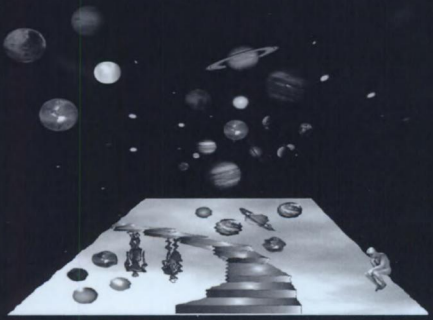
KNOWLEDGE IN ACTION

Logical Foundations for Specifying and Implementing Dynamical Systems

RAYMOND REITER



Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems

Raymond Reiter

The MIT Press
Cambridge, Massachusetts
London, England

This book was set in Times Roman by the author using the LaTeX document preparation system and was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Reiter, Raymond.
Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems / Raymond Reiter
p. cm.
Includes bibliographical references and index.
ISBN 0-262-18218-1 (hc.: alk. paper)
1. Knowledge representation (Information theory) 2. Expert systems (Computer science). 3. Logic, symbolic and mathematical.
I. Title.
Q387 .R48 2001
006.3′32—dc21 2001030522
CIP

Acknowledgments

This book began in 1992, while Hector Levesque and I were at a retreat sponsored by the Canadian Institute for Advanced Research. It was a lovely spring afternoon, and we went out for a walk. I had recently written a paper on solving the frame problem in the situation calculus, and Hector suggested that maybe this solution could serve as a foundation for some kind of programming language. From this suggestion came Golog and a long-term effort by us and our collaborators to enrich the expressivity of the situation calculus. Much of this book is the result of that research program, and Hector's intellectual contributions are present on every page.

I had the good luck to write large chunks of this book during several wonderful summers in Rome. Gigina Aiello and Fiora Pirri made this possible by providing me with a second academic home at the University of Rome, La Sapienza. Grazie Fiora. Grazie Gigina. I am also grateful to Dov Gabbay who, in his usual resourceful way, managed to find me a fellowship to King's College, London, in the spring of 1998, during which time I worked on some of the ideas for this book.

In developing Chapters 10 and 12, I was on very shaky ground since I had very little previous experience with planning, and virtually none in probability and decision theory. Fahiem Bacchus was a great help in guiding me through the treacherous literature on planning. Craig Boutilier patiently acted as my primary informant about what's going on these days in probability and decision theory; he also provided valuable feedback on a preliminary draft of Chapter 12. Fahiem also helped out here by advising me on some of the many subtleties in the foundations of probability theory. Many others provided feedback on earlier drafts of parts of this book, and I'd like here to thank them for their contributions: Gerd Brewka, Robert Demolombe, Hojjat Ghaderi, Arie Gurfinkel, Pat Hayes, Danny House, Eric Joanis, Hesham Khalil, Iluju Kiringa, Yves Lespérance, Mohammad Mahdian, Victor Marek, Maurice Pagnucco, Dimitrie Paun, Javier Pinto, Fiora Pirri, Angus Stewart, and Michael Thielscher.

Working with a physical robot focuses the mind in ways that no amount of armchair theorizing can do. In getting our group's robot off the ground, we were helped enormously by Dieter Fox and Sebastian Thrun, who provided us with their worry-free RWI B21 navigation software, and who visited us on several occasions to help with its installation.

A number of the ideas in this book have been influenced by, and in many cases developed by, past and present members of the University of Toronto Cognitive Robotics Group. My thanks for your many contributions to: Alfredo Gabaldon, Sam Kaufman, Todd Kelley, Iluju Kiringa, Yves Lespérance, Hector Levesque, Fangzhen Lin, Yongmei Liu, Daniel Marcu, Sheila McIlraith, Javier Pinto, Richard Scherl, Sebastian Sardiña, Steven Shapiro, Misha Soutchanski, and Eugenia Ternovskaia. Apart from members of the group, discussions with a wide variety of people have influenced my thinking on dynamical systems. The list reads like a who's who of researchers in knowledge representation and the theory

of actions: Fahiem Bacchus, Chitta Baral, Alex Borgida, Craig Boutilier, Gerd Brewka, Tom Costello, Giuseppe De Giacomo, Robert Demolombe, Marc Denecker, Pat Doherty, Charles Elkan, Alberto Finzi, Michael Fisher, Dov Gabbay, Michael Gelfond, Neelakantan Kartha, Bob Kowalski, Gerhard Lakemeyer, Vladimir Lifschitz, Jorg Lobo, Alan Mackworth, Victor Marek, John McCarthy, John-Jules Meyer, Rob Miller, John Mylopoulos, Maurice Pagnucco, Edwin Pednault, Fiora Pirri, David Poole, Stuart Russell, Fariba Sadri, Erik Sandewall, Len Schubert, Marek Sergot, Murray Shanahan, and Michael Thielscher. Thanks to you all.

One of the joys of doing science in Canada is that, by and large, its funding agencies make little attempt to steer researchers towards whatever are the currently fashionable application areas. I have benefited over the years from this no-strings-attached funding policy from the Canadian Institute for Advanced Research, the National Science and Engineering Research Council of Canada, the Institute for Robotics and Intelligent Systems, and the Centre for Information Technology of Ontario, and I want here to acknowledge their wisdom in sponsoring curiosity-driven research.

Finally, I'd like to thank Bob Prior, Katherine Innis, and the other staff at MIT Press for their help in getting this book out, Eugenia Ternovskaia for patiently preparing all the figures, Antonina Kolokolova for help with Latex fonts, and Gerhard Lakemeyer for providing me with his fine tuning of MIT's macros.

Ray Reiter
Toronto
January, 2001

1 Introduction

Heraclitus reminds us that we cannot step into the same river twice. As goes the river, so goes virtually every modeling task in artificial intelligence, computer animation, robotics, software agents, decision and control theory, simulation, databases, programming languages, etc. The world simply won't sit still, and all attempts to model any but its simplest features must take change seriously. It is not trivial to address this problem in its full generality, as suggested by the following partial list of phenomena that a comprehensive theory of dynamical systems and autonomous agents must accommodate:

- The causal laws relating actions to their effects.
- The conditions under which an action can be performed.
- Exogenous and natural events.
- Probabilistic action occurrences and action effects.
- Decision theory: Determining what to do and when to do it.
- Complex actions and procedures.
- Discrete and continuous time.
- Concurrency.
- Continuous processes.
- Hypothetical and counterfactual reasoning about action occurrences and time.
- Perceptual actions and their effects on an agent's mental state.
- Deciding when to look and what to look for.
- Unreliable sensors and effectors.
- Agent beliefs, desires and intentions and how these influence behaviour.
- Real time (resource bounded) behaviour.
- Non deliberative (reactive) behaviour.
- Revising an agent's beliefs in the presence of conflicting observations.
- Planning a course of actions.
- Execution monitoring of a course of actions; recognizing and recovering from failures.

Despite the many existing disciplines that focus on modeling dynamical systems of one kind or another, a story as general as this has yet to be told. This is not to say that your average system modeler lacks for tools of the trade; there are plenty of formalisms to choose from, including Petri nets, process algebras, dynamic and temporal logics, finite automata, Markov decision processes, differential equations, STRIPS operators, influence diagrams, etc. But as this list suggests, what's available is more like a Tower of Babel than

a unifying representational and computational formalism. To be fair, this state of affairs is the natural outcome of disciplines organized by their applications; discrete event control theory is concerned with different problems than, say, programming language design, and neither appear to have anything in common with semantics for tense in natural language. We all solve problems that arise in our own, sometimes narrowly circumscribed fields of specialization, and in this sense, we are like the proverbial blind men, each acting in isolation, and each trying to figure out the elephant. Nevertheless, one can't help thinking that there really is an elephant out there, that at a suitable level of abstraction, there must be a unifying "theory of dynamics", one that subsumes the many special purpose mechanisms that have been developed in these different disciplines, and that moreover accommodates all the features of autonomous dynamical systems listed above.

During the past 15 years or so, a number of researchers in artificial intelligence have been developing mathematical and computational foundations for dynamical systems that promise to deliver the elephant.[1] The methodological foundations of all these approaches —indeed, of much of the theory and practice of artificial intelligence—are based on what Brian Smith [202] has called the *Knowledge Representation Hypothesis*:

> *Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge.*

Adopting this hypothesis has a number of important consequences:

1. We are naturally led to employ mathematical logic as a foundation for the "propositional account of the knowledge that the overall process exhibits" called for in part a) of the hypothesis.
2. This "propositional account" differs substantially from the state-based approaches central, for example, to decision and control theory. Instead of explicitly enumerating states and their transition function, the Knowledge Representation Hypothesis favours sentences—descriptions of what is true of the system and its environment, and of the causal laws in effect for that domain.
3. Part b) of the Knowledge Representation Hypothesis calls for a causal connection between these sentences and the system's *behaviours*. How can sentences lead to behaviour? In logic, sentences beget sentences through logical entailment, so it is natural to view system behaviours as appropriate *logical consequences* of the propositional account of the domain. On this perspective, the *computational* component of a logical representation for a dynamical system consists of deduction. Determining how a sys-

1 Needless to say, this is still just a promise.

tem behaves amounts to *deducing how it must behave*, given the system's description.

4. Providing a propositional account for some domain amounts to giving an abstract *specification* for that problem. Even by itself, having a non-procedural specification for a problem domain is a good thing; at least it's clear what modeling assumptions are being made. But in addition to this, because these are logical specifications, one can hope to prove, entirely within the logic, various properties of the specification. In other words, there is a direct mechanism, namely logical deduction, for establishing correctness properties for the system.
5. In those cases where deduction can be performed efficiently, these system specifications are also *executable*. This means that, as a side effect of providing a logical specification, we often obtain a simulator for the system.

