

$$\Sigma_{\mathcal{R}} \models (\exists\, \mathbf{x})\ \Phi(\mathbf{x}).$$

is true, the Oracle answers "false" or it responds with an assignment for **x** that makes the expression "true". However, in order to predict the future behavior of a dynamic system, we are interested in obtaining an answer to the following question: *given that*

$$(\exists\, t)\ \phi(t) \tag{6.49}$$

is true, what is the smallest non-negative real number that satisfies (6.49)? Such a question arises when we use axiom (6.16) to determine what is the next time that some natural action is expected to occur. However, the query language that we have defined for the Oracle is not powerful enough to express such a request. Given these expressive limitations, it turns out that there is no bound in the number of queries necessary to determine when the next action may occur. This arises if (6.49) has an infinite number of solutions over a finite interval of time.

The evaluation of the predicate $Poss(a, s)$ for natural actions using (6.16) involves finding the earliest time that satisfies a sentence of the form (6.49) along with evaluating some fluents in the state s. Thus, the evaluation of $Poss$ for a given natural action in a situation whose state is known takes a time proportional to the time it takes to find the earliest t for which a formula of the form (6.49) is satisfied. Furthermore, given the state of a situation, determining the next natural action that is possible takes time that is proportional to the the number of distinct natural actions in the domain, times the time it takes to evaluate $Poss$ for some natural action in a known state.

On the surface, it appears that there are many problems that cannot be addressed using the extension of the situation calculus studied here. For example, consider the problem of modeling the bouncing of a rubber ball. If the behavior of the ball is modeled in such a way that Zeno's paradox arises [12], we may face a contradictory theory. In fact, if the ball bounces an infinite number of times in a finite time interval, and if we model each bounce of the ball as a natural event, then there will be a line of situations that contains an infinite number of situations lapsing a finite time. Problems arise if we state that some situation exists after the ball stops bouncing. In fact, the infinite number of bounces (each one represented as an occurrence), coupled with an existing situation after the ball stops, is reflected in a structure of situations that is not consistent with the induction axiom (2.1).

A possible way around the problem mentioned above, is to model the bouncing of the ball with a single function of time and without treating the *bounce* as an event. Thus, we would have a real function of time completely describing the behavior of the ball with its many bounces. Furthermore, we would state that there is a single situation in which the ball follows the trajectory dictated by this function. Clearly, this solution is not general. In fact, a function to describe the complete behavior of the system may not be available. Furthermore, one of the reasons to model natural phenomena as sequences of natural events is to avoid having to specify real functions to model the entire behavior of the dynamic system.

Unfortunately, even if we model the behavior of the system as described above, we still need to ensure that the axioms for natural processes do not introduce inconsistencies with the induction axiom for situations. It seems that the only way to do this is by providing extra-logical proofs that infinite series of natural events in finite time may not arise.

Another issue that we did not address has to do with inherent uncertainty. In fact, we stated that we only deal with processes whose mathematical models are completely determinate. Thus, we assumed that, given a complete specification of the initial situation, there is only one way in which the world may evolve. As we saw before, this is reflected in the situation tree by the fact that, in the absence of non-natural actions, all possible events line up in a single branch. This reflects a view of the world in which there is no uncertainty. However, if we want to model situations in which there is uncertainty with regards to which natural events may arise, we can simply allow for more than one natural action to be possible in some situations.

There are several problems regarding continuous properties that we have not addressed. For instance, we may want to provide a taxonomy of properties of objects that must obey certain continuity properties. For example, we may want to state that the derivative of the function that describes the position of an object in space must always exist, except in break points (e.g., points in which some action abruptly modifies the behavior of the parameter). Problems of these nature are analyzed by Sandewall [50] in the framework of another temporal logic. A problem that arises is that we might need to introduce inter-state constraints, which we have not considered. For example, we might want to state that for any action a and situation s, the position of an object at the end of s must be the same as the position at the start of $do(a, s)$. Thus, adding a more general description of physical phenomena may require us to re-study the solution to the frame and ramification problems in the presence of inter-state constraints.

Chapter 7

The Situation Calculus and Other Temporal Logics.

7.1 Introduction and Motivation.

In the early chapters of this work, we presented the *discrete situation calculus.* This logic is based on John McCarthy's original work in which he proposed the language of the situation calculus as a logical tool to write theories of action and time. Also within the framework of the situation calculus, we discussed how to approach two outstanding problems in knowledge representation research, namely the *frame* and *ramification* problems. Later on, we discussed how to extend this theoretical framework to address some important problems in logical representation of knowledge about action and time. In particular, we addressed the problem of knowledge representation with a linear temporal structure in a framework with branching time. We studied how to extend the ontology of actions to deal with concurrent actions, and we studied the problems of representing properties of the world that vary continuously with time. Some of the same issues raised in this work have been previously addressed within the framework of other temporal logics. In this chapter, we discuss the relationship that exists between the work presented here and some alternative frameworks for temporal reasoning. We study three different temporal logics, chosen because they represent the most widely utilized frameworks for reasoning about action and time in Knowledge Representation.

First, we discuss the *Interval Temporal Logic* proposed by James Allen [2, 3]. Basically, we discuss the issues that were the primary motivation for the development of the interval logic, and how their treatment contrasts with the approach used in this work.

Secondly, we carefully analyze Kowalski and Sergot's *calculus of events* [25], which is a theory of time based on logic programming. We argue that this approach has certain technical flaws when interpreted as a first order theory. Furthermore, we present a situation calculus based logic program as an alternative to the one based on the calculus of events.

Finally, we discuss the relationship between modal temporal logics of time and our version of the situation calculus. In particular, we show that the propositional modal temporal logic presented in [19] can be embedded within the situation calculus. This embedding suggests that much of the expressiveness of temporal logics of time is present in the situation calculus.

7.2 Interval Temporal Logic.

In [2, 3], James Allen proposed a linear temporal logic in which the structure of time is based on *intervals*. Thus, the primitive temporal element is an interval. Allen introduces a set of 13 binary relations that are possible among temporal intervals (e.g., equal, before, during, etc.). In the Interval Temporal Logic, the structure of time is such that temporal intervals can always be subdivided in sub-intervals, with the exception of *moments*. In this framework, moments are non-zero length intervals without internal structure. Thus, aside from moments, all other intervals can be subdivided into smaller subintervals. Allen argues that "the formal notion of a time point, which would not be decomposable, is not useful " [2, p. 834].

In later work, Allen and Hayes [5] argue that the difference between interval based and point based temporal structures is motivated by "different sources for intuitions." While the interval temporal logic is meant to model time as used in natural language, point-based time is used in the realm of classical physics.

Interestingly, the original situation calculus does not deal with the issue of point versus interval based time. In fact, no commitment is made with regards to time, other than it branches. In the extended situation calculus presented in this work, we use a point based temporal structure. Nonetheless, the lack of commitment about temporal *granularity* in the original situation calculus makes it possible to utilize an interval based model of time. However, our choice of temporal structure is partly motivated by our desire to integrate physical models of natural phenomena with logical theories of action based on the situation calculus. Obviously, this requires that time be modeled as isomorphic to the non-negative real numbers. Thus, we must impose a point based temporal structure in our logic.

Naturally, the question arises as to whether there is some loss of expressive power when choosing one structure of time over another. In [5], Allen and Hayes show that their logic accepts models in which the structure of time is such that intervals are modeled as ordered binary tuples of real numbers. Unfortunately, due to the properties of the reals, moments cannot be appropriately modeled, thus they cannot exist. Hence, if intervals are modeled as pairs of distinct real numbers, the temporal structure would satisfy the axioms of the interval temporal logic. Of course, it is also important to determine whether or not any temporal structure that satisfies the interval logic axioms can be mapped into the structure of the reals. It turns out that this is not generally the case. First, Allen and Hayes *moments* are incompatible with the view of intervals as ordered pairs of real numbers. Secondly, the interval logic is first order, thus it cannot be strong enough as required to characterize the reals. Later, in section (7.3.2), we present a characterization of intervals within the situation calculus. Given that we use a temporal structure isomorphic to the reals, this characterization cannot include the notion of *moment* as defined by Allen.

So far, our discussion has been centered on the characteristics of the structure of time in the interval temporal logic. Another important aspect of the logic has to do with the notions of events and actions. In [3], Allen discusses the issue of integrating a notion of event with the logic's temporal structure. Naturally, given the interval based structure of time of the logic, events have duration. Therefore, event occurrences may coincide, overlap, be one after the other, etc. Most of the work on this temporal logic concentrates on the logical specification of events, and provides a mechanism to specify the effects that events have on the world. However, until very recently [1], the frame problem had not been addressed. In [1], Allen and Ferguson address the frame problem by appealing to Schubert's explanation closure axioms. However, they do not offer, as Reiter does [47], a general mechanism to automatically generate frame axioms. Also, they do not address the ramification problem. Furthermore, for their approach to work, they require a complete list of the actions that are attempted. This way it can be concluded that nothing happens unless explicitly stated in the axioms of the theory.

There are several areas in which the interval temporal logic is more developed than the situation calculus. First, they offer interesting taxonomies of actions (e.g., *pushbutton* actions, *maintenance* actions, etc.[1]). They utilize the notion of *trying* to perform an action, as opposed to a simple execution of the action. Another interesting development is the study of planning algorithms within the interval temporal logic [6]. Unfortunately, this latter work seems to lack a careful analysis of soundness and completeness with respect to the underlying theory. It seems, however, that it would be very beneficial to study these issues in the realm of the situation calculus.

7.3 Calculus of Events.

7.3.1 A Critique.

The calculus of events was proposed as a general temporal logic framework. It was originally presented as a logic program [25] in which negation as failure plays an essential role. We have found it very difficult to ascribe a semantics to the calculus of events. In this section we point out some of the difficulties we have encountered with this proposal as a formal theory of time.

These difficulties arise from the use of *equality*, the use of *negation as failure*, and the interaction between negation as failure and incomplete knowledge.

As Kowalski and Sergot point out, their logic has been partly inspired by the interval temporal logic. Thus, intervals of time are an essential component of their ontology. However, the first problem that we encountered, arises from the treatment of temporal intervals. In the event calculus, two functions, `after` and `before`, are used to deal with time periods. Both are two-place functions that take an action and a fluent as arguments. According to Kowalski, a term of the form `after(a, f)` "names" a time period. Furthermore, according to Kowalski and Sergot [25, p.87], the sentence:

$$\texttt{Holds(p)} \tag{7.1}$$

"expresses that the relationship associated with `p` holds for the time period `p`." For example, we can write:

$$\texttt{Holds(after(paint(Green), colour(Green)))}$$

that can be taken to mean that `paint(Green)` is an event that starts a time period in which the fluent `colour(Green)` is true. Now, if `p1` and `p2` are two time periods which are started and ended by the same events, then they must be different names for the same temporal interval. Otherwise, `p` in (7.1) could not denote a *temporal interval*. As an example, consider figure 7.1, in which we

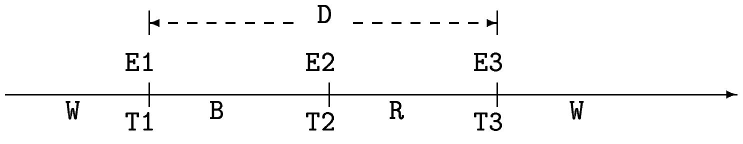


Figure 7.1: Simple Example

have three events `E1`, `E2` and `E3` occurring at times `T1`, `T2` and `T3` respectively. Moreover, suppose that we have four fluents: `W` which holds only before `T1` and after `T3`; `B` which holds only between `T1` and `T2`; `R` which holds only between `T2` and `T3`; and `D` which holds only between `T1` and `T3`.

Among others, the time period `after(E1,D)` holds; this is the same as the time periods ⟨T1 − T3⟩ and `before(E3,D)`. Now, from rule G3 in Kowalski and Sergot's program [25, p.88] we know that:

$$\texttt{start}(\texttt{after}(\texttt{e},\texttt{u}),\texttt{e}). \tag{7.2}$$

Therefore, we may infer that the time period `after(E1,R)` is started by `E1` at time `T1`. Now, we can easily provide axioms that make event `E2` terminate only `B` and nothing else. Therefore, the interval `after(E1,R)` would not be terminated by `E2`. However, we know that `E3` must terminate `R`. Therefore, we conclude that the time period denoted by `after(E1,R)` is terminated by `E3`. Hence, `after(E1,R)` is the time period ⟨T1 − T3⟩, which we know holds. We conclude that `Holds(after(E1,R))` must be true, which is clearly unintended.

This problem stems from the choice of ontology, and seems to be caused by the `Holds` predicate, which states that time periods hold. In [24, p. 142], Kowalski suggests that for a very specific class of problems "time periods can be eliminated altogether." This is suggested for computational, rather than representational reasons. However, it seems that time periods should be eliminated from the ontology for the proposal to work.