This book deals with a logical approach to modeling dynamical systems based on a dialect of first order logic called the *situation calculus*, a language first proposed by John McCarthy in 1963 [136]. The material presented here has evolved over a number of years in response to the needs of the University of Toronto Cognitive Robotics Project, and therefore, has been heavily influenced by the problems that arise there. Broadly speaking, the newly emerging field of cognitive robotics has, as its long term objectives, the provision of a uniform theoretical and implementation framework for autonomous robotic or software agents that reason, act and perceive in changing, incompletely known, unpredictable environments. It differs from "traditional" robotics research in emphasizing "higher level" cognition as a determiner for agent behaviours. Therefore, one focus of cognitive robotics is on modeling an agent's beliefs and their consequences. These include beliefs about what is true of the world it inhabits, about the actions that it and other agents (nature included) can perform and the effects of those actions on the agent and its environment, about the conditions under which such actions can be performed, about the mental states and physical abilities of its fellow agents in the world, and about the outcomes of its perceptions. In keeping with the Knowledge Representation Hypothesis, these beliefs are represented as logical sentences, in our case, using the situation calculus. Such beliefs condition behaviour in a variety of ways, and one goal of cognitive robotics is to provide a theoretical and computational account of exactly how it is that deliberation can lead to action. None of this is meant to suggest that these objectives are peculiar to cognitive robotics; many control theorists and roboticists are concerned with modeling similar phenomena. What distinguishes cognitive robotics from these other disciplines is its emphasis on beliefs and how they condition behaviour, and by its commitment to the Knowledge Representation Hypothesis, and therefore, to logical sentences as the fundamental mathematical representation for dynamical systems and agent belief states. Nevertheless, despite these differences in emphasis and methodology, the representational and computational problems that

Uppercase Roman will be a constant.

- We will often omit leading universal quantifiers in writing sentences. The convention will be that any free variables in such formulas are implicitly universally quantified. So,

$$P(x, y) \supset (\exists z)[Q(y, z) \wedge R(x, w, z)]$$

abbreviates

$$(\forall x, y, w)[P(x, y) \supset (\exists z)[Q(y, z) \wedge R(x, w, z)]].$$

- The "dot" notation: In logical languages, a quantifier's scope must be indicated explicitly with parentheses. An alternative notation, which often obviates the need for explicit parentheses, is the "dot" notation, used to indicate that the quantifier preceding the dot has maximum scope. Thus, $(\forall x).P(x) \supset Q(x)$ stands for $(\forall x)[P(x) \supset Q(x)]$.

$$[(\forall x)(\exists y).A(x, y) \wedge B(x, y) \supset C(x, y)] \wedge R(x, y)$$

stands for

$$[(\forall x)(\exists y)[A(x, y) \wedge B(x, y) \supset C(x, y)]] \wedge R(x, y).$$

- Parenthesis reduction. We shall assume that $\wedge$ takes precedence over $\vee$, so that $P \wedge Q \vee R \wedge S$ stands for $(P \wedge Q) \vee (R \wedge S)$. Also, $\supset$ and $\equiv$ bind with lowest precedence, so $P \wedge Q \supset R \vee S$ stands for $(P \wedge Q) \supset (R \vee S)$, and $P \wedge Q \equiv R \vee S$ stands for $(P \wedge Q) \equiv (R \vee S)$.

2.1.2 Semantics

We begin, as usual, with the definition of a *structure* (sometimes called an *interpretation*) for a first-order language. This will tell us

1. What collection of things the universal quantifier symbol ($\forall$) ranges over—the *domain or universe of quantification*.
2. What the other parameters—the predicate and function symbols—denote with respect to the domain of the structure.

Formally, a structure $\mathcal{S}$ for a given first-order language is a function whose domain is the set of parameters of the language, and is defined by:

1. $\forall^{\mathcal{S}}$ is a nonempty set, called the *universe* or the *domain* of the structure $\mathcal{S}$. The universe is usually written $|\mathcal{S}|$.
2. For each n-ary predicate symbol P of the first-order language, $P^{\mathcal{S}} \subseteq |\mathcal{S}|^n$. These n-tuples of the universe are understood to be all, and only, those tuples on which P is true in the structure. This is called the *extension* of P in the structure $\mathcal{S}$.
3. For each n-ary function symbol f of the first-order language, $f^{\mathcal{S}}$ is an n-ary function

on $|\mathcal{S}|$, i.e. $f^{\mathcal{S}} : |\mathcal{S}|^n \to |\mathcal{S}|$. In particular, when $n = 0$, so that f is a constant symbol, $f^{\mathcal{S}}$ is simply some element of the universe.

SEMANTICS: TRUTH IN A STRUCTURE

We want to define when a sentence σ is *true* in a structure $\mathcal{S}$, denoted by $\models_{\mathcal{S}} \sigma$. To do so, we need a more general notion of truth for a formula (not necessarily a sentence). Suppose

1. ϕ is a formula of the given first-order language.
2. $\mathcal{S}$ is a structure for the language,
3. $s : V \to |\mathcal{S}|$ is a function, called a *variable assignment*, from the set V of variables of the language into the universe of $\mathcal{S}$.

We can now define $\models_{\mathcal{S}} \phi[s]$, meaning that the formula ϕ is true in the structure $\mathcal{S}$ when its free variables are given the values specified by s in the universe.

1. **Terms**: Define an extension $\overline{s} : T \to |\mathcal{S}|$ of the function s from the set T of all terms of the language into the universe.
   (a) For each variable v, $\overline{s}(v) = s(v)$.
   (b) If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol, then
   $$\overline{s}(f(t_1, \ldots, t_n)) = f^{\mathcal{S}}(\overline{s}(t_1), \ldots, \overline{s}(t_n)).$$
2. **Atomic Formulas**:
   (a) For terms t_1 and t_2,
   $$\models_{\mathcal{S}} t_1 = t_2[s] \text{ iff } \overline{s}(t_1) = \overline{s}(t_2).^1$$
   (b) For an n-ary predicate symbol P,
   $$\models_{\mathcal{S}} P(t_1, \ldots, t_n)[s] \text{ iff } < \overline{s}(t_1), \ldots, \overline{s}(t_n) > \in P^{\mathcal{S}}.$$
3. **Well-Formed Formulas:**
   (a) $\models_{\mathcal{S}} \neg\phi[s]$ iff not $\models_{\mathcal{S}} \phi[s]$.
   (b) $\models_{\mathcal{S}} (\phi \supset \psi)[s]$ iff $\models_{\mathcal{S}} \neg\phi[s]$ or $\models_{\mathcal{S}} \psi[s]$.
   (c) $\models_{\mathcal{S}} (\forall x)\phi[s]$ iff for every $d \in |\mathcal{S}|$, $\models_{\mathcal{S}} \phi[s(x|d)]$.
   Here, $s(x|d)$ is the function that is exactly like s except that for the variable x it assigns the value d.

---

1 Notice that we are slightly abusing notation here by using the same symbol $=$ for two different purposes. We could have avoided this ambiguity by denoting the equality predicate symbol in the language of first-order logic by something different than $=$, for example $\approx$, reserving $=$ for its standard usage in the metalanguage. But that seemed a bit pedantic.

Our interest will always be in the truth of sentences, not of arbitrary formulas. We needed the above notions for well-formed formulas in order to give a recursive definition of truth in a structure, even when our only interest is in sentences. With this definition in hand, we can now consider only sentences.

1. **Satisfiability:** A structure $\mathcal{S}$ *satisfies* a sentence σ, or σ *is true* in $\mathcal{S}$, iff $\models_{\mathcal{S}} \sigma[s]$ for every variable assignment s. A sentence is *satisfiable* iff there is a structure that satisfies it.
2. **Models:** A structure $\mathcal{S}$ is a model of a sentence σ iff $\mathcal{S}$ satisfies σ. $\mathcal{S}$ is a model of a set of sentences, possibly infinite, iff it is a model of each sentence in the set.
3. **Validity**: The sentence σ is valid, written $\models \sigma$, iff every structure of the first-order language is a model of the sentence σ.
4. **Unsatisfiability:** A sentence is unsatisfiable iff it has no models.
5. **Logical entailment**: Suppose Γ is a set of sentences, possibly infinite, and σ is a sentence. $\Gamma \models \sigma$ (Γ *entails* σ) iff every model of Γ is a model of σ. It is a standard result of first-order logic that $\Gamma \models \sigma$ iff there is a finite subset Γ' of Γ such that $\models \bigwedge \Gamma' \supset \sigma$, where $\bigwedge \Gamma'$ denotes the conjunction of the sentences in Γ'. This is often called the *Compactness Theorem for first-order logic*.

2.1.3 Soundness and Completeness

First-order logic can be axiomatized using a suitable set of sentences called *axioms* (usually a finite set of sentence *schemas*), together with *rules of inference* that permit the derivation of new formulas from old. These systems are used to construct *proofs* of sentences from sets of sentences called *premises*. $A \vdash \sigma$ means that, with respect to some given axiomatization of first-order logic, there is a proof of σ from the premises A. An axiomatization of first-order logic is *complete* iff $A \models \sigma$ implies $A \vdash \sigma$. An axiomatization of first-order logic is *sound* iff $A \vdash \sigma$ implies $A \models \sigma$.

First-order logic has many sound and complete axiomatizations, all of them equivalent. Examples include Gentzen and Hilbert style systems, etc. In this book, we won't be much concerned with particular axiomatizations of first-order logic, although the need for theorem-proving will arise frequently. So the symbol "$\vdash$" will not be used very often. Virtually all our claims and arguments will be semantic.

First-order logic is *not* decidable; the non-valid sentences are not recursively enumerable, although the valid sentences are.

2.1.4 Many-Sorted First-Order Languages

In the situation calculus, we shall find it convenient to distinguish three kinds of objects—situations, actions and other things. To do this, we will appeal to a *sorted* first-order lan-

guage. In general, a sorted first-order language has variables and terms of different sorts. Semantically, the universe is partitioned into disjoint sub-universes, one such sub-universe for each sort. A variable will range over its own sub-universe, and a term will denote an element in its corresponding sub-universe. A predicate will be syntactically restricted to take arguments of certain prespecified sorts, as will functions. Moreover, functions will be required to deliver values of a prespecified sort. Many-sorted logical languages are strongly analogous to typed programming languages.

Syntax of a Many-Sorted First-Order Language. Assume given a nonempty set I, whose members are called *sorts*.