Another problem arises from the use of negation as failure. Assume that we query a temporal reasoner based on the calculus of events with `holdsAt(F,T)`, i.e. we want to know whether some fluent `F` holds at time `T`. Furthermore, assume that we obtain a negative answer. How are we to interpret this answer? In general, we want a clear characterization of what the system's answers mean. In this case, there are two possible interpretations:

1. Assuming complete information about the events that have occurred, and assuming complete knowledge about the initial conditions and the effects of events, it is the case that `F` does not hold in `T`.

2. We have been unable to ascertain whether `F` holds or not because of lack of information.

It is simple to imagine examples of the first case. The second possibility arises, for example, with case 3 in [25, p.90], where two events `e` and `e′` are known, the first one initiates a fluent `u` and the second one terminates a fluent `u′`, with `u` and `u′` mutually exclusive. It turns out that some intervening event (or events) must have occurred between `e` and `e′` which terminates `u` and initiates `u′`. The problem is that for any particular time point `T` after `e` and before `e′`, the query `holdsAt(u,T)` will fail. This is a problem inherent to the use of negation as failure, which is, in this case, too strong. Similar difficulties arise with concurrent events which are not precluded by the event calculus.

In the event calculus, problems also arise with negation as failure in the presence of incomplete information in settings having nothing to do with incompleteness about the temporal relations between events. The event calculus appeals to *case semantics*. The basic idea is that events are treated as first order objects which may be predicated by so-called *cases*. For example, events may have *agents*, *objects*, etc. An apparent advantage of *case semantics* is that event cases need not always be known in order to reason about events. Unfortunately, under negation as failure, this advantage is not always realized. For example, assume that John gave a book away, but it is not known to whom. Negation as failure will conclude that for each individual in the universe, it is not the case that that individual owns the book. Therefore, nobody owns the book after John gave it to somebody.

It is well known that logic programming is inappropriate for reasoning with non-categorical theories. In fact, the extension of the logic programming paradigm to work with such theories is a very active area of research [15]. Therefore, it should not be surprising that the examples of overcommitment mentioned above arise. Our concern with the event calculus, formulated as it is as a logic program with negation as failure, is that it provides no clear guidelines to the programmer about when overcommitments can arise, which is to say, it does not characterize a class of *sound* programs. Of course, soundness is possible only relative to some prior, universally accepted specification of the task. In this chapter, we provide such a specification for the kinds of temporal reasoning tasks for which the event calculus was designed. In doing so, we shall appeal to the extension of the situation calculus, enriched with time and event occurrences, as described in chapter 4. With this specification in hand, we shall be in a position to derive a logic program which is sound with respect to it.

Murray Shanahan [55, 56] has extended the calculus of events in several interesting ways. However, he bases his research on an event calculus which loosely corresponds to the original calculus of events. In fact, Shanahan's calculus of events avoids the problems we describe in this section by eliminating the notion of temporal interval from the logical language altogether. His departure from the original calculus is more extreme in his later work [56] in which he does not rely on negation-as-failure as the *non-monotonic* approach to deal with the frame problem. It would be interesting to study the logic programming implementation of Shanahan's event calculus and how it relates to our proposal.

7.3.2 Defining an Interval Based Ontology.

The version of the situation calculus we have introduced has an ontology based on time points. On the other hand, the calculus of events and the interval temporal logic have an ontology in which the primitive temporal objects are intervals of time. In this section we show how the expressiveness of the interval based language can be realized within the situation calculus. To extend the language, we borrow the terminology used in the calculus of events (using predicate names for terminations or initiations of intervals, etc.). Therefore, when axiomatizing a given domain, one has the freedom to choose whatever view of time seems most appropriate for the application. However, as mentioned earlier, the notion of an undivisible interval (i.e., a *moment*) cannot be modeled with our temporal structure.

Before introducing predicates on intervals, we extend the language of the situation calculus with two functions *pred* and *act* which identify the unique predecessor of a situation and the action that leads to that situation. From (2.1)-(2.1) it follows that:

$$s \neq S_0 \supset (\exists\, !a, s')\, s = do(a, s'). \tag{7.3}$$

That is, every situation, except for S_0, has a unique predecessor. Also, there is a unique action connecting a situation different from S_0 to its predecessor. Thus, we use the function $pred : \mathcal{S} \to \mathcal{S}$ to name the predecessor of a situation (we leave it undefined for S_0), and the function $act : \mathcal{S} \to \mathcal{A}$ to name the action that leads to the situation (also left undefined for S_0). Therefore:

$$s \neq S_0 \supset do(act(s), pred(s)) = s. \tag{7.4}$$

Now, we introduce a set of predicates aimed at capturing the essential aspects of the language of the calculus of events. These predicates are:

- $term \subseteq \mathcal{F} \times \mathcal{S}$, which is true of a situation s and fluent f if s terminates f. Thus, f is true[1] from some situation prior to s up to the situation s:

$$term(f, s) \equiv \neg holds(f, s) \wedge [s \neq S_0 \supset holds(f, pred(s))]. \tag{7.5}$$

[1]We say that f is true (or false) in a situation when f *holds* (or does not) in that situation.

- *init* $\subseteq \mathcal{F} \times \mathcal{S}$, which is true of a situation s and fluent f if f was false before s and true at s:

$$init(f, s) \equiv holds(f, s) \wedge [s \neq S_0 \supset \neg holds(f, pred(s))]. \tag{7.6}$$

- *broken* $\subseteq \mathcal{F} \times \mathcal{S} \times \mathcal{S}$, which is true of a pair of situations s_1 and s_2, and fluent f if $s_1 < s_2$ and the fluent f does not hold in some situation between s_1 and s_2.

$$broken(f, s_1, s_2) \equiv (\exists\, s)(s_1 \leq s < s_2) \wedge \neg holds(f, s). \tag{7.7}$$

- *maximal* $\subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{F}$, which is true of a pair of situations s_1 and s_2, and fluent f if $s_1 < s_2$, f holds in all situations between s_1 and s_2, f also holds in s_1 and f does not hold in s_2. The predicate maximal is used to define the notion of a maximal interval in which some fluent holds.

$$\begin{aligned} &maximal(s_1, s_2, f) \equiv \\ &\quad (s_1 < s_2) \wedge \neg broken(f, s_1, s_2) \wedge \\ &\quad init(f, s_1) \wedge term(f, s_2). \end{aligned} \tag{7.8}$$

- Finally, *incompatible* $\subseteq \mathcal{F} \times \mathcal{F}$, which is true of two fluents if they can not hold at the same time.

$$incompatible(f, f') \equiv [holds(f, s) \equiv \neg holds(f', s)]. \tag{7.9}$$

These have the following consequences:

$$term(f, s) \wedge s \neq S_0 \supset (\exists\, s')maximal(s', s, f), \tag{7.10}$$

$$\begin{aligned} &[term(f, s) \wedge incompatible(f, f') \wedge term(f', s') \wedge s < s'] \supset \\ &\qquad (\exists\, s_1)maximal(s_1, s', f') \wedge s < s_1, \end{aligned} \tag{7.11}$$

$$\begin{aligned} &[init(f, s) \wedge incompatible(f, f') \wedge init(f', s') \wedge s < s'] \supset \\ &\qquad (\exists\, s_2)maximal(s, s_2, f') \wedge s_2 < s', \end{aligned} \tag{7.12}$$

$$\begin{aligned} &[init(f, s) \wedge term(f', s') \wedge incompatible(f, f')] \supset \\ &\qquad (\exists\, s_1, s_2)(s < s_1 \leq s_2 < s') \wedge \\ &\qquad maximal(s, s_1, f) \wedge maximal(s_2, s', f'). \end{aligned} \tag{7.13}$$

The definition of *maximal* (axiom (7.8)) and theorems (7.10)-(7.10) capture the intuitions behind the cases of start and end points of intervals as discussed in [25, pp. 90-93].

It is important to note that the predicates introduced in this subsection do not depend on the definition of *actual*. However, by integrating the notion of an *interval* with the notion of an *actual* time line we gain the same representational features claimed for the event calculus.

Without a precise axiomatization for the calculus of events a formal argument that our language subsumes that of the calculus of events cannot be made. The alternative we followed in the definitions above, is to show that, within the extended situation calculus, an interval-based ontology can be defined analogous to that of the event calculus. In other words, we argue that the extended situation calculus is suitable for those applications where something like the event calculus provides the right ontology.

It is also worth noting that with the introduction of terminations and initiations (*term*, *init* and *maximal*) we can use an efficient algorithm for dealing with the frame problem in the cases in which complete information is at hand. Following Kowalski's proposal [24, pp.138-142], we can define an algorithm to answer queries of the form $holds(F, S)$, in which S is a completely determined situation. Such an algorithm would deal with a unique sequence of actions, which is required to be totally ordered, and the effects of each action have to be completely known. The algorithm uses a simple table of terminations and initiations. For every action that is given in the sequence, the algorithm keeps track of all the fluents that are affected by the action[2]. If a query is given with regards to a fluent F, the algorithm looks in its data structure for the last action that affected that fluent and answers whether the fluent holds based on the information encoded.

7.3.3 The Extended Situation Calculus and Logic Programming.

Here, we utilize the notion of *actual* line of situations axiomatized in chapter 4, along with the notion of occurrences and argue that the extended situation calculus subsumes the expressivity that is wanted for the calculus of events. We use our extended version of the situation calculus [41] to derive a logic program for temporal reasoning. The program is based on the language we presented in chapters 2 and 4 along with the addition of some defined predicates. These defined predicates are added to facilitate the derivation of the logic program.

7.3.4 Basis for the Logic Program.

In this subsection we present the theorems of the situation calculus (as defined in chapters 2 and 4) that we will use to develop the logic program presented later. First, our goal is to derive a program to reason about which fluents hold or do not hold as a result of performing a completely specified set of actions. Furthermore, as detailed later, we assume that this set of actions corresponds to the set of all actions that *occur*. Hence, reasoning is constrained to the *actual* line of situations. Before we do so, however, it is necessary to extend the language in two simple ways. First, we introduce the definition:

$$holds(not(f), s) \equiv \neg holds(f, s). \tag{7.14}$$

Thus, for the derivation for the program, we do not use the notation $holds(\neg f, s)$ as an abbreviation. Rather, *not* is a new fluent function symbol. Also, we define the predicate $start \subseteq \mathcal{S} \times \mathcal{T}$ which is introduced to replace the function symbol *start* for the purposes of the logic programming implementation[3]. Since we are only concerned with the actual line of situations, $start(s, t)$ is defined for pairs $\langle s, t\rangle$ if s is an actual situation whose starting time is t. The predicate is defined as follows:

$$start(s, t) \equiv t = start(s) \wedge actual(s). \tag{7.15}$$

Proposition 7.1 *From the basic situation calculus axioms Σ_b[4], the axioms about the function start (4.4)-(4.4) and the definition (7.15) it follows that:*

$$\begin{aligned}
&start(s, t) \wedge start(s, t') \supset t = t',\\
&start(s, t) \wedge start(do(a, s), t') \supset t < t',\\
&start(S_0, 0),\\
&start(s, t) \wedge start(s', t') \supset [s < s' \equiv t < t'].
\end{aligned}$$

[2]The primitive fluents or their negations.

[3]In the axiomatization we distinguish between the predicate symbol *start* and the function symbol *start* from the context.

[4]Recall that Σ_b denotes the axioms (2.1)-(2.1) and (2.7)-(2.7).