1. **Logical symbols:** As before, except that for each sort i, there are infinitely many variables $x_1^i, x_2^i, \ldots$, of sort i.
2. **Parameters**:
   (a) *Quantifier symbols:* For each sort i there is a universal quantifier symbol $\forall_i$.
   (b) *Predicate symbols:* For each $n \geq 0$ and each n-tuple $< i_1, \ldots, i_n >$ of sorts, there is a set (possibly empty) of n-ary predicate symbols, each of which is said to be of sort $< i_1, \ldots, i_n >$.
   (c) *Function symbols:* For each $n \geq 0$ and each $(n+1)$-tuple $< i_1, \ldots, i_n, i_{n+1} >$ of sorts, there is a set (possibly empty) of n-ary function symbols, each of which is said to be of sort $< i_1, \ldots, i_n, i_{n+1} >$.

Each term is assigned a unique sort, as follows:

1. Any variable of sort i is a term of sort i.
2. If $t_1, \ldots, t_n$ are terms of sort $i_1, \ldots, i_n$ respectively, and f is a function symbol of sort $< i_1, \ldots, i_n, i_{n+1} >$, then $f(t_1, \ldots, t_n)$ is a term of sort i_{n+1}.

Atomic formulas are defined as follows:

1. When t and t' are terms of the same sort, $t = t'$ is an atomic formula.
2. When P is an n-ary predicate symbol of sort $< i_1, \ldots, i_n >$ and $t_1, \ldots, t_n$ are terms of sort $i_1, \ldots, i_n$ respectively, then $P(t_1, \ldots, t_n)$ is an atomic formula.

Well-formed formulas are defined as usual, except that quantifiers must be applied to variables of the same sort. So $(\forall_i x^i)$ is permitted, but not $(\forall_i x^j)$ when i and j are different. Semantically, sorted formulas are interpreted by *many-sorted structures* as follows: As before, $\mathcal{S}$ is a function on the set of parameters, except this time it must assign to each parameter the correct sort of object.

1. $\mathcal{S}$ assigns to $\forall_i$ a nonempty set $|\mathcal{S}|_i$, called the *universe* of $\mathcal{S}$ of sort i. The universes of different sorts are required to be disjoint.

2. To each predicate symbol P of sort $< i_1, \ldots, i_n >$, $\mathcal{S}$ assigns a relation
$$P^{\mathcal{S}} \subseteq |\mathcal{S}|_{i_1} \times \cdots \times |\mathcal{S}|_{i_n}.$$
3. To each function symbol f of sort $< i_1, \ldots, i_n, i_{n+1} >$, $\mathcal{S}$ assigns a function
$$f^{\mathcal{S}} : |\mathcal{S}|_{i_1} \times \cdots \times |\mathcal{S}|_{i_n} \rightarrow |\mathcal{S}|_{i_{n+1}}.$$

The definitions of truth and satisfaction are as expected, under the intuition that $\forall_i$ means "for all members of the universe $|\mathcal{S}|_i$."

2.1.5 Reducing Many-Sorted Logic to Standard Logic

It turns out that many-sorted logics are no more powerful than ordinary unsorted first-order logic. To see why, consider an ordinary first-order language with all the predicate and function symbols of a given many-sorted language. In addition, it will have a new unary predicate symbol Q_i for each sort i. The intended meaning of $Q_i(t)$ is that term t is of sort i. Then we can transform every sentence σ of the many-sorted language into an ordinary first-order sentence σ^* in the following way:
Replace every subexpression $(\forall_i x^i)E(x^i)$ of σ by $(\forall x)[Q_i(x) \supset E(x)]$ where x is a variable chosen not to conflict with the others. Next define the following set Φ of ordinary first-order sentences:

1. $(\exists x)Q_i(x)$ for each sort i.
2. $(\forall x_1, \ldots, x_n).Q_{i_1}(x_1) \wedge \cdots \wedge Q_{i_n}(x_n) \supset Q_{i_{n+1}}(f(x_1, \ldots, x_n))$ for each function symbol f of sort $< i_1, \ldots, i_n, i_{n+1} >$.

Theorem In a many-sorted language, $\Sigma \models \sigma$ iff in its corresponding ordinary first-order language, $\Sigma^* \cup \Phi \models \sigma^*$.

Finally, a convention: When writing formulas in a sorted language, we will not use sorted quantifiers; their sorts will always be obvious from context.

2.1.6 Some Useful First-Order Inference Rules

For finding proofs by hand of the complex logical theorems required throughout this book, it is essential to use a collection of theorem-proving rules that reflect the natural thought processes that a mathematician invokes in proving theorems. Appendix A describes one such set of rules. They are not complete, but seem to work well surprisingly often. Of course, they are sound. If you have your own favorite theorem-proving methods, suitable for proving complicated theorems by hand, you can safely skip over this material.

2.1.7 A Limitation of First-Order Logic

Certain kinds of relations are not definable in first-order logic, for example, the *transitive closure* of a binary relation. We can think of a binary relation G as a directed graph

1. $s(x)$ is a member of the universe.
2. $s(X^n)$ is an n-ary relation on the universe, i.e. a set of n-tuples of elements of the universe.
3. $s(F^n)$ is an n-ary function, i.e. a function from n-tuples of elements of the universe into the universe.

Now extend s to $\overline{s}$ as follows: $\overline{s}(F(t_1, \ldots, t_n)) = s(F)(\overline{s}(t_1), \ldots, \overline{s}(t_n))$. Here, F is an n-ary function variable. Next, extend the definition of satisfaction of a formula by a structure $\mathcal{S}$:

- Satisfaction of atomic formulas must be extended: For an n-ary predicate variable X, $\models_{\mathcal{S}} X(t_1, \ldots, t_n)[s]$ iff $< \overline{s}(t_1), \ldots, \overline{s}(t_n) > \in s(X)$.
- Satisfaction must be extended to include the new quantifiers:
  1. $\models_{\mathcal{S}} (\forall X^n)\phi[s]$ iff for every n-ary relation $R \subseteq |\mathcal{S}|^n$, $\models_{\mathcal{S}} \phi[s(X^n|R)]$. Here, $s(X^n|R)$ is the function that is exactly like s except that for the predicate variable X^n it takes the value R.
     This is the formal way of capturing the intuition that $(\forall X^n)\phi$ means that no matter what extension the n-ary predicate variable X^n has in a structure, ϕ will be true in that structure.
  2. $\models_{\mathcal{S}} (\forall F^n)\phi[s]$ iff for every n-ary function $f : |\mathcal{S}|^n \rightarrow |\mathcal{S}|$, $\models_{\mathcal{S}} \phi[s(F^n|f)]$. Here, $s(F^n|f)$ is the function that is exactly like s except that for the function variable F^n it takes the value f.

The notions of model, validity, unsatisfiability and logical entailment for first-order sentences, as defined in Section 2.1.2, have their natural generalization to second-order languages.

2.2.3 Inductive Definitions and Second-Order Logic

In this book, we shall be appealing to second-order logic to characterize the situation calculus. We will not need to quantify over function variables to do this. The major use we will make of second-order logic will be to describe *smallest* sets with certain properties. This is the sort of thing one does in logic and computer science all the time. For example, the concept of an arithmetic expression over the natural numbers can be defined to be the intersection of all sets P such that:

1. Every natural number is in P.
2. If e_1 and e_2 are in A, then so also are $-(e_1)$, $+(e_1, e_2)$ and $*(e_1, e_2)$.

The following second-order sentence is a way to specify the set of all such arithmetic expressions:

$$
\begin{aligned}
&(\forall w).ae(w) \equiv \\
&\qquad (\forall P).\{(\forall x)[natnum(x) \supset P(x)] \wedge \\
&\qquad\qquad (\forall e)[P(e) \supset P(-(e))] \wedge \\
&\qquad\qquad (\forall e_1, e_2)[P(e_1) \wedge P(e_2) \supset P(+(e_1, e_2))] \wedge \\
&\qquad\qquad (\forall e_1, e_2)[P(e_1) \wedge P(e_2) \supset P(*(e_1, e_2))]\} \\
&\qquad\qquad \supset P(w).
\end{aligned}
$$

The informal concept of the intersection of all sets P with the above two properties is captured by universal second-order quantification over P.

Here is transitive graph closure again, which informally is defined as follows: The transitive closure, $closure(G)$, of G is the smallest set such that:

1. $(v, v') \in closure(G)$ whenever $(v, v') \in G$.
2. If $(v, v') \in G$ and $(v', v'') \in closure(G)$, then $(v, v'') \in closure(G)$.

Its second-order definition is:

$$
\begin{aligned}
&(\forall x, y).T(x, y) \equiv \\
&\qquad (\forall P).\{(\forall v, v')[G(v, v') \supset P(v, v')] \wedge \\
&\qquad\qquad (\forall v, v', v'')[G(v, v') \wedge P(v', v'') \supset P(v, v'')]\} \\
&\qquad\qquad \supset P(x, y).
\end{aligned}
$$

The definitions of the above two concepts—the arithmetic expressions, and the transitive closure of a graph—are examples of so-called *inductive definitions*. They normally consist of one or more *base cases*, together with one or more *inductive cases*. For the arithmetic expressions, the base case treats natural numbers, and there are three inductive cases, one for characterizing the negative expressions, one for addition, and one for multiplication.

One cannot freely formulate inductive definitions and expect always to obtain something meaningful. There are rules to this game; for an example, and one of the rules, see Exercise 5 below.

The second-order versions of such inductive definitions naturally lead to second-order *induction principles* for proving properties of the members of these sets. Induction will play a major role in our theory of actions. To see how induction principles emerge from such second-order inductive definitions, consider the above definition of the arithmetic expressions. The following sentence is a trivial consequence of the definition:

$$
\begin{aligned}
&(\forall P).\{(\forall x)[natnum(x) \supset P(x)] \wedge \\
&\qquad (\forall e)[P(e) \supset P(-(e))] \wedge \\
&\qquad (\forall e_1, e_2)[P(e_1) \wedge P(e_2) \supset P(+(e_1, e_2))] \wedge \\
&\qquad (\forall e_1, e_2)[P(e_1) \wedge P(e_2) \supset P(*(e_1, e_2))]\} \\
&\qquad\qquad \supset (\forall e).ae(e) \supset P(e).
\end{aligned}
$$

This informs us that in order to prove that every arithmetic expression has a certain property P, say for the purposes of proving the correctness of an algorithm for parsing such expressions, it is sufficient to show that:

1. All natural numbers have property P.
2. Whenever e has property P, so does $-(e)$.
3. Whenever e_1 and e_2 have property P, so do $+(e_1, e_2)$ and $*(e_1, e_2)$.

This is simply the principle of induction on the syntactic structure of arithmetic expressions.

In the same way, we can obtain the following induction principle for transitive closure:

$$(\forall P).\{(\forall v, v')[G(v, v') \supset P(v, v')] \wedge (\forall v, v', v'')[G(v, v') \wedge P(v', v'') \supset P(v, v'')]\} \supset (\forall x, y).T(x, y) \supset P(x, y).$$

2.2.4 The Incompleteness of Second-Order Logic

Second-order logic is incomplete, meaning that there is no recursive axiomatization and rules of inference that can recursively enumerate all and only the valid second-order sentences. This is a consequence of the Gödel incompleteness theorem for arithmetic. So why, in this book, do we appeal at all to second-order logic? The main reason is that we shall be interested in *semantically characterizing* actions and their properties. This will mean, however, that our theory of actions will be incomplete in the same way as number theory is incomplete. We shall therefore also be interested in finding important special cases of our theory of actions, and special kinds of computations, that have first-order axiomatizations.

2.3 Exercises

1. Give second-order definitions for the following sets. In each case, obtain a second-order induction principle from the definition.
   (a) The set of natural numbers is the smallest set containing 0 and that is closed under the successor function. In other words, the smallest set N such that:
      i. $0 \in N$.
      ii. If $x \in N$ then $s(x) \in N$.
   (b) The set of LISP S-expressions is the smallest set containing all atoms, and that is closed under the binary function *cons*. Assume that the unary predicate *atom* has already been defined.
   (c) The set of all of Adam and Eve's descendents, defined as the smallest set D such

that:

i. $Adam \in D$ and $Eve \in D$.

ii. If $x \in D$ and y is an offspring of x, then $y \in D$.

(d) The set of all propositional well-formed formulas, defined to be the smallest set P such that:

i. Every propositional atom is in P.

ii. If w_1 and $w_2 \in P$, then so also are $not(w_1)$ and $implies(w_1, w_2)$.

Assume that the unary predicate *atom* has already been defined.

2. Give second-order definitions for the following relations on LISP lists. Use the constant symbol Nil to denote the empty list, and the binary function symbol *cons*, where $cons(a, l)$ denotes the result of adding the element a to the front of list l.

(a) $evenLength(l)$, meaning that list l has an even number of elements.

(b) $sublist(l_1, l_2)$, meaning that list l_1 is the result of deleting 0 or more elements from list l_2.

(c) $append(l_1, l_2, l)$, meaning that list l is the result of appending list l_1 to l_2.

3. This exercise will lead you to a proof that transitive closure is not first-order definable. It relies on the Compactness Theorem, which informs us that any unsatisfiable set of first-order sentences has a finite unsatisfiable subset. The proof is by contradiction, so assume that $\tau(x, y)$ is a first-order formula with two free variables, defining the transitive closure of a binary relation G. By this, we mean that for any structure $\mathcal{S}$ assigning an extension to the binary predicate symbol G, and for any variable assignment s, $\models_{\mathcal{S}} \tau(x, y)[s]$ iff s assigns to the variables x and y a pair of domain elements of $\mathcal{S}$ that is in the transitive closure of G in that structure. All of which is a fancy way of saying that the set of pairs of domain elements of $\mathcal{S}$ that are in the transitive closure of G is precisely the set of domain element pairs that make τ true in the structure. Notice that the same formula τ must work for every graph G and every structure assigning an extension to G. Notice, finally, that $\tau(x, y)$, by hypothesis, says that, for some n, there is a sequence of n edges in the graph of G leading from x to y. Consider the following, infinitely many first-order formulas, each with two free variables x and y:

$$
\begin{array}{ll}
\alpha_1(x, y) & G(x, y), \\
\alpha_2(x, y) & (\exists z_1).G(x, z_1) \wedge G(z_1, y), \\
\ldots & \\
\alpha_n(x, y) & (\exists z_1, \ldots z_{n-1}).G(x, z_1) \wedge G(z_1, z_2) \wedge \cdots \wedge G(z_{n-1}, y), \\
\ldots &
\end{array}
$$

$\alpha_n(x, y)$ says that there is a sequence of n edges in the graph of G leading from vertex

x to vertex y. Now, assume that the underlying first-order language has at least two constant symbols A and B. (Later, we shall see that this assumption is not necessary; it does, however, considerably simplify the proof.) For $n \geq 1$, let β_n be the sentence:

$$\tau(A, B) \wedge \neg\alpha_1(A, B) \wedge \cdots \wedge \neg\alpha_n(A, B).$$

β_n says that (A, B) is in the transitive closure of G, and A and B are not connected by any sequence of edges of length n or less.

Consider the infinite set of sentences $\{\beta_1, \ldots, \beta_n, \ldots\}$. Notice what this set claims: that (A, B) is in the transitive closure of G, but no finite sequence of edges in the graph G leads from A to B. So this set is obviously unsatisfiable. Now, use the compactness theorem to obtain a contradiction.

The above proof assumed that the underlying first-order language had at least two constant symbols A and B. But this assumption is unimportant; it is easy to see that if $\tau(x, y)$ defines the transitive closure of G in a first order language, it continues to do so for the enlarged language obtained by adding two new constant symbols A and B to the original language. We then use exactly the same proof as before.

4. Statements about the size of the domain of discourse are often not first-order definable.

   (a) Using the compactness theorem—see the previous exercise—prove that there is no first-order sentence that is true in a structure iff that structure has a finite domain. As before, the proof is by contradiction. Suppose that ϕ is such a first-order sentence. For $n \geq 2$, let $\sigma_{\geq n}$ be the sentence

   $$(\exists x_1, \ldots, x_n).x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \cdots \wedge x_{n-1} \neq x_n.$$

   Here, there are $n(n-1)/2$ conjuncts, expressing that the x's are pairwise unequal. $\sigma_{\geq n}$ says that there are at least n distinct individuals in the domain of discourse. Next, consider the infinite set of sentences $\{\phi, \sigma_{\geq 2}, \sigma_{\geq 3}, \ldots, \}$. This is unsatisfiable. Now use compactness to obtain a contradiction.

   (b) Prove the stronger result that there is no set (finite or infinite) of first-order sentences all of which are true on a structure iff that structure has a finite domain.

5. **Weird Inductive Definitions.** The very idea of an inductive definition can be problematic. Consider the following "inductive definition" of a smallest set W of natural numbers such that:

   (a) $0 \in W$.

   (b) If $x \notin W$, then $x + 1 \in W$.

   Give some examples of sets W that can reasonably be said to be characterized by this definition. What set(s) do we get by replacing the words "a smallest set W" by the words "the intersection of all sets W"?

reached by performing the action sequence s, the robot r will be close to the object x.

3.1.2 Axiomatizing Actions in the Situation Calculus

The first observation one can make about actions is that they have *preconditions*: requirements that must be satisfied whenever they can be executed in the current situation. We introduce a predicate symbol $Poss$; $Poss(a, s)$ means that it is possible to perform the action a in that state of the world resulting from performing the sequence of actions s. Here are some examples:

- If it is possible for a robot r to pick up an object x in situation s, then the robot is not holding any object, it is next to x, and x is not heavy:

  $$Poss(pickup(r, x), s) \supset [(\forall z)\neg holding(r, z, s)] \wedge \neg heavy(x) \wedge nextTo(r, x, s).$$

- Whenever it is possible for a robot to repair an object, then the object must be broken, and there must be glue available:

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

The next feature of dynamic worlds that must be described are the causal laws—how actions affect the values of fluents. These are specified by so-called *effect axioms*. The following are some examples:

- The effect on the relational fluent $broken$ of a robot dropping a fragile object:

  $$fragile(x, s) \supset broken(x, do(drop(r, x), s)).$$

  This is the situation calculus way of saying that dropping a fragile object causes it to become broken; in the current situation s, if x is fragile, then in that successor situation $do(drop(r, x), s)$ resulting from performing the action $drop(r, x)$ in s, x will be broken.

- A robot repairing an object causes it not to be broken:

  $$\neg broken(x, do(repair(r, x), s)).$$

- Painting an object with colour c:

  $$colour(x, do(paint(x, c), s)) = c.$$

3.1.3 The Qualification Problem for Actions

With only the above axioms, nothing interesting can be proved about when an action is possible. For example, here are some preconditions for the action $pickup$:

$$Poss(pickup(r, x), s) \supset [(\forall z)\neg holding(r, z, s)] \wedge \neg heavy(x) \wedge nextTo(r, x, s).$$

The reason nothing interesting follows from this is clear; we can never infer when a $pickup$ is possible. We can try reversing the implication:

$$[(\forall z)\neg holding(r, z, s)] \wedge \neg heavy(x) \wedge nextTo(r, x, s) \supset Poss(pickup(r, x), s).$$

Now we can indeed infer when a pickup is possible, but unfortunately, this sentence is *false*! We also need, in the antecedent of the implication:

$$\neg gluedToFloor(x, s) \wedge \neg armsTied(r, s) \wedge \neg hitByTenTonTruck(r, s) \wedge \cdots$$

i.e, we need to specify all the *qualifications* that must be true in order for a $pickup$ to be possible! For the sake of argument, imagine succeeding in enumerating all the qualifications for $pickup$. Would that help? Suppose the only facts known to us about a particular robot R, object A, and situation S are:

$$[(\forall z)\neg holding(R, z, S)] \wedge \neg heavy(A) \wedge nextTo(R, A, S).$$

We still cannot infer $Poss(pickup(R, A), S)$ because we are not given that the above qualifications are true! Intuitively, here is what we want: When given only that the "important" qualifications are true:

$$[(\forall z)\neg holding(R, z, S)] \wedge \neg heavy(A) \wedge nextTo(R, A, S),$$

and if we *don't know* that any of the "minor" qualifications—$\neg gluedToFloor(A, S)$, $\neg hitByTenTonTruck(R, S)$—are true, infer $Poss(pickup(R, A), S)$. But if we happen to know that any one of the "minor" qualifications is false, this will block the inference of $Poss(pickup(R, A), S)$. Historically, this has been seen to be a problem peculiar to reasoning about actions, but this is not really the case. Consider the following fact about birds, which has nothing to do with reasoning about actions:

$$bird(x) \wedge \neg penguin(x) \wedge \neg ostrich(x) \wedge \neg pekingDuck(x) \wedge \cdots \supset flies(x).$$

But given only the fact $bird(Tweety)$, we want intuitively to infer $flies(Tweety)$. Formally, this is the same problem as action qualifications:

- The "important" qualification is $bird(x)$.
- The "minor" qualifications are: $\neg penguin(x), \neg ostrich(x), \cdots$

This is the classical example of the need for *nonmonotonic reasoning* in artificial intelligence. For the moment, it is sufficient to recognize that the qualification problem for actions is an instance of a much more general problem, and that there is no obvious way to address it. We shall adopt the following (admittedly idealized) approach: Assume that for each action $A(\vec{x})$, there is an axiom of the form

$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s),$$

where $\Pi_A(\vec{x}, s)$ is a first-order formula with free variables $\vec{x}, s$ that does not mention the function symbol do. We shall call these *action precondition axioms*. For example:

$$Poss(pickup(r, x), s) \equiv [(\forall z)\neg holding(r, z, s)] \wedge \neg heavy(x) \wedge nextTo(r, x, s).$$

In other words, we choose to ignore all the "minor" qualifications, in favour of necessary and sufficient conditions defining when an action can be performed. Appendix B further discusses the qualification problem along with its close relative, the ramification problem.