As discussed later, the information that the logic program will take as input corresponds to a set of actions that are known to have occurred. For this reason, we need to make use of the following propositions:

Proposition 7.2 *The following can be easily established from (2.9) and (4.7):*

$$actual(s) \equiv s = S_0 \vee (\exists a, s').s = do(a, s') \wedge occurs(a, s').$$

Furthermore, the definitions for $holds_{\mathcal{T}}$ and *during* ((4.9) and (4.9)) can be rewritten in the following equivalent manner:

$$holds_{\mathcal{T}}(f, t) \equiv (\exists\, s).during(t, s) \wedge holds(f, s), \tag{7.16}$$

$$\begin{aligned} during(t, s) \equiv\ & start(s, t_1) \wedge t_1 < t \wedge \\ & [((\exists\, a, t_2).start(do(a, s), t_2) \wedge t \leq t_2) \vee \\ & \quad (\neg(\exists\, a)actual(do(a, s)))]. \end{aligned} \tag{7.17}$$

We also make use of the following:

Proposition 7.3 *From the definition of* $occurs_{\mathcal{T}}$ *in (4.8) and (7.15), it follows that:*

$$occurs(a, s) \equiv (\exists\, t).occurs_{\mathcal{T}}(a, t) \wedge start(do(a, s), t).$$

In the same manner as we introduced the defined predicate *occursBet* in 4.11, we introduce the predicate $occursBet_{\mathcal{T}}(a, t_1, t_2)$

$$occursBet_{\mathcal{T}}(a, t_1, t_2) \equiv (\exists\, t).t_1 < t < t_2 \wedge occurs_{\mathcal{T}}(a, t). \tag{7.18}$$

Proposition 7.4 *From* Σ_b *(4.8), proposition 7.1 and (7.18), it follows that:*

$$start(s, t) \wedge start(do(a, s), t') \supset \neg(\exists\, a').occursBet_{\mathcal{T}}(a', t, t').$$

Also,

Proposition 7.5 *From* Σ_b*, propositions 7.1-7.4 and (7.15) it follows that:*

$$\begin{aligned} & start(s, t) \equiv \\ & \qquad [s = S_0 \wedge t = 0] \vee \\ & \qquad [(\exists\, a, s_p, t_p).s = do(a, s_p) \wedge occurs_{\mathcal{T}}(a, t) \wedge \\ & \qquad\quad start(s_p, t_p) \wedge \neg(\exists\, a').occursBet_{\mathcal{T}}(a', t_p, t)]. \end{aligned}$$

To facilitate the implementation described later, specifically, to eliminate a potential source of non-terminating loops in the logic program corresponding to these axioms, we need:

Proposition 7.6 *From* Σ_b*, (4.1)-(4.1) and proposition 7.5, it follows that:*

$$\begin{aligned} & start(s, t) \equiv \\ & \qquad [s = S_0 \wedge t = 0] \vee \\ & \qquad [(\exists\, a, s_p, t_p).s = do(a, s_p) \wedge occurs_{\mathcal{T}}(a, t) \wedge \\ & \qquad\quad start(s_p, t_p) \wedge \neg(\exists\, a').occursBet_{\mathcal{T}}(a', t_p, t) \ \wedge \\ & \qquad\quad t_p < t \wedge (t_p = 0 \vee (\exists\, a_p)occurs_{\mathcal{T}}(a_p, t_p))]. \end{aligned}$$

7.3.5 A Logic Programming Implementation and a Soundness Argument.

In this section we describe a logic programming implementation of a fragment of our extended situation calculus axiomatization, and show its soundness with respect to Clark's completion semantics [11], under suitable circumstances. The program exhibits much of the functionality of the event calculus, but in view of its soundness, is on firmer ground. Many of the assumptions we make in what follows are clearly too strong, and can be relaxed; exactly how to do this will be the subject of future research.

Problem Independent Clauses

```
holds(n(F),S):- not holds(F,S).
                      /* If half of axiom (7.14). */
holdsT(F,T):- during(T,S), holds(F,S).
                      /* If half of (7.16). */
actual(s_0).          /* Part of if half of Proposition 7.2. */
actual(do(A,S)):- occurs(A,S).
                      /* Rest of if half of Proposition 7.2. */
occurs(A,S):- occursT(A,T), start(do(A,S),T).
                      /* If half of Proposition 7.3. */
start(s_0,0).         /* Part of if half of Proposition 7.6. */
start(do(A,S),T):- occursT(A,T),          /* Rest of if half */
                   (occursT(Ap,Ts);Ts=0), /* of Prop.  7.6   */
                   Ts<T,
                   not occursBetTp(Ts,T),
                   start(S,Ts).
occursBetTp(Tp,T):- occursBetT(E,Tp,T).
occursBetT(E,Tp,T):- occursT(E,Tpp), Tp<Tpp, Tpp<T.
                      /* If half of Axiom 7.18. */
during(T,S):- start(S,T1),         /* Part of if half */
              start(do(A,S),T2), /* of Axiom 7.16.    */
              T1<T, T=<T2.
during(T,S):- start(S,T1), T1<T,  /* Rest of if half */
              not actualAft(S)./* of 7.16.      */
actualAft(S):- actual(do(A,S)).
```

Notice that, in the above, we use the if half of Proposition 7.6 instead of Proposition 7.5. This is to eliminate a source of non-termination in the program, which the use of Proposition 7.5 would cause. Furthermore, when translating the definition of *during* (7.16) and proposition 7.6 we introduce two auxiliary predicates `occursBetTp` and `actualAft`. These predicates are used to properly translate a subformula of the type $\neg(\exists a)p(a)$ into a literal.

Problem Specific Clauses

Initial State: We assume that the specification of the initial state consists of a sentence of the form:

$$holds(f, S_0) \equiv f = F_1 \vee \ldots \vee f = F_k.$$

In other words, we assume complete initial information about the *holds* predicate. The if half of such a specification yields the corresponding logic programming clauses:

```
holds(f1,s_0).   holds(f2,s_0). ... holds(fk,s_0).
```

Event Occurrences: We assume that the specification of event occurrences consists of a sentence of the form:

$$occurs_{\mathcal{T}}(e,t) \equiv [e = E_1 \wedge t = T_1] \vee \ldots \vee [e = E_p \wedge t = T_p].$$

In other words, we assume complete information about the $occurs_{\mathcal{T}}$ predicate. The if half of such a specification yields the corresponding logic programming clauses:

```
occursT(e1,t1).  occursT(e2,t2). ..., occursT(ep,tp).
```

Notice that there are two possible ways to interpret a declaration that at time T an event E has occurred:

1. The assertion $occurs_{\mathcal{T}}(E,T)$ is an *implicit* assertion that, in addition, $Poss(E,S)$ is true, where S is the state which includes time T (see axiom (4.1)). This perspective is problematic in the logic programming setting (see Kowalski [25] for an example).
2. The preconditions of the event E are known to be true at time T. In this case, event preconditions are being treated as *integrity constraints*; before an assertion of the form $occurs_{\mathcal{T}}(E,T)$ can be accepted by the database, the precondition (integrity constraint) $Poss(E,S)$ must be proved true, where S is the state which includes time T. If the precondition is false (or unknown), the update $occurs_{\mathcal{T}}(E,T)$ is rejected.

In this paper, we adopt the latter interpretation. This allows us to assume that all event preconditions are identically true, i.e.

$$Poss(a,s) \equiv true. \tag{7.19}$$

Of course, this makes sense only when integrity constraint enforcement has been incorporated into the temporal database, and is invoked whenever event occurrences are declared to the database. We do not further discuss integrity maintenance in this paper.

Successor State Axioms: As discussed in section 2.2, these are sentences of the following form, one for each fluent F:

$$Poss(a,s) \supset holds(F, do(a,s)) \equiv \Phi_F(a,s).$$

In view of our assumption about integrity maintenance (axiom (7.19)), successor state axioms will have the particularly simple form:

$$holds(F, do(a,s)) \equiv \Phi_F(a,s).$$

For example, the following is a successor state axiom for the fluent $rank$ in an education database:

$$\begin{aligned} & holds(rank(x,y), do(e,s)) \equiv \\ & \quad [(\exists z)e = promote(x,z,y) \vee e = hire(x,y) \vee \\ & \quad\; \neg[(\exists z)e = promote(x,y,z) \vee e = leave(x,y)] \wedge holds(rank(x,y),s)]]. \end{aligned}$$

This has, as its if half, the logic programming clauses:

```
holds(rank(X,Y),do(E,S)):- E = hire(X,Y) ; E = promote(X,Yp,Y).
holds(rank(X,Y),do(E,S)):- not E=promote(X,Y,Z),
                           not E=leave(X,Y), holds(rank(X,Y),S).
```

We shall assume that all logic programming clauses corresponding to successor state axioms are if halves of such axioms.

An Example.

As an example, in figure 7.2 we show a set of problem specific clauses to implement Kowalski and Sergot's promotion example. Of course, these clauses are in addition to the problem independent clauses of Section 7.3.5 above.

```
holds(rank(X,Y),do(E,S)):- E = hire(X,Y) ; E = promote(X,Yp,Y).
holds(rank(X,Y),do(E,S)):- not E=promote(X,Y,Z), not E=leave(X,Y),
                           holds(rank(X,Y),S).
occursT(promote(mary,lecturer,assisProf),1).
occursT(promote(john,lecturer,assisProf),2).
occursT(leave(john,assisProf),3).
holds(rank(mary,lecturer),s_0).
holds(rank(john,lecturer),s_0).
```

Figure 7.2: The Promotion Example

7.3.6 A Soundness Argument.

We are now in a position to argue the soundness of the above description of a logic programming implementation for temporal reasoning. Soundness will be with respect to the Clark completion semantics of the program.

1. **iff:** All the program clauses are if halves of corresponding iff axioms in our extended situation calculus.

2. **Equality Theory:** Unique names for states holds (immediate consequence of proposition 2.1), as they do for events (section 2.2). We assume they hold for all other domain functions, for example, that $AssisProf \neq AssocProf$, etc. In other words, the axioms satisfy Clark's equality theory.

3. **The Frame Problem:** Our specification of a solution to the frame problem had two components (section 4.3): successor state persistence and the assumption of no intervening events. The former is handled by successor state axioms, which we have already incorporated into our axiomatization. The latter involves minimizing event occurrences. It is easy to see that under the assumption about event occurrences of Section 7.3.5, the models of the theory are all already minimal with respect to the action occurrences. So the axioms satisfy the assumption of no intervening events. It follows that the axioms satisfy our specification of a solution to the frame problem.

The axioms satisfy all the conditions of the Clark completion semantics of the program. We conclude that the program is sound with respect to Clark's semantics.

7.3.7 Completeness.

An important assumption that has to be made when working with logic programs is that we work with a Herbrand universe. Thus, we work with interpretations in which the objects that exist are all and only the ones that have names.

Proposition 7.7 *Given the axiomatization of the situation calculus given in chapters 2 and 4, along with the extensions introduced in this section, it follows that the above logic program is complete for ground queries of the form* `holdsT(f,t)`*, where* `f` *is some fluent constant and* `t` *is a numeric constant representing a time point.*

The proof of this proposition is in the appendix. This proof is inductive and shows the completeness of the program relative to the ground literals of the program. An alternative is to appeal to a more general result due to Apt and Bezem [7]. It can be proven that the logic program presented in this section falls in the category of *acyclic logic programs* defined by Apt and Bezem [7]. Acyclic logic programs are defined in the following manner: Let P be a program and M be a mapping from the elements of the Herbrand Base of the program to the natural numbers. A mapping M for program P is acyclic if for any ground instance of a clause in P of the form:

$$\mathtt{A{:}\text{-}L_1,\ldots,L_n.}$$

M is such that $M(\mathtt{A})) > M(\mathtt{L_i})$ $(1 \leq \mathtt{i} \leq n)$ and $M(\mathtt{A}) = M(\neg\mathtt{A})$. A logic program is acyclic if and only if it has an acycling mapping.

It is not difficult to find an acyclic mapping for our program. We find such a mapping in the following way: First, we map all ground unit clauses to 0 (this includes all `occursT` literals along with the literals that specify the initial situation). All `occursBetT` literals are mapped to 1, `occursBetTp` literals are mapped to 2. All other literals in the Herbrand Base name a situation term. We build a mapping for these literals that increases monotonically with the length of the situation term (measured by the nesting of the `do` term). Thus:

$$M(\mathtt{holds(f_1,do(a_1,do(a_2,s_0)))}) > M(\mathtt{holds(f_2,do(a_1,s_0))}).$$

Mappings for the other literals can be trivially found in a similar fashion.

As shown by Apt and Bezem, acyclic programs enjoy many interesting properties. Of particular relevance to us are: Firstly, acyclic programs are guaranteed to terminate for a wide class of goals. In particular, ground queries of the sort we mention in proposition 7.7 belong to this class. Secondly, if we add a domain closure axiom, the resulting theory is complete and decidable with respect to a class of formulas that includes the ground queries of proposition 7.7.

Thus, the fact that our program is acyclic provides a stronger foundation to our approach to temporal reasoning in logic programming.

7.4 The Situation Calculus and Modal Temporal Logics.

In this section, we show how the modal temporal logic of concurrency [19] can be embedded in the situation calculus. This embedding shows that this modal logic is strictly less expressive than the extended situation calculus we propose in chapter 4. Of particular interest is the fact that none of the modal logics of which we are aware give first class status to events and their occurrences, which is a major asset of the situation calculus.

In [38], McCarthy and Hayes argue, without proof, that the expressive power of modal logics can be gained without the use of modalities using an ordinary truth-functional logic. The proposal is to identify each "possible world" with a situation in a situation calculus language. In this section we lend support to this intuition by showing how one particular such logic – the Temporal Logic of Concurrency – can be embedded in the extended situation calculus.