3.1.4 The Frame Problem

As if the qualification problem were not bad enough, there is another well known problem associated with axiomatizing dynamic worlds; axioms other than effect axioms are required. These are called *frame axioms*, and they specify the action *invariants* of the domain, i.e, those fluents unaffected by the performance of an action. For example, the following is a positive frame axiom, declaring that the action of robot r' painting object x' with colour c has no effect on robot r holding object x:

$$holding(r, x, s) \supset holding(r, x, do(paint(r', x', c), s)).$$

Here is a negative frame axiom for not breaking things:

$$\neg broken(x, s) \wedge [x \neq y \vee \neg fragile(x, s)] \supset \neg broken(x, do(drop(r, y), s)).$$

Notice that these frame axioms are truths about the world, and therefore must be included in any formal description of the dynamics of the world. The problem is that there will be a vast number of such axioms because only relatively few actions will affect the value of a given fluent. All other actions leave the fluent invariant, for example: An object's colour remains unchanged after picking something up, opening a door, turning on a light, electing a new prime minister of Canada, etc. Since, empirically in the real world, most actions have no effect on a given fluent, we can expect of the order of $2 \times \mathcal{A} \times \mathcal{F}$ frame axioms, where $\mathcal{A}$ is the number of actions, and $\mathcal{F}$ the number of fluents.

These observations lead to what is called the *frame problem*:

1. The axiomatizer must think of, and write down, all these quadratically many frame axioms. In a setting with 100 actions and 100 fluents, this involves roughly 20,000 frame axioms.
2. The implementation must somehow reason efficiently in the presence of so many axioms.

What Counts as a Solution to the Frame Problem?

Suppose the person responsible for axiomatizing an application domain has specified all the causal laws for that domain. More precisely, she has succeeded in writing down *all* the effect axioms, i.e. for each relational fluent F and each action A that causes F's truth value to change, axioms of the form:

$$R(\vec{x}, s) \supset (\neg)F(\vec{x}, do(A, s)),^{1}$$

and for each functional fluent f and each action A that can cause f's value to change, axioms of the form:

$$R(\vec{x}, y, s) \supset f(\vec{x}, do(A, s)) = y.$$

Here, R is a first-order formula specifying the contextual conditions under which the action A will have its specified effect on F and f. There are no restrictions on R, except that it must refer only to the current situation s. Later, we shall be more precise about the syntactic form of these effect axioms.

A solution to the frame problem is a systematic procedure for generating, from these effect axioms, all the frame axioms. If possible, we also want a *parsimonious* representation for these frame axioms (because in their simplest form, there are too many of them).

WHY SEEK A SOLUTION TO THE FRAME PROBLEM?

Frame axioms are necessary to reason about the domain being formalized; they cannot be ignored. Nevertheless, one could argue that there is no need to have a solution to the frame problem; instead, the onus should be on the axiomatizer to provide the frame axioms. Still, a solution to the frame problem would be very convenient by providing:

- *Modularity.* As new actions and/or fluents are added to the application domain, the axiomatizer need only add new effect axioms for these. The frame axioms will be automatically compiled from these (and the old frame axioms suitably modified to reflect these new effect axioms).
- *Accuracy.* There can be no accidental omission of frame axioms.

We shall also find that a systematic solution to the frame problem, in particular the one we shall describe shortly, will make possible a very rich theory of actions, accompanied by a natural implementation.

3.2 A Simple Solution to the Frame Problem (Sometimes)

This section describes a straightforward solution to the frame problem, one that can be very efficiently computed and that yields extremely compact axioms. It is not, however, completely general; hence the caveat "Sometimes" in the section title. In fact, the solution applies only to deterministic actions *without* ramifications (state constraints). Readers unfamiliar with the concept of state constraints can safely ignore this comment for the time being; we shall return to this topic in Section 4.3.2 and Appendix B.

---

1 The notation $(\neg)$ means that the formula following it may, or may not, be negated.

The solution that we shall describe is a synthesis of two proposals, one by Edwin Pednault, the other by Andrew Haas, as elaborated by Len Schubert, and proposed independently by Ernie Davis. To begin, we shall only consider relational fluents, i.e. relations whose truth values depend on the situation. Later, we shall consider functional fluents.

3.2.1 Frame Axioms: Pednault's Proposal

We illustrate this proposal with an example.

Example 3.2.1: Consider a simple electrical circuit consisting of various lightbulbs, each having its own on-off switch. When lightbulb x is on, flipping its switch, $flip(x)$, causes it to go off, and symmetrically when x is off. The fluent *on* has the following positive and negative effect axioms:

$$\neg on(x,s) \supset on(x, do(flip(x), s)),$$

$$on(x,s) \supset \neg on(x, do(flip(x), s)).$$

Now, rewrite these in the logically equivalent forms:

$$\neg on(x,s) \wedge y = x \supset on(x, do(flip(y), s)),$$

$$on(x,s) \wedge y = x \supset \neg on(x, do(flip(y), s)).$$

Next, suppose that these are *all* the causal laws relating the action $flip$ and the fluent *on*; we have described all the ways that flipping a switch can affect a light. Now, suppose that both $on(x,s)$ and $\neg on(x, do(flip(y), s))$ are true. In other words, light x was on in situation s, and in the situation resulting from flipping switch y, light x was off. Therefore, flipping the switch for y must have *caused* x to become off. Because we have axiomatized all the ways action $flip$ can affect *on*, the only way $\neg on(x, do(flip(y,s))$ could have become true is if the antecedent of its causal law, namely $on(x,s) \wedge y = x$ was true. Therefore, we conclude:

$$on(x,s) \wedge \neg on(x, do(flip(y), s)) \supset on(x,s) \wedge y = x.$$

This is logically equivalent to:

$$on(x,s) \wedge y \neq x \supset on(x, do(flip(y), s)).$$

This is exactly a positive frame axiom. It says that the action $flip(y)$ has no effect on the fluent $on(x,s)$ whenever y is different than x. We have obtained a frame axiom through purely syntactic manipulation of the effect axioms, by appealing to the assumption that these effect axioms capture all the causal laws relating the action $flip$ and the fluent *on*.

A symmetric argument yields the following negative frame axiom:

$$\neg on(x,s) \wedge y \neq x \supset \neg on(x, do(flip(y), s)).$$

as a frame axiom, rewrite it in the logically equivalent form:

$$holding(r, x, s) \wedge a \neq putDown(r, x) \wedge a \neq drop(r, x) \supset holding(r, x, do(a, s)). \tag{3.3}$$

This says that all actions other than $putDown(r, x)$ and $drop(r, x)$ leave $holding$ invariant,[2] which is the standard form of a frame axiom (actually, a set of frame axioms, one for each action distinct from $putDown$ and $drop$).

In general, an *explanation closure axiom* has one of the two forms:

$$F(\vec{x}, s) \wedge \neg F(\vec{x}, do(a, s)) \supset \alpha_F(\vec{x}, a, s),$$

$$\neg F(\vec{x}, s) \wedge F(\vec{x}, do(a, s)) \supset \beta_F(\vec{x}, a, s).$$

In these, the action variable a is universally quantified. These say that if ever the fluent F changes truth value, then α_F or β_F provides an exhaustive explanation for that change.

As before, to see how explanation closure axioms function like frame axioms, rewrite them in the logically equivalent form:

$$F(\vec{x}, s) \wedge \neg \alpha_F(\vec{x}, a, s) \supset F(\vec{x}, do(a, s)),$$

and

$$\neg F(\vec{x}, s) \wedge \neg \beta_F(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s)).$$

These have the same syntactic form as frame axioms with the important difference that action a is universally quantified. Whereas, in the worst case, we expect $2 \times \mathcal{A} \times \mathcal{F}$ frame axioms, there are just $2 \times \mathcal{F}$ explanation closure axioms. This parsimonious representation is achieved by quantifying over actions in the explanation closure axioms.

Schubert argues that explanation closure axioms are independent of the effect axioms, and it is the axiomatizer's responsibility to provide them. Like the effect axioms, these are domain-dependent. In particular, Schubert argues that they cannot be obtained from the effect axioms by any kind of systematic transformation. Thus, Schubert and Pednault entertain conflicting intuitions about the origins of frame axioms.

Like Pednault, Schubert's appeal to explanation closure as a substitute for frame axioms involves an assumption.

The Explanation Closure Assumption

α_F completely characterizes all those actions a that can cause the fluent F's truth value to change from true to false; similarly for β_F.

---

2 To accomplish this, we require unique names axioms like $pickup(r, x) \neq drop(r', x')$. We shall explicitly introduce these later.

We can see clearly the need for something like this assumption from the example explanation closure axiom (3.3). If, in the intended application, there were an action (say, $eat(r, x)$) that could lead to r no longer holding x, axiom (3.3) would be false.

Summary: The Davis/Haas/Schubert Proposal

- Explanation closure axioms provide a compact representation of frame axioms: $2 \times \mathcal{F}$ of them. (This assumes the explanation closure axioms do not become too long. Later we shall provide an argument why they are likely to be short.)
- But Schubert provides no systematic way of automatically generating them from the effect axioms. In fact, he argues this is impossible in general.