7.4.1 The Temporal Logic.

The *Temporal Logic of Concurrency* (*TL* for short) [19] is a propositional modal logic, in which the "possible worlds" are temporally ordered *states*. *TL* formulas are defined in terms of atomic propositions by the following BNF expression:

$$A ::= f \mid \neg A \mid A \rightarrow A \mid \Box A \mid \bigcirc A \mid A\,\mathcal{U}\,A.$$

Informally, the meanings of the sub-expressions are as follows: f represents an elementary proposition that is true *now*; $\Box A$ means A is true from now on. $\bigcirc A$ means, true at the next state. $A\,\mathcal{U}\,B$ means A is true until B is true; from then on, A is unrestricted. The formula $\neg\Box\neg A$ is abbreviated as $\Diamond A$[5]. A model $\mathcal{M}$ of a *TL* theory is a structure $\langle S, \sigma, V\rangle$, in which S is an enumerable set of states, σ is a surjective function from the non-negative integers to S (σ_i denotes the state resulting from applying σ to the non-negative integer i). The sequence $\sigma_0 \ldots \sigma_i \ldots$ represents a temporally ordered sequence of states. V is a function from the atomic formulas or propositions to a subset of states in which the atomic formulas are *true*. $\mathcal{M} \models_i A$ means that A is true in the state σ_i; $\mathcal{M} \models A$ means that A is true in every state. For a complete definition of the $\models_i$ relation see [19, p.72].

Axiom Schemata of *TL*:

$$\begin{aligned}
&\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B),\\
&\bigcirc(A \rightarrow B) \rightarrow (\bigcirc A \rightarrow \bigcirc B),\\
&\bigcirc\neg A \leftrightarrow \neg \bigcirc A,\\
&\Box A \rightarrow A \wedge \bigcirc\Box A,\\
&\Box(A \rightarrow \bigcirc A) \rightarrow (A \rightarrow \Box A), \qquad (7.20)\\
&A\,\mathcal{U}\,B \rightarrow \Diamond B,\\
&A\,\mathcal{U}\,B \leftrightarrow B \vee (A \wedge \bigcirc(A\,\mathcal{U}\,B)).
\end{aligned}$$

TL also includes all tautologies.

Notice that (7.20) is an induction axiom.

Rules of Inference of TL:

- Detachment: from A and $A \rightarrow B$ infer B.
- Necessitation: from A infer $\Box A$ and $\bigcirc A$.

7.4.2 The Embedding.

It is important to observe that even though the logic just presented is called a *temporal logic of concurrency*, it does not provide a mechanism for dealing with true *concurrency*. In fact, in this setting, concurrency means that there might be multiple agents co-existing but it is not possible for them to execute actions in a concurrent manner (i.e., simultaneously). Therefore, we do not need true concurrent actions to embed *TL* in the situation calculus.

In order to perform the embedding, we give a translation of *TL* formulas to the situation calculus, and also provide a correspondence between the notions of $\models_i$ and $\models$ in *TL* and entailment in our

[5]The other usual logical connectives ($\vee$, $\wedge$, $\leftrightarrow$) are defined in terms of negation ($\neg$) and implication ($\rightarrow$) in the standard way.

version of the situation calculus that includes the notion of an actual line of situations (chapter 4). Since *TL* takes time to be linear, we draw a correspondence between the states σ_i and the states of the *actual* line. Furthermore, in what follows, all our axioms refer to *actual* states.

Syntactic Characterization. Each atomic formula in *TL* is characterized by a constant of sort fluent ($\mathcal{F}$). Let Λ be an arbitrary formula in *TL*. Its translation into the situation calculus is:

$$(\forall\, s).actual(s) \supset holds(\Lambda, s).$$

where $holds(\Lambda, s)$ abbreviates a situation calculus formula defined as follows[6]:

$$\begin{aligned}
&holds(\neg\Lambda, s) = \neg holds(\Lambda, s)\\
&holds(\Lambda_1 \rightarrow \Lambda_2, s) = holds(\Lambda_1, s) \supset holds(\Lambda_2, s)\\
&holds(\Box\Lambda, s) = (\forall\, s').s' \geq s \wedge actual(s') \supset holds(\Lambda, s')\\
&holds(\bigcirc\Lambda, s) = (\forall\, a).actual(do(a, s)) \supset holds(\Lambda, do(a, s))\\
&holds(\Lambda_1\, \mathcal{U}\, \Lambda_2, s) = (\exists\, s').s' \geq s \wedge actual(s') \wedge holds(\Lambda_2, s') \wedge\\
&\qquad\qquad (\forall\, s'')\ [s \leq s'' < s' \supset holds(\Lambda_1, s'')]
\end{aligned}$$

It is trivial to show that the abbreviation for $holds(\neg\Box\neg\Lambda, s)$ is equivalent to the intuitively correct expression:

$$(\exists\, s').s' \geq s \wedge actual(s') \wedge holds(\Lambda, s').$$

Semantic Concerns. Models in *TL* are structures in which the function σ defines an infinite sequence of states. The intuition behind this sequence is that the world passes through each state in this sequence in the order prescribed by it. In order to ensure that our models contain an infinite sequence of actual situations, we need to add the following axiom:

$$(\forall\, s).actual(s) \supset (\exists\, a).actual(do(a, s)). \tag{7.21}$$

This axiom says that every actual situation must have a *next* actual situation, which, given axiom (4.1), is unique.

We must show that the translation of each instance of an axiom schema of *TL* is entailed by our enriched situation calculus.

Proposition 7.8 *If* Λ *is an instance of an axiom schema of* TL*, then*

$$\Sigma \models (\forall\, s).actual(s) \supset holds(\Lambda, s),$$

where Σ *is the set of situation calculus axioms (2.1),(2.1),(2.1), (4.1), (4.1) and (7.21).*

For most *TL* schemata, the proof is straightforward, but somewhat tedious. Perhaps, the most interesting proof is that of the *TL* induction schema (7.20) which, not surprisingly, requires the use of our situation calculus induction axiom (2.1).

Also:

[6]The translation of formulas involving the defined logical connectives ($\vee$, $\wedge$, $\leftrightarrow$) is straightforward.

Proposition 7.9 *The necessitation rules for $\Box$ and $\bigcirc$ are sound, since it is trivial to show that if Σ includes the situation calculus axioms (2.1),(2.1),(2.1), (4.1), (4.1) and (7.21):*

$$\Sigma \models (\forall s).actual(s) \supset holds(\Lambda, s) \;\; \Rightarrow \;\; \Sigma \models (\forall s).actual(s) \supset holds(\Box\Lambda, s),$$

as well as:

$$\Sigma \models (\forall s).actual(s) \supset holds(\Lambda, s) \;\; \Rightarrow \;\; \Sigma \models (\forall s).actual(s) \supset holds(\bigcirc\Lambda, s).$$

Furthermore, let Φ_{TL} be an arbitrary set of *TL* sentences, and let ϕ be a sentence of *TL*. If Φ_{sc} corresponds to the translation of Φ_{TL} to the situation calculus, then it is not difficult to see that $\Phi_{TL} \models \phi$[7] iff $\Sigma \cup \Phi_{sc} \models (\forall s).actual(s) \supset holds(\phi, s)$.

Finally, it is also the case that $\Phi_{TL} \models_i \phi$ iff:

$$\Sigma \cup \Phi_{sc} \models (\forall a_1, \ldots, a_i).actual(do([a_1, \ldots, a_i], S_0)) \supset holds(\phi, do([a_1, \ldots, a_i], S_0)), \qquad (7.22)$$

where $do([a_1, \ldots, a_i], S_0)$, denotes the situation term:

$$do(a_i, do(a_{i-1}, \ldots do(a_1, S_0) \ldots)).$$

Since, by axiom (4.1), for any i there exists a unique sequence $[a_1, \ldots, a_i]$ of actions such that $do([a_1, \ldots, a_i], S_0)$ is actual, we can write (7.22) as:

$$\Sigma \cup \Phi_{sc} \models (\exists a_1, \ldots, a_i).actual(do([a_1, \ldots, a_i], S_0)) \wedge holds(\phi, do([a_1, \ldots, a_i], S_0)).$$

We conclude that every entailment of a *TL* theory is an entailment of its situation calculus translation.

In summary:

Theorem 7.4.1 *The extended situation calculus subsumes the temporal logic of concurrency.*

7.4.3 Further Comments.

We have embedded the modal logic *TL* in the situation calculus. The question arises: Is it possible to realize this embedding without appealing to the extended situation calculus? The answer is yes. However, the resulting language is greatly impoverished. To see why, consider the axiom schema:

$$\bigcirc\neg A \leftrightarrow \neg \bigcirc A,$$

which ensures that $\bigcirc$ acts as a functional operator, i.e., every state has exactly one successor. We achieved this by mapping *TL* states to the extended situation calculus' *actual situations*, which also have a unique successor situation in the actual line. This can also be realized in the standard situation calculus by allowing only one action. In this way, every situation would have exactly one successor. The problem with this solution is that it gives up the ability to talk about arbitrary actions. If only one action exists (whose only purpose is to produce a state change), then we lose the ability to represent different actions and their effects.

With our approach, on the other hand, not only can we gain the same expressiveness of the *TL* modal logic, but we retain the basic features of the original situation calculus, plus the ability to talk about event occurrences. The only limitation introduced is that axiom (7.21) prevents us from expressing *deadlock* situations, i.e., situations in which no actions are possible. This is not a serious limitation since we can achieve the effect of a deadlock by introducing a *Wait* action with no effects.

[7] $\Phi_{TL} \models \phi$ means that ϕ is true in all the states in all models of Φ_{TL}

Chapter 8

Summary and Future Work.

8.1 Summary.

The main objective of this thesis is to design a logical language to represent knowledge about dynamic domains. We chose to use the situation calculus as the basic logical language. This choice is partly motivated by the expressive power of the language. However, a stronger motivation is the availability of solutions to the frame problem within the framework. Also, we benefited from the work of Lin and Reiter to approach the ramification problem within the situation calculus. As emphasized in this thesis, an important contribution of our work has been to extend the situation calculus language to deal with a wide variety of representational aspects of dynamic worlds. The contribution of our work to the problem of representing knowledge about dynamic worlds can be summarized as:

- Extend Lin and Reiter's work on the ramification problem by providing mechanisms to deal with theories of action that include binary state constraints and/or stratified definitions.
- Provide an approach to represent determinate knowledge about the future within the situation calculus.
- Extend the situation calculus to deal with concurrent actions. This problem is addressed by separating the problem into a *precondition interaction* problem and an *effect interaction* problem.
- Present an enriched ontology of the situation calculus that allows the representation of knowledge about continuous properties of the world.
- Show that the situation calculus provides a better logical foundation for reasoning about actions and time than other popular temporal logics.

8.2 Future Work.

The work reported in this thesis can be extended in many ways. Extensions are of two kinds. First, we can enrich the language in order to deal with a wider variety of representational issues. In particular:

- Multiple Agents: In this thesis we have presented an approach that permits one to express occurrences of external events. Also, we extended the discrete situation calculus to allow for concurrency. With these new elements, it should not be difficult to explicitly introduce the notion of *agent*. Thus, we would probably introduce a new sort for agents, with actions indexed by the agent that performs the action.

- Complex Events and Concurrency: The work on concurrency that we presented in this thesis deals with concurrent execution of primitive actions. We have not addressed the problem of representing concurrency of complex events. Dealing with this problem is also essential for settings in which there are multiple agents, each of which may be performing complex actions.
- Complex Actions as First Class Citizens: In CR's approach to complex actions that we discussed in chapter 5, these are treated as abbreviations. Thus, complex actions are not objects in the language. This choice limits the representational capabilities of the language. Treating complex actions as objects in the language would allow to write general properties about them.
- Relationship to other approaches. As discussed in section 3.1, our approach to solve the frame problem in the presence of binary state constraints is related to Vladimir Lifschitz's approach [28]. We are interested in establishing a formal relationship between both approaches.

  Also, we are interested in establishing a relationship between Shanahan's work on the representation of continuous change in the *circumscriptive event calculus* and our approach presented in chapter 6.

The second kind of extensions of this research deals with possible applications. For example:

- Specification of Dynamic Processes. The ability to deal with continuous change allows us to represent physical processes that are commonly encountered in industry. Thus, with the extended situation calculus described in this thesis, we provide a tool to specify these processes utilizing what is commonly called a *hybrid* representational mechanism. Thus, all the processes that can be modeled using standard mathematical models can be integrated with the models of actions and effects of the discrete situation calculus. The extensions to the situation calculus presented in chapters 4, 5 and 6 provide the foundations for such applications.
- Planning with External Events. The planning approaches built around the situation calculus normally consider single agent worlds in which all actions are the responsibility of the single agent. Thus, we can investigate the extension of this planning research to problems in which there are external events which may be the result of other agents present in the world, the effects of nature, or simply delayed effects resulting from actions performed by the same agent.
- Logic Programming Implementation. We would like to extend the logic program presented in chapter 7 to deal with a wider variety of problems. For example, it should not be difficult to extend the program to handle concurrent events and continuous change.