Can we combine the best of the Pednault and Davis/Haas/Schubert ideas? The next section shows how to do this.

3.2.3 A Simple Solution (Sometimes)

We illustrate the method with an example.

Example 3.2.4: Suppose there are two positive effect axioms for the fluent *broken*:

$$fragile(x, s) \supset broken(x, do(drop(r, x), s)),$$

$$nextTo(b, x, s) \supset broken(x, do(explode(b), s)),$$

i.e, exploding a bomb next to an object will break the object. These can be rewritten in the logically equivalent form:

$$[(\exists r)a = drop(r, x) \wedge fragile(x, s) \\ \vee (\exists b)\{a = explode(b) \wedge nextTo(b, x, s)\}] \\ \supset broken(x, do(a, s)). \tag{3.4}$$

Similarly, consider the negative effect axiom for *broken*:

$$\neg broken(x, do(repair(r, x), s)).$$

In exactly the same way, this can be rewritten as:

$$(\exists r)a = repair(r, x) \supset \neg broken(x, do(a, s)). \tag{3.5}$$

Now appeal to the following causal completeness assumption:

Axiom (3.4) characterizes all the circumstances that can cause x to become broken.

Next, suppose $\neg broken(x, s)$ and $broken(x, do(a, s))$ are both true. Then action a must have caused x to become *broken*, and by the completeness assumption, this can happen only because

$$(\exists r)a = drop(r, x) \wedge fragile(x, s) \vee (\exists b)\{a = explode(b) \wedge nextTo(b, x, s)\}$$

was true. This intuition can be formalized by the following explanation closure axiom:

$$\neg broken(x,s) \wedge broken(x,do(a,s)) \supset (\exists r)a = drop(r,x) \wedge fragile(x,s) \vee (\exists b)\{a = explode(b) \wedge nextTo(b,x,s)\}.$$

Similarly, (3.5) yields the following explanation closure axiom:

$$broken(x,s) \wedge \neg broken(x,do(a,s)) \supset (\exists r)a = repair(r,x).$$

3.2.4 Aside: Normal Forms for Effect Axioms

In the previous example, we rewrote one or more positive effect axioms as a single, logically equivalent positive effect axiom with the following syntactic normal form:

$$\gamma_F^+(\vec{x},a,s) \supset F(\vec{x},do(a,s)),$$

Similarly, we rewrote one or more negative effect axioms in the normal form:

$$\gamma_F^-(\vec{x},a,s) \supset \neg F(\vec{x},do(a,s)).$$

Here, $\gamma_F^+(\vec{x},a,s)$ and $\gamma_F^-(\vec{x},a,s)$ are first-order formulas with free variables among $\vec{x}, a, s$. The automatic generation of frame axioms appealed to these normal forms for the effect axioms. In this section, we precisely describe this transformation to normal form sentences. Readers who already understand this mechanism can skip to the next section.

Transformation of Effect Axioms to Normal Form: Each of the given positive effect axioms has the form:

$$\phi_F^+ \supset F(\vec{t},do(\alpha,s)).$$

Here α is an action term (e.g. pickup(x), put(A,y)) and the $\vec{t}$ are terms. We write these effect axioms without leading quantifiers, so the free variables (if any) in these axioms are implicitly universally quantified.
Write this in the following, logically equivalent form:

$$a = \alpha \wedge \vec{x} = \vec{t} \wedge \phi_F^+ \supset F(\vec{x},do(a,s)). \tag{3.6}$$

Here, $\vec{x} = \vec{t}$ abbreviates $x_1 = t_1 \wedge \cdots \wedge x_n = t_n$, and $\vec{x}$ are new variables, distinct from one another and distinct from any occurring in the original effect axiom. Now, suppose $y_1, \ldots, y_m$ are all the free variables, except for the *situation* variable s, occurring in the original effect axiom, i.e. all the variables (if any) that are implicitly universally quantified in this axiom. Then (3.6) is itself logically equivalent to:

$$(\exists y_1, \ldots, y_m)[a = \alpha \wedge \vec{x} = \vec{t} \wedge \phi_F^+] \supset F(\vec{x},do(a,s)).$$

So, each positive effect axiom for fluent F can be written in the logically equivalent form:

$$\Psi_F \supset F(\vec{x},do(a,s)),$$

where Ψ_F is a formula whose free variables are among $\vec{x}, a, s$. Do this for each of the k positive effect axioms for F, to get:

$$\Psi_F^{(1)} \supset F(\vec{x}, do(a, s)),$$
$$\vdots$$
$$\Psi_F^{(k)} \supset F(\vec{x}, do(a, s)).$$

Next, write these k sentences as the single, logically equivalent

$$[\Psi_F^{(1)} \vee \cdots \vee \Psi_F^{(k)}] \supset F(\vec{x}, do(a, s)).$$

This is the normal form for the positive effect axioms for fluent F.

Similarly, compute the normal form for the negative effect axioms for fluent F.

Readers familiar with the Clark completion semantics for logic programs will recognize the above transformation to normal form as very similar (but not identical) to the preliminary transformation of logic program clauses, in preparation for computing a program's *completion*.

Example 3.2.5: Suppose the following are all the positive effect axioms for fluent *tired*:

$$tired(Jack, do(walk(A, B), s)),$$
$$\neg marathonRunner(y) \wedge distance(u, v) > 2km \supset tired(y, do(run(u, v), s)).$$

Their normal form is:

$$\{[a = walk(A, B) \wedge x = Jack] \vee [(\exists u, v, y).a = run(u, v) \wedge \neg marathonRunner(y) \wedge distance(u, v) > 2km \wedge x = y]\}$$
$$\supset tired(x, do(a, s)).$$

By properties of equality and existential quantification, this simplifies to:

$$\{[a = walk(A, B) \wedge x = Jack] \vee [(\exists u, v).a = run(u, v) \wedge \neg marathonRunner(x) \wedge distance(u, v) > 2km]\}$$
$$\supset tired(x, do(a, s)).$$

3.2.5 A Simple Solution: The General Case

The example of Section 3.2.3 obviously generalizes. Suppose given, for each fluent F, the following two normal form effect axioms:

Positive Normal Form Effect Axiom for Fluent F

$$\gamma_F^+(\vec{x}, a, s) \supset F(\vec{x}, do(a, s)). \tag{3.7}$$

Negative Normal Form Effect Axiom for Fluent F

$$\gamma_F^-(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s)). \tag{3.8}$$

Here, $\gamma_F^+(\vec{x}, a, s)$ and $\gamma_F^-(\vec{x}, a, s)$ are first-order formulas with free variables among $\vec{x}, a, s$.

We make the following:

Causal Completeness Assumption:

Axioms (3.7) and (3.8), respectively, characterize all the conditions under which action a causes F to become true (respectively, false) in the successor situation.

In other words, these two sentences completely describe the causal laws for fluent F.

Hence, if F's truth value changes from *false* in the current situation s to *true* in the next situation $do(a, s)$ resulting from doing a, then $\gamma_F^+(\vec{x}, a, s)$ must have been *true*; similarly, if F's truth value changes from *true* to *false*. This informally stated assumption can be captured axiomatically by the following:

Explanation Closure Axioms

$$F(\vec{x}, s) \wedge \neg F(\vec{x}, do(a, s)) \supset \gamma_F^-(\vec{x}, a, s), \tag{3.9}$$

$$\neg F(\vec{x}, s) \wedge F(\vec{x}, do(a, s)) \supset \gamma_F^+(\vec{x}, a, s). \tag{3.10}$$

To make this work, we need:

Unique Names Axioms for Actions.

For distinct action names A and B,

$$A(\vec{x}) \neq B(\vec{y}).$$

Identical actions have identical arguments:

$$A(x_1, ..., x_n) = A(y_1, ..., y_n) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n.$$

Proposition 3.2.6: *Suppose that T is a first-order theory that entails*

$$\neg(\exists \vec{x}, a, s).\gamma_F^+(\vec{x}, a, s) \wedge \gamma_F^-(\vec{x}, a, s). \tag{3.11}$$

Then T entails that the general effect axioms (3.7) and (3.8), together with the explanation closure axioms (3.9) and (3.10), are logically equivalent to:

$$F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s). \tag{3.12}$$

Proof: Straightforward, but tedious. ∎

The requirement that $\neg(\exists \vec{x}, a, s).\gamma_F^+(\vec{x}, a, s) \wedge \gamma_F^-(\vec{x}, a, s)$ be entailed by the background theory T simply guarantees the consistency of the effect axioms (3.7) and (3.8). To see why, suppose this requirement were violated, so that for some $\vec{X}, A, S$ we have $\gamma_F^+(\vec{X}, A, S)$, and $\gamma_F^-(\vec{X}, A, S)$. Then we could simultaneously derive $F(\vec{X}, do(A, S))$

Continuing with the above blocks world example, and assuming that the above effect axioms describe all the ways an action can affect the height of a block (the causal completeness assumption for this example), we obtain the following successor state axiom for *height*:

$$height(b, do(a, s)) = y \equiv$$
$$\gamma_{height}(b, y, a, s) \vee y = height(b, s) \wedge \neg(\exists y')\gamma_{height}(b, y', a, s),$$

where $\gamma_{height}(b, y, a, s)$ is the above bracketed formula.

3.2.7 A Simple Solution: Summary

Our proposed solution to the frame problem appeals to the following axioms:

1. Successor state axioms:
   (a) For each relational fluent F:
   $$F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s).$$
   (b) For each functional fluent f:
   $$f(\vec{x}, do(a, s)) = y \equiv \gamma_f(\vec{x}, y, a, s) \vee f(\vec{x}, s) = y \wedge \neg(\exists y')\gamma_f(\vec{x}, y', a, s).$$
2. For each action A, a single action precondition axiom of the form:
   $$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s),$$
   where $\Pi_A(\vec{x}, s)$ is a first-order formula with free variables among $\vec{x}, s$.
3. Unique names axioms for actions.