Appendix A

Proofs.

Proposition 2.1.

- $$(\forall\, s).\, S_0 \leq s. \tag{A.1}$$

  Let $\varphi(s) = [S_0 \leq s]$. Clearly $\varphi(S_0)$ is identically true. Furthermore, from (2.1) it follows that $S_0 \leq s \supset S_0 \leq do(a, s)$. Therefore, the antecedent of (2.1) is true for our choice of φ. Hence, (A.1) follows.

- $$(\forall\, a, s). \neg\, s < s. \tag{A.2}$$

  By contradiction using (2.1) with $s_1 = s_2 = s$.

- $$(\forall\, s)\ \neg s < S_0. \tag{A.3}$$

  This is easily proven by using (2.9), (2.1) and (2.9).

- $$(\forall\, s).\, s \neq S_0 \supset (\exists\, a, s').\, s = do(a, s'). \tag{A.4}$$

  Let $\varphi(s) = [s \neq S_0 \supset (\exists\, a, s'). s = do(a, s')]$. $\varphi(S_0)$ holds trivially. Since $\varphi(do(a, s))$ is identically true, $(\forall\, a, s).\varphi(s) \supset \varphi(do(a, s))$ follows. Therefore $(\forall\, s).\varphi(s)$ holds, thus (2.9) holds as well.

- $$(\forall\, s_1, s_2, s_3).\, s_1 < s_2 \wedge s_2 < s_3 \supset s_1 < s_3. \tag{A.5}$$

  Again, we use our induction axiom for situations. Let

  $$\varphi(s) = (\forall\, s_1, s_2).\, s_1 < s_2 \wedge s_2 < s \supset s_1 < s.$$

  $\varphi(S_0)$ holds trivially ($s_2 < S_0$ is identically false). We have to show that $\varphi(s) \supset \varphi(do(a, s))$. In fact, From (2.1), it follows that $s_1 < s \supset s_1 < do(a, s)$. Therefore:

  $$\varphi(s) \supset (\forall\, s_1, s_2)\ [s_1 < s_2 \wedge s_2 < s \supset s_1 < do(a, s)]. \tag{A.6}$$

Now, from (2.1) and (A.2) it follows that:

$$s_2 < do(a,s) \wedge s_2 \neq s \equiv s_2 < s.$$

Which allows to rewrite (A.6) as:

$$\varphi(s) \supset (\forall s_1, s_2)\ [s_1 < s_2 \wedge s_2 < do(a,s) \wedge s_2 \neq s \supset s_1 < do(a,s)]. \tag{A.7}$$

From (2.1):

$$(\forall s_1, s_2).s_1 < s_2 \supset s_1 < do(a, s_2).$$

Thus:

$$(\forall s_1, s_2).s_1 < s_2 \wedge s_2 < do(a,s) \wedge s_2 = s \supset s_1 < do(a,s).$$

Therefore, we can conclude that:

$$\varphi(s) \supset (\forall s_1, s_2)\ [s_1 < s_2 \wedge s_2 < do(a,s) \supset s_1 < do(a,s)]. \tag{A.8}$$

In other words $\varphi(s) \supset \varphi(do(a,s))$, Finally, we conclude that (A.5) is a consequence of the axioms.

•
$$(\forall a, s).s < do(a,s). \tag{A.9}$$

Trivially follows from (2.1).

•
$$do(a_1, s_1) < do(a_2, s_2) \supset s_1 < s_2. \tag{A.10}$$

Assume:

$$do(a_1, s_1) < do(a_2, s_2).$$

From (2.1):

$$do(a_1, s_1) \leq s_2.$$

From (A.9) $s_1 < do(a, s_1)$, which along with (A.5) leads to:

$$s_1 < s_2.$$

So that (A.10) holds.

•
$$(\forall a, s, s').do(a,s) \leq s' \supset s < s'. \tag{A.11}$$

Follows trivially from (2.9) and (2.9).

•
$$(\forall a, s_1, s_2).do(a, s_1) = do(a, s_2) \supset s_1 = s_2. \tag{A.12}$$

This can be shown by contradiction. Let's extend the language with new constants A, S_1, and S_2 such that:

$$do(A, S_1) = do(A, S_2) \wedge S_1 \neq S_2.$$

Obviously, $S_1 < do(A, S_1)$, since $do(A, S_1) = do(A, S_2)$, we get $S_1 < do(A, S_2)$. From (2.1), we obtain $S_1 \leq S_2$; since $S_1 \neq S_2$, we get $S_1 < S_2$. By symmetry, we also obtain $S_2 < S_1$, which contradicts (2.1). Therefore, (A.12) follows.

- $$(\forall\, a, s, s')\neg(s < s' < do(a, s)),$$

  follows directly from (2.1), (2.1) and (2.9).

- $$(\forall\, s_1, s_2).s_1 \prec s_2 \supset s_1 < s_2. \tag{A.13}$$

  This is shown by induction, using:

  $$\varphi(s) = (\forall\, s_1)\; s_1 \prec s \supset s_1 < s.$$

Proposition 2.2. This proposition establishes that from axioms (2.1-2.1) and (2.20) it follows that:

$$\begin{aligned} s_1 < s_2 \wedge holds(f, s_1) \wedge \neg holds(f, s_2) \supset& \\ (\exists\, s, a)\, (s_1 < do(a, s) \leq s_2) \wedge ab(f, a, s).& \end{aligned} \tag{A.14}$$

The proof is by contradiction. We extend the language with new constants S_1, S_2, and F, such that:

$$S_1 < S_2 \wedge holds(F, S_1), \tag{A.15}$$

$$\neg holds(F, S_2), \tag{A.16}$$

and:

$$(\forall\, s, a)\, (S_1 < do(a, s) \leq S_2) \supset \neg ab(F, a, s). \tag{A.17}$$

We will show that (A.15) and (A.17) contradict (A.15). The contradiction arises by applying the law of inertia (2.20) and the induction axiom (2.1) with:

$$\varphi(s) = (S_1 \leq s \leq S_2) \supset holds(F, s).$$

$\varphi(S_0)$ is immediately true, since $(S_1 \leq S_0 \leq S_2) \supset S_0 = S_1$ and we assumed that $holds(F, S_1)$. We now prove that $\varphi(s) \supset \varphi(do(a, s))$. We do this by cases:

- Here we prove that:

  $$(\forall\, a, s)\, \neg(S_1 \leq s \leq S_2) \wedge \varphi(s) \supset \varphi(do(a, s)).$$

  Assume that S is an arbitrary situation such that $\neg(S_1 \leq S \leq S_2)$. If $S_1 \leq do(a, S) \leq S_2$ is false, then trivially $\varphi(do(a, S))$ is true. Now, if $S_1 \leq do(a, S) \leq S_2$ is true, then $do(a, S) = S_1$, otherwise $(S_1 \leq S \leq S_2)$ would hold. Since $\neg ab(F, a, S_1)$ follows from (A.17), we infer, using the law of inertia, that $holds(F, do(a, S))$. Hence, $\varphi(do(a, s))$ also holds in this case.

- Now we prove that:

  $$(\forall\, a, s)\, (S_1 \leq s \leq S_2) \wedge \varphi(s) \supset \varphi(do(a, s)).$$

  Assume that S is an arbitrary situation such that $(S_1 \leq S \leq S_2) \wedge \varphi(S)$ is true. Obviously $holds(F, S)$ follows; this along with (A.17) ensures that $\varphi(do(a, S))$ must also be true. (Notice that if $(S_1 \leq S \leq S_2) \supset do(a, S) \neq S_1$).

Therefore, it follows that $(\forall\, a, s)\varphi(s) \supset \varphi(do(a, s))$. So, applying (2.1) we infer that (A.15) is contradicted. Hence, the proposition must be true.

Proposition 3.1. As mentioned in the text of the proposition, we show this by utilizing induction. Thus, we assume that $T_{sc_2}(S_0)$ is true and use the induction axiom (2.1) with:

$$\varphi(s) = S_0 \preceq s \supset T_{sc_2}(s).$$

Obviously, $\varphi(S_0)$ is true, given that we are assuming that $T_{sc_2}(S_0)$ is true. Hence, we need to show that:

$$\varphi(s) \supset \varphi(do(a, s)).$$

Which can be rewritten as:

$$(S_0 \preceq s \supset T_{sc_2}(s)) \supset (S_0 \preceq do(a, s) \supset T_{sc_2}(do(a, s))). \tag{A.18}$$

We know that $\neg S_0 \preceq s \supset \neg S_0 \preceq do(a, s)$. If $S_0 \preceq s$ is false, then (A.18) is trivially true. If $S_0 \preceq s$ is true, then (A.18) translates to:

$$S_0 \preceq s \wedge T_{sc_2}(s) \wedge S_0 \preceq do(a, s) \supset T_{sc_2}(do(a, s)).$$

This is proven by contradiction. Thus, assume that:

$$S_0 \preceq s \wedge T_{sc_2}(s) \wedge S_0 \preceq do(a, s) \wedge \neg T_{sc_2}(do(a, s)).$$

If $\neg T_{sc_2}(do(a, s))$ holds, then there must be a state constraint in T_{sc_2} that is not satisfied. Thus, let

$$\alpha(s) = \neg\rho(\mathbf{x}_r) \supset holds(f_1(\mathbf{x}_1), s) \vee holds(f_2(\mathbf{x}_2), s),$$

be an arbitrary such a constraint[1]. Since $\rho(\mathbf{x}_r)$ is state independent, it follows that:

$$\alpha(s) \wedge \neg\alpha(do(a, s)) \supset \neg\rho(\mathbf{x}_r).$$

Thus, it also follows that:

$$holds(f_1(\mathbf{x}_1), s) \vee holds(f_2(\mathbf{x}_2), s), \tag{A.19}$$

$$\neg(holds(f_1(\mathbf{x}_1), do(a, s)) \vee holds(f_2(\mathbf{x}_2), do(a, s))). \tag{A.20}$$

Notice that the only way for this situation to arise is that the set T'_{ef} contain effect axioms like:

$$Poss(a, s) \wedge \gamma^-_{f_1}(a, s) \supset \neg holds(f_1(\mathbf{x}_1), do(a, s)), \tag{A.21}$$

$$Poss(a, s) \wedge \gamma^-_{f_2}(a, s) \supset \neg holds(f_2(\mathbf{x}_2), do(a, s)). \tag{A.22}$$

Each of these effect axioms either belong to the original set of effect axioms T_{ef} or are a consequence of $T_{sc_2} \cup T_{ef}$. In either case it follows that the original theory Σ_{sc_2} is inconsistent. Thus, the result follows.

Theorem 3.1.1. This theorem is an immediate consequence of theorem 2.3.1 and proposition 3.1.

[1]Notice that the argument that follows works whether or not the *holds* literals are positive (as in this case) or negative.

Proposition 3.3.

1. To prove this result we show that there is a polynomial upper bound to the time it takes to generate and evaluate the successor state axioms for each fluent constant in the language. Let n_{ef} be the number of successor state axioms in Σ_{sc}. Let n_f be the number of fluents in the domain. To obtain the successor state axioms we do the following:

   - Given T_{sc} and T_{ef} we generate a set T'_{ef} of new effect axioms. Each element in T'_{ef} is obtained from an axiom in T_{ef} and one of the state constraints. Independently of the structure of the state constraints. At most $n_f - 1$ new effect axioms will be generated for each of the n_{ef} effect axioms in T_{ef}. Thus the size of $T_{ef} \cup T'_{ef}$ has an upper bound of $n_{ef} \times n_f$. Obviously, the set T'_{ef} can be generated in a time polynomial in n_{ef}, n_f and n_{sc}.
   - The derivation of the successor state axioms is done in time proportional to $n_f \times (n_{ef} + n'_{ef})$, where n'_{ef} corresponds to the size of T'_{ef}.
   - The length of any successor state axiom is bounded by $n_{ef} + n'_{ef}$ times the size of the longest effect axiom (which is constant).
   - Finally, determining the state of $do^{\mathcal{M}}(a, s)$ can be done by simple evaluation of each successor state axiom in time proportional to the length of the successor state axioms.

2. This is proven by reducing the 3-colouring of a graph problem to a problem of reasoning about actions in the presence of ternary constraints[2]. In fact, for each node i in the graph we introduce three fluents Red_i, $Green_i$ and $Blue_i$, along with the ternary constraint:

   $$holds(Red_i, s) \lor holds(Green_i, s) \lor holds(Blue_i, s).$$

   If j and k are neighbouring nodes in the graph we also add the three constraints:

   $$\begin{gathered} holds(Dummy, s) \lor \neg holds(Red_j, s) \lor \neg holds(Red_k, s), \\ holds(Dummy, s) \lor \neg holds(Green_j, s) \lor \neg holds(Green_k, s), \\ holds(Dummy, s) \lor \neg holds(Blue_j, s) \lor \neg holds(Blue_k, s). \end{gathered}$$

   T_{S_0} contains $holds(Dummy, S_0)$ along with:

   $$holds(Red_i, S_0) \land holds(Green_i, S_0) \land holds(Blue_i, S_0)$$

   for every node i. A single action $Colour$ is used with the effect axiom:

   $$Poss(Colour, s) \supset \neg holds(Dummy, do(Colour, s)).$$

   Obviously, the initial situation is such that the fluent associated to the colour of every node is determined (every node has all colours). Now, after action $Colour$ is performed, we need to find a colouring scheme that satisfies the constraint that neighbours cannot share a colour. In this colouring scheme a node may end up with more than one colour, but we could easily add binary constraints to eliminate this possibility. In any event, determining the colour of all nodes after $Colour$ is performed amounts to solving the 3-colouring of a graph problem. Hence, the result follows.

□

[2]We are grateful to Bart Selman for suggesting this reduction (personal communication).

Proposition 4.1. This proposition is proved by appealing to the induction axiom (2.1). The first part of the proposition follows by taking:

$$\varphi(s) = (\forall s').s' \leq s \supset [actual(s) \supset actual(s')],$$

and the second one, by taking

$$\varphi(s) = (\forall s').actual(s) \wedge actual(s') \supset s < s' \vee s' < s \vee s = s'.$$

Straightforward manipulation leads to:

$$(\forall a, s).\varphi(s) \supset \varphi(do(a, s)).$$

Which directly implies the results.

Proposition 4.2. It follows by using the induction axiom (2.1) with:

$$\varphi(s) = (\forall s').s' < s \supset start(s') < start(s).$$

Propositions 4.3 and 4.4. Both follow directly from the definitions of $holds_{\mathcal{T}}$, *during* and *occursBet*.

Proposition 4.5. Let O_s and O_t denote the left and right side of the equivalence. First we show that $O_s \supset O_t$:

From proposition 4.2 and using a simple induction on the length of $\mathcal{O}_<$ it follows that:

$$\mathcal{O}_<(s_1, \ldots, s_n) \supset \mathcal{O}_<(start(s_1), \ldots, start(s_n)). \tag{A.23}$$

In which the ordering $<$ in the left side refers to the order relation between situations, and the $<$ on the right side refers to the order relation between the reals.

From (2.9), (4.7) and propositions 4.1 and 4.2 it follows that:

$$s_1 < s_2 \wedge occurs(a_1, s_1) \wedge occurs(a_2, s_2) \supset do(a_1, s_1) < do(a_2, s_2). \tag{A.24}$$

Thus, from (A.24) and a simple induction on the length of $\mathcal{O}_<$ we obtain:

$$O_s \supset \mathcal{O}_<(do(A_1, S_1), \ldots, do(A_n, S_n)). \tag{A.25}$$

From (A.23), (A.25) and (4.8) we obtain:

$$\begin{aligned}
&(\exists s_1, \ldots, s_n).[occurs(A_1, s_1) \wedge \ldots \wedge occurs(A_n, s_n) \wedge \mathcal{O}_<(s_1, \ldots, s_n) \\
&\qquad \supset \\
&\qquad occurs_{\mathcal{T}}(A_1, start(do(A_1, s_1))) \wedge \ldots \wedge occurs_{\mathcal{T}}(A_n, start(do(A_1, s_n))) \wedge \\
&\qquad \mathcal{O}_<(start(do(A_1, s_1)), \ldots, start(do(A_1, s_n)))].
\end{aligned}$$

Thus, the we have proven that $O_s \supset O_t$.

Now we show that $O_t \supset O_s$:

From (4.8) we obtain:

$$\begin{aligned}(\exists\, t_1,\ldots,t_n).&\mathit{occurs}_{\mathcal{T}}(A_1,t_1)\wedge\ldots\wedge \mathit{occurs}_{\mathcal{T}}(A_n,t_n)\wedge \mathcal{O}_{<}(t_1,\ldots,t_n)\\ &\supset\\ &(\exists\, s_1,\ldots,s_n).\mathit{occurs}(A_1,s_1)\wedge\ldots\wedge \mathit{occurs}(A_n,s_n)\wedge\\ &\qquad \mathcal{O}_{<}(\mathit{start}(\mathit{do}(A_1,s_1)),\ldots,\mathit{start}(\mathit{do}(A_1,s_n))).\end{aligned} \tag{A.26}$$

From (2.9) and propositions 4.1 and 4.2 it follows that:

$$\mathit{start}(\mathit{do}(a_1,s_1))<\mathit{start}(\mathit{do}(a_2,s_2))\wedge \mathit{actual}(\mathit{do}(a_1,s_1))\wedge \mathit{actual}(\mathit{do}(a_2,s_2))\supset s_1<s_2. \tag{A.27}$$

From (A.26) and (A.27) it immediately follows that $O_t \supset O_s$.

□

Proposition 7.1. The proof is direct from the axioms mentioned in the text of the proposition.

Proposition 7.2. From (2.9) it follows that:

$$\mathit{actual}(s)\equiv s=S_0\wedge \mathit{actual}(S_0)\vee(\exists a,s').s=\mathit{do}(a,s')\wedge \mathit{actual}(\mathit{do}(a,s')).$$

From (4.1) and (4.7) it follows that:

$$\mathit{actual}(s)\equiv s=S_0\wedge \mathit{actual}(S_0)\vee(\exists a,s').s=\mathit{do}(a,s')\wedge \mathit{actual}(\mathit{do}(a,s')).$$

□

Proposition 7.3. Simply replace $\mathit{occurs}_{\mathcal{T}}$ by its definition.

Proposition 7.4. From (2.9) and (4.11) it follows that:

$$\neg(\exists\, a')\mathit{occursBet}(a',s,\mathit{do}(a,s)).$$

From here it is trivial to show that:

$$\neg(\exists\, a')\mathit{occursBet}_{\mathcal{T}}(a',\mathit{start}(s),\mathit{start}(\mathit{do}(a,s))).$$

which leads us to the result.

Proposition 7.5. By equivalence preserving transformations using (2.9), (4.11), propositions 7.1 and 7.3 we obtain:

$$\begin{aligned}\mathit{start}(s,t)\equiv s=S_0\wedge t=0\vee&\\ &(\exists\, a,s')s=\mathit{do}(a,s')\wedge \mathit{occurs}_{\mathcal{T}}(a,t)\wedge \mathit{start}(\mathit{do}(a,s'),t).\end{aligned}$$

using the fact that:

$$\neg(\exists\, a')\mathit{occursBet}_{\mathcal{T}}(a',\mathit{start}(s),\mathit{start}(\mathit{do}(a,s))).$$

the result follows.

Proposition 7.7. First, observe that we require that `occursT(A,T)` be complete. Thus, we know all the actions that occur along with their times of occurrence. Therefore, it follows that the predicate `occursBetT(E,Tp,T)` is also completely determined as long as `Tp` and `T` are ground terms. Therefore, `occursBetTp` is also complete for ground `Tp` and `T`.

Now, we show that `start(S,T)` is complete. Thus, we show that if the query `start(S,T)` is given to the program, then every pair of ground literals ⟨`S,T`⟩ that satisfies the query is returned by the program as an answer. We do so by induction:

- The initial step is trivial given that the pair `s_0,0` is returned by the first `start` clause.
- Induction hypothesis. Assume that if a situation `S` starts at time `T` then the program returns the ground literals `S,T`, as answer to the original query.
- Induction step. We need to show that the program will also return pair `do(A,S),T` when `start(do(A,S),T)` is true. Notice that the first three literals on the right side of the second `start` clause completely enumerate all the pairs `A,T` such that action `A` occurs at time `T`. After these three literals have been proven true, the terms `A`, `T`, and `Ts` are *instantiated* and that `Ts<T` holds. The next literal `not occursBetTp(Ts,T)`, succeeds if `Ts` corresponds to the time in which the situation `S` started. Thus, `start(S,Ts)` must succeed according to the induction hypothesis.

  Finally, the completeness of `occurs` follows directly from the completeness of `start` and `occursT`. Thus, it follows that `actual` and `during` are also complete. Hence, it remains to be shown that `holdsT` is indeed complete. This follows if `holds` is shown to be complete. Now, `holds` is a domain specific predicate which is specified by a set of effect axioms as specified earlier. It is simple to see that `holds` is complete from the assumptions made about the program. In fact, we assume that `holds(F,S)` is complete determined when `S` is instantiated with `s_0`, and that `holds(F,do(A,S))` is completely given from the state of `S` and the effect axioms. A simple induction argument yields the completeness of `holds`.

Proposition 7.8. We have to prove that:

$$\Sigma \models (\forall\, s).actual(s) \supset holds(\Lambda, s), \tag{A.28}$$

where Λ is replaced by each instance of an axiom schema of TL. For this proof, we will prove the right hand side of (A.28) making this assumption.

- $\Lambda = \Box(A \to B) \to (\Box A \to \Box B)$. We have to show that:

  $$(\forall\, s).holds(\Box(A \to B), s) \supset (holds(\Box A, s) \supset holds(\Box B, s)).$$

  This is done by manipulating $holds(\Box(A \to B), s)$ and using the fact that:

  $$[(\forall\, x)\ (p(x) \supset q(x))] \supset [(\forall\, x)\ p(x) \supset (\forall\, x)\ q(x)].$$

- $\Lambda = \bigcirc(A \to B) \to (\bigcirc A \to \bigcirc B)$. Showing that this is true involves exactly the same steps as the proof above, except that instead of quantifying over situations we quantify over actions.
- $\Lambda = \bigcirc\neg A \leftrightarrow \neg \bigcirc A$. So, we need to show that:

  $$\begin{aligned} actual(s) \supset [[(\forall\, a).actual(do(a,s)) \supset \neg holds(A, do(a,s))] \equiv \\ [\neg(\forall\, a).actual(do(a,s)) \supset holds(A, do(a,s))]]. \end{aligned} \tag{A.29}$$

For this proof we use the fact that every actual situation has exactly one successor actual situation. This is so, since axiom (7.21) requires that each actual situation have at least one successor actual situation; and axiom (4.1), says that every actual situation must have at most one successor actual situation. With this observation, the proof from left to right is immediate, since the only possibility for this not to hold is to have no action a with $actual(do(a,s))$ true. From right to left is also very simple. In fact, the right hand side is equivalent to:

$$(\exists a).actual(do(a,s)) \wedge \neg holds(A, do(a,s)).$$

Since the a that exists is unique, then it is obvious that this implies:

$$(\forall a).actual(do(a,s)) \supset \neg holds(A, do(a,s)).$$

- $\Lambda = \Box A \to A \wedge \bigcirc\Box A$. Given that s is actual, we have to prove that:

$$holds(\Box A, s) \supset holds(A, s) \wedge holds(\bigcirc\Box A, s).$$

Now, $holds(\Box A, s)$ abbreviates:

$$(\forall s').(s' \geq s) \wedge actual(s') \supset holds(A, s'),$$

which is equivalent to:

$$holds(A, s) \wedge (\forall s').(s' > s) \wedge actual(s') \supset holds(A, s').$$

Now, if s is actual, we know that there is a unique action α_s such that $do(\alpha_s, s)$ is actual. It is easy to show, using trichotomy for actual situations and axiom (2.1), that $do(\alpha_s, s) \leq s'$ follows. Therefore, the previous expression implies that:

$$holds(A, s) \wedge [actual(do(\alpha_s, s)) \supset (\forall s').(s' \geq do(\alpha_s, s)) \wedge actual(s') \supset holds(A, s')],$$

which is equivalent to:

$$holds(A, s) \wedge [actual(do(\alpha_s, s)) \supset holds(\Box A, do(\alpha_s, s))],$$

since α_s is the only action such that $actual(do(\alpha_s, s))$, we can derive:

$$holds(A, s) \wedge (\forall a).[actual(do(a, s)) \supset holds(\Box A, do(\alpha_s, s))],$$

which is equivalent to:

$$holds(A, s) \wedge holds(\bigcirc\Box A, s).$$

- $\Lambda = \Box(A \to \bigcirc A) \to (A \to \Box A)$. Given that s is actual, we show that:

$$holds(\Box(A \to \bigcirc A) \to (A \to \Box A), s) \tag{A.30}$$

follows from the situation calculus axioms. In fact, by applying the translation axioms we arrive at the following equivalent expression:

$$holds(\Box(A \to \bigcirc A), s) \wedge holds(A, s) \supset holds(\Box A, s). \tag{A.31}$$

From where we also obtain:

$$(\forall s').[[s' \geq s \wedge actual(s') \wedge holds(A, s') \supset holds(\bigcirc A, s')] \\ \wedge holds(A, s)] \supset holds(\Box A, s). \tag{A.32}$$

To prove this last expression, we introduce a new situation constant S, corresponding to an arbitrary actual situation, to the language and prove that the same sentence is true for this arbitrary constant (thus eliminating the quantifier). Furthermore, let us call the left hand side of (A.32) LHS. In order to prove (A.32), we will assume LHS to be true, and use our induction axiom (2.1) to infer that the right hand side of (A.32) must be true.