Ignoring the unique names axioms (whose effects can be compiled), this axiomatization requires $\mathcal{F}+\mathcal{A}$ axioms in total, compared with the roughly $2\times\mathcal{A}\times\mathcal{F}$ explicit frame axioms that would otherwise be required. Here, $\mathcal{F}$ is the number of fluents and $\mathcal{A}$ the number of actions. There still remains the possibility that fewer axioms come at the expense of prohibitively long successor state axioms, but fortunately, this is unlikely.

- A successor state axiom's length is roughly proportional to the number of actions that affect the value of the fluent.
- The intuition leading to the frame problem is that most actions do not affect the fluent. So few actions affect it. So its successor state axiom is short.

The conciseness and perspicuity of this axiomatization relies on two things:

1. Quantification over actions.
2. The assumption that relatively few actions affect a given fluent.

3.2.8 Some Limitations of These Action Descriptions

Recall that the solution to the frame problem applies only when the effect axioms have the special syntactic form:

$$R(\vec{x}, s) \supset (\neg)F(\vec{x}, do(A, s)).$$

Such axioms describe *deterministic* actions, and preclude *indeterminate* actions with uncertain effects, for example:

$$heads(do(flip, s)) \vee tails(do(flip, s)),$$

$$(\exists x)holding(x, do(pickupAblock, s)).$$

Effect axioms, and therefore the solution to the frame problem, are for primitive actions only; as yet, there are no constructs for complex actions, like the following:

- Conditional actions:

  if $carInDriveway$ **then** $drive$ **else** $walk$ **endif**.

- Iterative actions:

  while $[(\exists\ block)ontable(block)]$ **do** $removeAblock$ **endwhile**.

- Nondeterministic actions:

  $$removeAblock = (\pi\ block)[pickup(block);\ putOnFloor(block)].$$

  Here, ; means sequence and $(\pi\ block)$ means nondeterministically pick a block, and for that choice of a block, do the complex action following this operator.

- Recursive actions: If $down$ means move an elevator down one floor, define $d(n)$, meaning move the elevator down n floors.

  proc $d(n)$
  if $n = 0$ **then** no_op
  else $down$; $d(n-1)$ **endif**
  endproc

In Chapter 6, we shall see how to define such complex actions within the situation calculus. Moreover, having done this, we will have a situation calculus-based programming language for tasks like discrete event simulation and high level robotic control. Probabilistic actions, like flipping a coin, will be treated in Chapter 12.

3.3 Deductive Planning with the Situation Calculus

Historically, the situation calculus has been most strongly identified with planning applications in artificial intelligence. This has basically been a textbook identification, since

(rightly or wrongly) very few people actually use the situation calculus to do real world planning. Despite this, there is considerable value in giving a logical account of the planning problem; at the very least, this provides a *specification* of the planning task, one that planning systems—logic-based or not—ought to respect. This section gives such a formal account of planning within the situation calculus.

The planning problem is this: Given an axiomatized initial situation, and a goal statement, find an action sequence that will lead to a state in which the goal will be true.

Example 3.3.1: We treat the simple setting of a one-handed robot that can hold at most one object in its hand, pick up and drop things, and walk about.

Action precondition axioms:

$$Poss(pickup(r, x), s) \equiv robot(r) \wedge [(\forall z)\neg holding(r, z, s)] \wedge nextTo(r, x, s),$$

$$Poss(walk(r, y), s) \equiv robot(r),$$

$$Poss(drop(r, x), s) \equiv robot(r) \wedge holding(r, x, s).$$

Effect axioms:

$$holding(r, x, do(pickup(r, x), s)),$$

$$\neg holding(r, x, do(drop(r, x), s)),$$

$$nextTo(r, y, do(walk(r, y), s)),$$

$$nextTo(r, y, s) \supset nextTo(x, y, do(drop(r, x), s)),$$

$$y \neq x \supset \neg nextTo(r, x, do(walk(r, y), s)),$$

$$onfloor(x, do(drop(r, x), s)),$$

$$\neg onfloor(x, do(pickup(r, x), s)).$$

The solution to the frame problem of the previous section yields the following:

Successor state axioms:

$$holding(r, x, do(a, s)) \equiv a = pickup(r, x) \vee holding(r, x, s) \wedge a \neq drop(r, x), \tag{3.16}$$

$$nextTo(x, y, do(a, s)) \equiv a = walk(x, y) \vee (\exists r)[nextTo(r, y, s) \wedge a = drop(r, x)] \vee nextTo(x, y, s) \wedge \neg(\exists z)[a = walk(x, z) \wedge z \neq y], \tag{3.17}$$

$$onfloor(x, do(a, s)) \equiv (\exists r)a = drop(r, x) \vee onfloor(x, s) \wedge \neg(\exists r)a = pickup(r, x). \tag{3.18}$$

Initial situation:

$$chair(C),\quad robot(R),\quad nextTo(R, A, S_0),\quad (\forall z)\neg holding(R, z, S_0). \tag{3.19}$$

The distinguished constant S_0 always denotes the initial situation in the situation calculus. So initially, a robot R is next to the object A; moreover, R is not holding anything.

As specified in our approach to the frame problem, we also need:

Unique names axioms for actions:

$$pickup(r, x) \neq drop(r', y),$$

$$pickup(r, x) \neq walk(r', y),$$

$$walk(r, y) = walk(r', y') \supset r = r' \wedge y = y',$$

etc.

Here are some facts derivable from these axioms:

From (3.16),

$$holding(R, A, do(pickup(R, A), S_0)). \tag{3.20}$$

From (3.20), (3.16) and unique names for actions,

$$holding(R, A, do(walk(R, y), do(pickup(R, A), S_0))). \tag{3.21}$$

From (3.17),

$$nextTo(R, y, do(walk(R, y), do(pickup(R, A), S_0))). \tag{3.22}$$

From (3.22) and (3.17),

$$nextTo(A, y, do(drop(R, A), do(walk(R, y), do(pickup(R, A), S_0)))). \tag{3.23}$$

From (3.18),

$$onfloor(A, do(drop(R, A), do(walk(R, y), do(pickup(R, A), S_0)))). \tag{3.24}$$

Suppose we want to derive:

$$(\exists s).nextTo(A, B, s) \wedge onfloor(A, s).$$

i.e., that there is an action sequence leading to a state of the world in which the object A is next to B and A is on the floor. The above is a constructive proof of this sentence, with

$$s = do(drop(R, A), do(walk(R, B), do(pickup(R, A), S_0))).$$

We can interpret this situation term as a *plan* to get A onto the floor next to B: First, R picks up A, then it walks to B, then it drops A. *The key idea here is that plans can be synthesized as a side-effect of theorem-proving.*

So the general picture of planning in the situation calculus, with respect to some background axioms, is to prove that some situation satisfies the goal statement G:

$$Axioms \vdash (\exists s)G(s).$$

Any variable-free binding for s obtained as a side-effect of a proof is a plan guaranteed to yield a situation satisfying the goal G. This is exactly the idea behind Prolog, in which programs are executed by a theorem-prover for the side-effect of obtaining bindings for the existentially quantified variables in the goal theorem.

There is one slight problem with this pretty picture of deductive planning that needs fixing. Notice that the proof of (3.23) and (3.24), from which the above plan was obtained, did not use any of the action precondition axioms. Therefore, the following two sentences could just as easily have been derived:

$$nextTo(A, y, do(drop(C, A), do(walk(C, y), do(pickup(C, A), S_0)))),$$

$$onfloor(A, do(drop(C, A), do(walk(C, y), do(pickup(C, A), S_0)))).$$

These also give a constructive proof of the sentence

$$(\exists s).nextTo(A, B, s) \wedge onfloor(A, s),$$

with

$$s = do(drop(C, A), do(walk(C, B), do(pickup(C, A), S_0))).$$

In other words, the chair C picks up A, then it walks to B, then it drops A. The obvious problem here is that the first plan, in which the robot R does the work, conforms to the action precondition axioms, while the second plan does not; according to these axioms, robots can pick things up, and go for walks, but chairs cannot. More precisely, the robot's plan is *executable* according to the action precondition axioms, meaning that one can prove, from the axioms, that:

$$Poss(pickup(R, A), S_0) \wedge Poss(walk(R, B), do(pickup(R, A), S_0)) \wedge$$
$$Poss(drop(R, A), do(walk(R, B), do(pickup(R, A), S_0))).$$

In other words, the action $pickup(R, A)$ is possible initially; $walk(R, B)$ is possible in the next situation resulting from doing the first action; finally, $drop(R, A)$ is possible in the situation resulting from doing the first two actions. On the other hand, there is no proof that this sequence of actions, as performed by the chair, is executable, and so the chair's actions should not be viewed as a plan. These considerations lead to the following,

Definition 3.3.2: Official Definition of a Plan for the Situation Calculus

Let $\mathcal{D}$ be a set of situation calculus axioms characterizing some application domain, σ a variable-free situation term, and $G(s)$ a situation calculus formula whose only free variable is the situation variable s. Then σ is a plan for G (relative to $\mathcal{D}$) iff

$$\mathcal{D} \models executable(\sigma) \wedge G(\sigma).$$

One way to determine such a σ is to prove the sentence $(\exists s).executable(s) \wedge G(s)$. Any

- A student may drop a course iff the student is currently enrolled in that course:

$$Poss(drop(st, c), s) \equiv enrolled(st, c, s).$$

Transaction Effect Axioms:

$$\neg enrolled(st, c, do(drop(st, c), s)),$$

$$enrolled(st, c, do(register(st, c), s)),$$

$$grade(st, c, g, do(change(st, c, g), s)),$$

$$g' \neq g \supset \neg grade(st, c, g', do(change(st, c, g), s)).$$

By solving the frame problem with respect to these effect axioms, we obtain the following successor state axioms for the database relations:

$$enrolled(st, c, do(a, s)) \equiv a = register(st, c) \vee enrolled(st, c, s) \wedge a \neq drop(st, c),$$

$$grade(st, c, g, do(a, s)) \equiv a = change(st, c, g) \vee grade(st, c, g, s) \wedge (\forall g')a \neq change(st, c, g').$$

3.4.4 Querying a Situation Calculus Database

Notice that on the above situation calculus perspective on databases, all updates are *virtual*; the database, consisting as it does of a set of logical sentences, is never physically changed. Axioms are forever. How then do we query a database after some sequence of transactions has been "executed"? For example, suppose we want to know whether John is enrolled in any courses after the transaction sequence $drop(John, C100), register(Mary, C100)$ has been "executed". The trick is to formulate this as a query with respect to the hypothetical future resulting from the performance of this transaction sequence:

$$Database \models (\exists c)enrolled(John, c, do(register(Mary, C100), do(drop(John, C100), S_0))).$$

This is an example of what is called the *projection problem* in AI planning. Such a sequence of update transactions is called a database *log* in database theory. In this setting, the AI projection problem becomes formally identical to the database problem of evaluating a query with respect to a database log. We shall have much more to say later about the projection problem, database logs and queries, and related issues.