Let $\varphi(s) = s \geq S \wedge actual(s) \supset holds(A, s)$. That is:

$$(\forall s).\varphi(s) \equiv holds(\Box A, S).$$

Obviously $\varphi(S_0)$ is true, since $S_0 \geq S \supset S_0 = S$ and from LHS we have $holds(A, S)$. Now, from LHS we get:

$$(\forall s').s' \geq S \wedge actual(s') \wedge holds(A, s') \supset holds(\bigcirc A, s'). \tag{A.33}$$

Further manipulation leads to:

$$(\forall a, s').s' \geq S \wedge actual(do(a, s')) \wedge holds(A, s') \supset holds(A, do(a, s')). \tag{A.34}$$

From (A.34) and assuming $\varphi(S')$ to be true for an arbitrary constant S' we obtain:

$$(\forall a).S' \geq S \wedge actual(do(a, S')) \supset holds(A, do(a, S')). \tag{A.35}$$

From where we infer $(\forall a).\varphi(do(a, S'))$. Therefore, we have proven that:

$$LHS \supset \varphi(S_0) \wedge (\forall s).[\varphi(s) \supset (\forall a).\varphi(do(a, s))].$$

By direct application of (2.1) we get:

$$LHS \supset (\forall s).\varphi(s).$$

Which completes the proof.

- $\Lambda = A \ \mathcal{U} \ B \to \Diamond B$. We show that $holds(A \ \mathcal{U} \ B \to \Diamond B, s)$ is true, given that s is actual. In fact, by applying the transformation rules, this is equivalent to:

  $$hold(A \ \mathcal{U} \ B, s) \supset holds(\Diamond B, s).$$

  The left hand side gets translated to:

  $$(\exists s').(s' \geq s) \wedge actual(s') \wedge holds(B, s') \wedge \\ (\forall s'').[(s \leq s'' < s') \supset holds(A, s'')].$$

  The first conjunct of this expression corresponds to $holds(\Diamond B, s)$, therefore, the implication is immediate.

- $\Lambda = A \,\mathcal{U}\, B \leftrightarrow B \vee (A \wedge \bigcirc(A \,\mathcal{U}\, B))$. Here we will apply simple transformations to the left hand side of the equivalence and transform it into the right hand side. Again, given that s denotes an actual situation, $holds(A \,\mathcal{U}\, B, s)$ is translated as:

$$\begin{aligned}&(\exists\, s').(s' \geq s) \wedge actual(s') \wedge holds(B, s') \wedge\\ &\qquad\qquad (\forall\, s'').[(s \leq s'' < s') \supset holds(A, s'')],\end{aligned}$$

which is equivalent to:

$$\begin{aligned}&(\exists\, s').(s' = s) \wedge actual(s') \wedge holds(B, s')\\ &\vee(s' > s) \wedge actual(s') \wedge holds(B, s') \wedge\\ &\qquad\qquad (\forall\, s'').[(s \leq s'' < s') \supset holds(A, s'')],\end{aligned}$$

this can be written as:

$$\begin{aligned}&holds(B, s) \vee\\ &(\exists\, s').(s' > s) \wedge actual(s') \wedge holds(B, s') \wedge\\ &\qquad\qquad (\forall\, s'').[(s \leq s'' < s') \supset holds(A, s'')],\end{aligned}$$

which can be rewritten as:

$$\begin{aligned}&holds(B, s) \vee\\ &holds(A, s) \wedge (\exists\, s').(s' > s) \wedge actual(s') \wedge holds(B, s') \wedge\\ &\qquad\qquad (\forall\, s'').[(s < s'' < s') \supset holds(A, s'')],\end{aligned}$$

within the scope of the existential quantifier, s' denotes an actual situation, therefore, the s'' within the scope of the universal quantifier must also be actual (since it is less than s'). It can be easily proven that:

$$actual(s_2) \wedge s_1 < s_2 \wedge actual(do(a, s_1)) \supset do(a, s_1) \leq s_2.$$

Therefore, we can rewrite the previous expression as:

$$\begin{aligned}&holds(B, s) \vee\\ &holds(A, s) \wedge (\forall\, a).actual(do(a, s)) \supset\\ &\qquad (\exists\, s').(s' \geq do(a, s)) \wedge actual(s') \wedge holds(B, s') \wedge\\ &\qquad\qquad (\forall\, s'').[(do(a, s) \leq s'' < s') \supset holds(A, s'')],\end{aligned}$$

which corresponds to:

$$holds(B \vee (A \wedge \bigcirc(A \,\mathcal{U}\, B)), s).$$

□

Proposition 7.9. It follows directly from the induction axiom and the translation rules.

Appendix B

Circumscription.

B.1 Parallel Circumscription.

The discussion in this section is based on [26].

In [36, 37] McCarthy proposed to base non-monotonic reasoning on the notion of truth in a set of most preferred models of a logical theory. Preferred models are selected based on the *minimization* of the extension of some predicate or predicates.

Assume that $\mathcal{L}$ is a second order language. Let $\mathbf{P}$ be a tuple of predicate constants and $\mathbf{Z}$ be a tuple of function and/or predicate constants disjoint with $\mathbf{P}$. Let $\Sigma(\mathbf{P},\mathbf{Z})$ be a sentence in the language $\mathcal{L}$. The circumscription of $\mathbf{P}$ in Σ with variable $\mathbf{Z}$ is denoted as[1]:

$$Circ(\Sigma(\mathbf{P},\mathbf{Z});\mathbf{P};\mathbf{Z}).$$

and corresponds to the second order sentence:

$$\Sigma(\mathbf{P},\mathbf{Z}) \wedge \neg(\exists\,\mathbf{p},\mathbf{z})(\Sigma(\mathbf{p},\mathbf{z}) \wedge \mathbf{p} < \mathbf{P}).$$

Where $\mathbf{p}$ and $\mathbf{z}$ are tuples of variables of the same type as those in $\mathbf{P}$ and $\mathbf{Z}$ respectively[2]. Furthermore, if Q and Q' are two predicate constants, then $Q \leq Q'$ is an abbreviation for $(\forall\, x)Q(x) \supset Q'(x)$. Furthermore, if $\mathbf{P}$ and $\mathbf{P}'$ are tuples of predicates of the same type, then $\mathbf{P} \leq \mathbf{P}'$ is an abbreviation for:

$$P_1 \leq P_1' \wedge \ldots P_n \leq P_n'$$

where $\mathbf{P} = P_1,\ldots,P_n$ and $\mathbf{P}' = P_1',\ldots,P_n'$. $\mathbf{P} < \mathbf{P}'$ stands for $\mathbf{P} \leq \mathbf{P}' \wedge \neg\mathbf{P}' \leq \mathbf{P}$.

Here, we describe the model theoretic meaning of this circumscription. For a more comprehensive discussion refer to [36, 37, 26, 46, 13].

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two arbitrary models of the theory Σ in the language $\mathcal{L}$ with the tuples $\mathbf{P}$ and $\mathbf{Z}$ as before. We write $\mathcal{M}_1 \leq^{\mathbf{P};\mathbf{Z}} \mathcal{M}_2$ if:

1. $\|\mathcal{M}_1\| = \|\mathcal{M}_2\|$. That is, $\mathcal{M}_1$ and $\mathcal{M}_2$ share the same domain.

2. $\mathcal{M}_1$ and $\mathcal{M}_2$ interpret every constant not in $\mathbf{P},\mathbf{Z}$ in the same way

[1]Later we write $Circ(\Sigma;\mathbf{P};\mathbf{Z})$ for a theory Σ, in which case the sentence being circumscribed corresponds to the conjunction of all the sentences in Σ.

[2]We consider a predicate or function constant R to be of the same type as R' if they have the same arity and the sorts of their respective arguments are the same.

3. The interpretation of P_i in $\mathcal{M}_1$ is a subset (not necessarily proper) of the extension of P_i in $\mathcal{M}_2$ for every P_i in $\mathbf{P}$.

A model $\mathcal{M}$ of Σ is minimal with respect to $\leq^{\mathbf{P};\mathbf{Z}}$, if there is no other model $\mathcal{M}'$ of Σ such that $\mathcal{M}' <^{\mathbf{P};\mathbf{Z}} \mathcal{M}$. Here, $\mathcal{M}' <^{\mathbf{P};\mathbf{Z}} \mathcal{M}$ if $\mathcal{M}' \leq^{\mathbf{P};\mathbf{Z}} \mathcal{M}$ but not $\mathcal{M} \leq^{\mathbf{P};\mathbf{Z}} \mathcal{M}'$. Also, we say that $\mathbf{P}$ in $\mathcal{M}_1$ is smaller than $\mathbf{P}$ in $\mathcal{M}_2$ if the interpretation of P_i in $\mathcal{M}_1$ is a subset (not necessarily proper) of the extension of P_i in $\mathcal{M}_2$ for every P_i in $\mathbf{P}$, and for some P_j in $\mathbf{P}$ the interpretation of P_j in $\mathcal{M}_1$ is a proper subset of the extension of P_j in $\mathcal{M}_2$.

Finally, it follows that $\mathcal{M}$ is a model of

$$Circ(\Sigma; \mathbf{P}; \mathbf{Z})$$

if and only if $\mathcal{M}$ is a minimal model of Σ with respect to $\leq^{\mathbf{P};\mathbf{Z}}$.

B.2 Circumscribing Skolemized Theories

It is well understood that existential theories are not logically equivalent to their skolemized versions. One of the consequences of this fact is that skolemization may change the set of minimal models of a theory [13]. On the positive side, we know that skolemization preserves satisfiability. Now, we would like to know if it is possible to retain this property by changing the circumscription policy. Thus, can we define a *skolemized circumscription*? In this appendix we show that this is indeed possible. In summary, we show that we preserve satisfiability if we let the skolem function symbols vary in the new circumscription policy.

Theorem B.2.1 *Let:*

- *Σ be a theory in a second order language $\mathcal{L}$.*
- *Σ^s be a skolemized version of Σ in the language $\mathcal{L}^s = \mathcal{L} \cup \mathbf{S_k}$, in which $\mathbf{S_k}$ denotes the tuple of skolem function constants introduced by the skolemization.*
- *$C = Circ(\Sigma; \mathbf{P}; \mathbf{Z})$ be the circumscription of the predicates in $\mathbf{P}$ in the theory Σ having the interpretation of the tuple of predicates and/or function constants in $\mathbf{Z}$ vary.*
- *$C^s = Circ(\Sigma^s; \mathbf{P}; \mathbf{Z}, \mathbf{S_k})$ be the circumscription of the predicates $\mathbf{P}$ in the theory Σ^s having the interpretation of the tuple of predicates and/or function constants in $\mathbf{Z}$ and the skolem function constants $\mathbf{S_k}$ vary.*
- *$\mathcal{M}$ and $\mathcal{M}'$ denote interpretations for $\mathcal{L}$.*
- *$\mathcal{M}^s$ and $\mathcal{M}^{s\prime}$ denote interpretations for $\mathcal{L}^s$.*
- *$sub(\mathcal{M}^s)$ denote the subinterpretation of $\mathcal{M}^s$ in which the interpretations for the skolem function constants have been dropped. Thus, $sub(\mathcal{M}^s)$ is an interpretation for $\mathcal{L}$.*

The following holds:

1. *For every model $\mathcal{M}$ of C there exists a model $\mathcal{M}^s$ of C^s, s.t. $\mathcal{M} = sub(\mathcal{M}^s)$.*
2. *If $\mathcal{M}^s$ is a model of C^s, then $sub(\mathcal{M}^s)$ is a model of C.*

Thus, if φ is any sentence in the language $\mathcal{L}$, then:

$$C \models \varphi$$

if and only if

$$C^s \models \varphi.$$

In summary, if we use circumscription for a theory Σ, and we want to skolemize Σ, then we need to let the skolem constants vary in the circumscription of the new theory in order to preserve satisfiability. Furthermore, it is well known that if the constants are not allowed to vary the previous result does not hold [13].

Before we present a proof of the theorem, we state the basic properties of skolemization in the following lemma:

Lemma B.2.1 *Using the notation of the statement of the theorem B.2.1, the following holds:*

1. *For every model $\mathcal{M}$ of Σ there exists a model $\mathcal{M}^s$ of Σ^s, s.t. $\mathcal{M} = sub(\mathcal{M}^s)$.*
2. *If $\mathcal{M}^s$ is a model of Σ^s, then $sub(\mathcal{M}^s)$ is a model of Σ.*

We use these properties of skolemization to construct the proof of the theorem.

Proof of theorem B.2.1. In the proof we write $<$ instead of $<^{\mathbf{P};\mathbf{Z}}$ and $<^s$ instead of $<^{\mathbf{P};\mathbf{Z},\mathbf{S_k}}$.

The theorem is proved by contradiction:

1. Assume that there is a model $\mathcal{M}$ of C for which there is no model $\mathcal{M}^s$ of C^s, such that $\mathcal{M} = sub(\mathcal{M}^s)$. Since $\mathcal{M}$ is a model of Σ, from the previous lemma we infer that there exists a model $\mathcal{M}^s$ of Σ^s, such that $\mathcal{M} = sub(\mathcal{M}^s)$. From our assumption, $\mathcal{M}^s$ cannot be a model of C^s. Thus, there must be a model $\mathcal{M}^{s\prime}$ of Σ^s such that $\mathcal{M}^{s\prime} <^s \mathcal{M}^s$. That is, $\mathbf{P}$ in $\mathcal{M}^{s\prime}$ is smaller than $\mathbf{P}$ in $\mathcal{M}^s$. Furthermore, $\mathcal{M}^s$ differs from $\mathcal{M}^{s\prime}$ only in the interpretation of $\mathbf{P}$, $\mathbf{Z}$ and $\mathbf{S_k}$. From the lemma, it follows that $sub(\mathcal{M}^{s\prime})$ is a model of Σ. Also, $sub(\mathcal{M}^{s\prime})$ and $sub(\mathcal{M}^s)$ only differ in the interpretation of $\mathbf{P}$ and $\mathbf{Z}$. Also, the $\mathbf{P}$ in $sub(\mathcal{M}^{s\prime})$ is smaller than $\mathbf{P}$ in $sub(\mathcal{M}^s)$. Since $\mathcal{M} = sub(\mathcal{M}^s)$, $\mathcal{M}$ cannot be minimal and therefore cannot be a model of C. Thus, the assumption is contradicted.

2. Assume that there is a model $\mathcal{M}^s$ of C^s and that $sub(\mathcal{M}^s)$ is not a model of C. From the lemma, $sub(\mathcal{M}^s)$ is a model of Σ. Since it is not minimal, there is some other model $\mathcal{M}$, such that $\mathcal{M} < sub(\mathcal{M}^s)$. Thus, $\mathcal{M}$ differs from $sub(\mathcal{M}^s)$ only in the way they interpret $\mathbf{P}$ and $\mathbf{Z}$. From the lemma, there is a model $\mathcal{M}^{s\prime}$ of Σ^s, such that $\mathcal{M} = sub(\mathcal{M}^{s\prime})$. Obviously, $\mathcal{M}^{s\prime}$ and $\mathcal{M}^s$ only differ on how they interpret $\mathbf{P}$, $\mathbf{Z}$ and $\mathbf{S_k}$. Furthermore, $\mathbf{P}$ in $\mathcal{M}^{s\prime}$ (which is the same as the $\mathbf{P}$ in $\mathcal{M}$) is smaller than $\mathbf{P}$ in $\mathcal{M}^s$ (which is the same $\mathbf{P}$ in $sub(\mathcal{M}^s)$). Thus, $\mathcal{M}^{s\prime} <^s \mathcal{M}^s$. Therefore, $\mathcal{M}^s$ cannot be minimal. Thus, $\mathcal{M}^s$ is not a model C^s, contradicting the initial assumption.

$\square$

Bibliography

[1] Allen, J., and Ferguson, G. Action in interval temporal logic. In *The Second Symposium on Logical Formalizations of Commonsense Reasoning* (Jan. 1993), pp. 12–22.

[2] Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM 26*, 11 (1983), 832–843.

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

[4] Allen, J. F., and Hayes, P. J. A common-sense theory of time. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)* (1985), pp. 528–531.

[5] Allen, J. F., and Hayes, P. J. Moments and points in an interval-based temporal logic. *Computational Intelligence 5* (Dec. 1989), 225–238.

[6] Allen, J. F., and Koomen, J. A. Planning using a temporal world model. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)* (1983), pp. 741–747.

[7] Apt, K., and Bezem, M. Acyclic programs. In *Logic Programming: Proceedings of the Seventh International Conference.* (1990), D. Warren and P. Szeredi, Eds., pp. 617–633.

[8] Bacchus, F., Tenenberg, J., and Koomen, J. A Non-Reified Temporal Logic. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (Toronto, Ontario, Canada, May 1989), R. Brachman, H. Levesque, and R. Reiter, Eds., Morgan Kaufmann, pp. 2–10.

[9] Baker, A. A Simple Solution to the Yale Shooting Problem. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (May 1989), H. L. R. Brachman and R. Reiter, Eds., Morgan Kaufmann, pp. 11–20.

[10] Baker, A. Nonmonotonic Reasoning in the Framework of the Situation Calculus. *Artificial Intelligence 49* (1991).

[11] Clark, K. Negation as Failure. In *Logic and Databases*, H. Gallaire and J. Minker, Eds. Plenum Press, New York, 1978, pp. 293–322.

[12] Davis, E. Infinite loops in finite time: Some observations. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)* (1992), C. R. B. Nebel and W. Swartout, Eds., Morgan Kaufmann Publishers, Inc., pp. 47–58.

[13] Etherington, D. W. *Reasoning with Incomplete Information, Investigations of Non-Monotonic Reasoning.* PhD thesis, The University of British Columbia, Apr. 1986.

[14] Galton, A. A critical examination of Allen's theory of action and time. *Artificial Intelligence 42*, 2-3 (1990), 159–188.

[15] Gelfond, M., and Lifschitz, V. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9* (1991), 365–385.

[16] Gelfond, M., Lifschitz, V., and Rabinov, A. What are the Limitations of the Situation Calculus? In *Working Notes, AAAI Spring Symposium Series. Symposium:Logical Formalization of Commonsense Reasoning.* (Mar. 1991), pp. 59–69.

[17] Genesereth, M. R., and Nilsson, N. J. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., Palo Alto, California, U.S.A., 1988.

[18] Georgeff, M. P. Many agents are better than one. In *The Frame Problem in Artificial Intelligence* (Los Altos, California, Apr. 1987), F. M. Brown, Ed., Morgan Kaufmann, pp. 58–76.

[19] Goldblatt, R. *Logics of Time and Computation.* CSLI, 1987.

[20] Haas, A. R. The Case for Domain-Specific Frame Axioms. In *The Frame Problem in Artificial Intelligence, Proceedings of the 1987 Workshop* (Los Altos, CA, 1987), F. M. Brown, Ed., Morgan Kaufmann, pp. 343–348.

[21] Hanks, S., and McDermott, D. Nonmonotonic logic and temporal projection. *Artificial Intelligence 33*, 3 (1987), 379–412.

[22] Kautz, H. The logic of persistence. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)* (1986), pp. 401–405.

[23] Kowalski, R. Database Updates in the Event Calculus. Tech. Rep. Doc 86/12, Imperial College, July 1986.

[24] Kowalski, R. Database Updates in the Event Calculus. *The Journal of Logic Programming 12* (1992), 121–146.

[25] Kowalski, R., and Sergot, M. A logic-based calculus of events. *New Generation Computing 4* (1986), 67–95.

[26] Lifschitz, V. Computing Circumscription. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)* (1985), pp. 121–127.

[27] Lifschitz, V. Formal Theories of Action. In *The Frame Problem in Artificial Intelligence* (Los Altos, California, Apr. 1987), F. M. Brown, Ed., Morgan Kaufmann, pp. 35–57.

[28] Lifschitz, V. Toward a metatheory of action. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning* (1991), J. Allen, R. Fikes, and E. Sandewall", Eds., Morgan Kaufmann, pp. 376–386.

[29] Lifschitz, V., and Rabinov, A. Miracles in Formal Theories of Action. *Artificial Intelligence 38* (1989), 225–237.

[30] Lin, F., Levesque, H., Reiter, R., and Scherl, R. Complex events in the situation calculus. Forthcoming, 1993.

[31] Lin, F., and Reiter, R. State constraints revisited. In *The Second Symposium on Logical Formalizations of Commonsense Reasoning* (Jan. 1993), pp. 114–121.

[32] Lin, F., and Shoham, Y. Provably Correct Theories of Action (preliminary report). In *Proceedings of the ninth National Conference on Artificial Intelligence (AAAI-91)* (1991), pp. 349–354.

[33] Lin, F., and Shoham, Y. Provably correct theories of action. Tech. rep., University of Toronto, 1992.

[34] Maruyama, F., and Fujita, M. Hardware Verification. *IEEE Computer* (February 1985), 22–32.

[35] McCarthy, J. Programs with common sense. In *Semantic Information Processing*, M. Minsky, Ed. The MIT Press, 1968, ch. 7, pp. 403–418.

[36] McCarthy, J. Circumscription a form of non-monotonic reasoning. *Artificial Intelligence 13* (1980), 27–39.

[37] McCarthy, J. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence 28* (1986), 89–116.

[38] McCarthy, J., and Hayes, P. J. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence* **4**, B. Meltzer and D. Michie, Eds. Edinburgh University Press, Edinburgh, Scotland, 1969, pp. 463–502.

[39] Miller, R., and Shanahan, M. Narratives in the situation calculus. Tech. rep., Imperial College, Department of Computing, 1993.

[40] Pednault, E. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence 4* (1988), 356–372.

[41] Pinto, J., and Reiter, R. Adding a time line to the situation calculus. In *The Second Symposium on Logical Formalizations of Commonsense Reasoning* (Jan. 1993), pp. 172–177.

[42] Przymusinski, T. C. On the declarative semantics of deductive databases and logic programs. ch. 5, pp. 193–216.

[43] Rayner, M. On the applicability of nonmonotonic logic to formal reasoning in continuous time. *Artificial Intelligence 49* (1991), 345–360.

[44] R.C.Holt, G.S.Graham, E.D.Lazowska, and M.A.Scott. *Structured Concurrent Programming with Operating Systems Applications.* Addison-Wesley, 1978.

[45] Reichgelt, H. A comparison of first order and modal logics of time. In *Logic-Based Knowledge Representation.* The MIT Press, 1989, pp. 143–76.

[46] Reiter, R. *Nonmonotonic Reasoning*, vol. 2 of *Annual Review of Computer Science.* Annual Reviews Inc., 1987, pp. 147–86.

[47] Reiter, R. *The Frame Problem in the Situation Calculus: A Simple Solution (sometimes) and a completeness result for goal regression.* Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy. Academic Press, San Diego, CA, 1991, pp. 359–380.

[48] REITER, R. On specifying database updates. Tech. Rep. KRR-TR-92-3, University of Toronto, July 1992.

[49] REITER, R. Proving Properties of States in the Situation Calculus. Submitted for publication, Feb. 1992.

[50] SANDEWALL, E. Combining Logic and Differential Equations for Describing Real-World Systems. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (Toronto, Ontario, Canada, May 1989), H. L. R. Brachman and R. Reiter, Eds., Morgan Kaufmann, pp. 412–420.

[51] SANDEWALL, E. Filter preferential entailment for the logic of action in almost continuous worlds. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-89)* (Detroit, MI, 1989), pp. 894–899.

[52] SANDEWALL, E. Features and Fluents. Tech. Rep. LiTH-IDA-R-91-29, Linköping University, Aug. 1991.

[53] SANDEWALL, E. Features and Fluents, second review version. Tech. Rep. LiTH-IDA-R-92-30, Linköping University, Sept. 1992.

[54] SCHUBERT, L. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In *Knowledge Representation and Defeasible Reasoning*, H. Kyberg, R. Loui, and G. Carlson, Eds. Kluwer Academic Press, 1990, pp. 23–67.

[55] SHANAHAN, M. Representing continuous change in the event calculus. In *Proceedings ECAI* (1990).

[56] SHANAHAN, M. A circumscriptive calculus of events. Tech. rep., Imperial College, Department of Computing, 1992.

[57] SHAPIRO, S. First-order Solutions to the Frame Problem. Master's thesis, University of Toronto, Oct. 1993.

[58] SHOHAM, Y. *Reasoning about Change.* The MIT Press, Cambridge, Mass., 1988.