3.5 Exercises

1. Suppose that the following are all of the effect axioms about an electrical system:

$$fuseOk(s) \wedge \neg lightOn(s) \supset lightOn(do(flipSwitch, s)),$$

$lightOn(s) \supset \neg lightOn(do(flipSwitch, s))$,

$\neg fuseOk(do(shortCircuit, s))$,

$lightOn(s) \supset \neg lightOn(do(shortCircuit, s))$.

(a) Give the normal form positive and negative effects axioms (Section 3.2.5) for the fluent $lightOn$.

(b) Obtain Pednault's solution to the frame problem for fluent $lightOn$.

(c) Give the explanation closure axioms (positive and negative) for the $lightOn$ fluent.

(d) Obtain the successor state axioms for the fluents $lightOn$ and $fuseOk$.

(e) Prove that the resulting action theory entails that

$$fuseOk(S_0) \supset (\exists s)lightOn(s).$$

2. The AI literature on change frequently appeals to one or more "classical" examples. Usually, despite their seeming simplicity, there is some historical reason for the example; it illustrates some desirable general property, or some special difficulty for existing (at the time) approaches. For each of the following examples, axiomatize it appropriately with action precondition and effect axioms, obtain successor state axioms, then solve the problem.

(a) **The monkey and bananas problem:** (The very first planning problem!)
A monkey is in a room containing a bunch of bananas hanging from the ceiling, and a chair. The monkey can't reach the bananas from the floor, but can if standing on the chair, provided the chair is underneath the bananas, which initially it is not. Neither, initially, is the monkey near the chair. In this scenario, four actions are possible:

- $walk(x)$ - The monkey walks to object x.
- $pushUnder(x, y)$ - The monkey pushes object x to a location under y.
- $climb(x)$ - The monkey climbs onto object x.
- $grab(x)$ - The monkey grabs object x.

Fluents:

- $onCeiling(x, s)$ - Object x is on the ceiling in situation s.
- $holding(x, s)$ - The monkey is holding x in situation s.
- $nextTo(x, s)$ - The monkey is next to x in situation s.
- $on(x, s)$, - The monkey is on x in situation s.
- $below(x, y, s)$ - Object x is below object y in situation s.

Deductively obtain a plan whereby the monkey gets the bananas. Make sure that the plan you obtain is executable.

(b) **The Yale shooting problem:**

Fluents:

- $alive(s)$ - Joe is alive in situation s.
- $loaded(s)$ - The gun is loaded in situation s.

Actions:

- *load* - Load the gun. This can always be done, whether or not the gun is currently loaded.
- *shoot* - Shoot the gun. This requires that the gun be loaded, and has the effect that Joe will not be alive.
- *wait* - A no-op; it has no effect on any fluent, and can always be performed.

Show that, regardless of the initial situation, Joe will not be alive after the sequence $load, wait, shoot$ takes place.

3. Axiomatize the game of tic-tac-toe on a 3×3 board, with two players, X and O who move alternately. When X moves, she chooses a blank square and marks an "X" in it. Player O marks unmarked squares with an "O". Use the following relational fluents:

   - $xsquare(squ, s)$: The square squ contains an "X" in situation s. Similarly, $osquare(squ, s)$ means squ contains an "O", and $bsquare(squ, s)$ means squ is blank. Assume the squares are numbered $1, \ldots, 9$.
   - $xturn(s)$: In situation s it is X's turn to make a move.
   - $oturn(s)$: In situation s it is O's turn to make a move.

   Use the following actions:

   - $xmove(squ)$: X places an X in square squ.
   - $omove(squ)$: O places an O in square squ.

   You will find it useful to introduce an abbreviation $wins(p, s)$, meaning that player p wins the game in situation s. $wins$ can be easily expressed in terms of the fluents $xsquare$ and $osquare$.

4. Let $favoriteBook(p, s)$ be a functional fluent, denoting person p's favorite book in situation s. Give effect axioms for the following facts about this fluent:
   After John reads what is, in the current situation, Mary's favorite book, that book will become his favorite book in the next situation.
   After Mary reads Moby Dick, her favorite book will be Moby Dick.

   Using these, determine the successor state axiom for $favoriteBook$. Then prove that

in the situation resulting from Mary reading Moby Dick, followed by Mary reading Middlemarch, followed by John reading what is, in the current situation, Mary's favorite book, John's favorite book will be Moby Dick.

5. Formalize the following toy airline reservation system along the lines of the example database system of this chapter. This will involve formulating suitable intuitively plausible effect and action precondition axioms, and deriving the corresponding successor state axioms.

Fluents:

- $seatsAvailable(flight\#, date, n, s)$ - There are n seats available in the situation s for $flight\#$ on $date$.
- $hasReservation(person, flight\#, date, s)$ - *person* has a reservation in the situation s for $flight\#$ on $date$.
- $seatReserved(person, seat\#, flight\#, date, s)$ - Obvious meaning.

Transactions:

- $reserve(person, flight\#, date)$ - Reserve space for *person* on *flight#* for *date*.
- $assignSeat(person, seat\#, flight\#, date)$ - Obvious meaning.
- $cancelReservation(person, flight\#, date)$ - Cancel the booking.

3.6 Bibliographic Remarks

The situation calculus has been a part of the artificial intelligence zeitgeist almost from the very beginning of the field. It is included in the standard material of every introductory course on AI, and it is the language of choice for investigations of various technical problems that arise in theorizing about actions and their effects. Nevertheless, historically the AI community has not taken the situation calculus seriously as a foundation for practical work in planning, control, simulation or robotics. There were good representational and computational reasons for this. Representationally, there were no suitably rich accounts for the most basic ingredients of a theory of actions, like time, concurrency, continuous processes, or procedures. Neither was there a good story for the frame and qualification problems. Computationally, it was seen as a first-order language requiring necessarily inefficient theorem-proving methods. One purpose of this book is to show that these historical limitations of the situation calculus have been largely overcome by recent research, and that the resulting enriched language has many representational and computational advantages for modeling dynamical systems.

The basic conceptual and formal ingredients of the situation calculus are due to John

McCarthy [136] in 1963. The frame problem was first observed by John McCarthy and Pat Hayes [141]; since then, it has been the subject of a large body of technical research (e.g. Brown [28]), as well as philosophical speculation (e.g. Pylyshyn [170]). Two excellent recent books on theories of actions, with specific focus on solutions to the frame and related problems are by Sandewall [187] and by Shanahan [195]. See also the book by Shoham [199] for an earlier treatment of causality and the frame problem. The companion to the frame problem—the qualification problem—was also first observed by McCarthy [137], although he had in mind a somewhat more general notion of the scope of this problem than that prevailing today, which concerns only the qualifications relevant to action preconditions.

The first uses of the situation calculus were in planning, following the influential proposal of Green [71]. Indeed, Shakey, the very first autonomous robot project, was based on Green's account of planning, using resolution theorem-proving with a situation calculus axiomatization of the robot's actions and environment. This was in the late 1960's and early 1970's, before very much was known about the frame problem or theorem-proving, and this first attempt at a situation calculus-based approach to high level robotics was abandoned in the face of the extreme computational inefficiencies that were encountered (Fikes and Nilsson [50]).

The solution to the frame problem of Section 3.2 was described in Reiter [173]; it combines elements of earlier proposals by Pednault [154], Davis [36], and by Haas [76], as elaborated by Schubert [192]. Independently, Elkan [41] proposed a similar solution. The significance of the consistency condition (3.11) in deriving successor state axioms (Proposition 3.2.6) was first noted by Pednault [154], and generalized slightly by Reiter [173] to fit his solution to the frame problem.

The situation calculus-based approach to formalizing databases evolving under update transactions was described by Reiter in [175]. This approach to database updates is extended by Bertossi, Arenas and Ferretti [18], who also provide an implementation that interfaces to a relational database system, and to automated theorem-provers for proving properties of the situation calculus database description. For more information on relational databases, see Maier [131], and for a relational database perspective on update transactions, see Abiteboul [1]. Our perspective on databases, in which the initial database is any first-order theory, not necessarily relational, is closest to that held by the deductive database community; see, for example, Minker [147].

The monkey-bananas problem—the very first planning problem—was proposed by John McCarthy in 1963 and reprinted in [136]; in that paper, McCarthy also gave an axiomatization from which he showed that the monkey can indeed get the bananas.

The Yale Shooting Problem was proposed by Hanks and McDermott in 1986 [80], not because it is very difficult—it is trivial under our approach to the frame problem—but be-

These are used to denote situation-independent functions like

$$sqrt(x), \quad height(MtEverest), \quad agent(pickup(person, object)).$$

- For each $n \geq 0$, a finite or countably infinite number of function symbols of sort

  $$(action \cup object)^n \rightarrow action.$$

  These are called *action functions*, and are used to denote actions like $pickup(x)$, $move(A, B)$, etc. In most applications, there will be just finitely many action functions, but we allow the possibility of an infinite number of them.

  Notice that we distinguish between function symbols taking values of sort *object* and those—the action functions—taking values of sort *action*. In what follows, the latter will be distinguished by the requirement that they be axiomatized in a particular way by what we shall call *action precondition axioms*.

- For each $n \geq 0$, a finite or countably infinite number of predicate symbols with arity $n+1$, and sorts $(action \cup object)^n \times situation$. These predicate symbols are called *relational fluents*. In most applications, there will be just finitely many relational fluents, but we do not preclude the possibility of an infinite number of them. These are used to denote situation dependent relations like $ontable(x, s)$, $husband(Mary, John, s)$, etc. Notice that relational fluents take just one argument of sort *situation*, and this is always its last argument.

- For each $n \geq 0$, a finite or countably infinite number of function symbols of sort

  $$(action \cup object)^n \times situation \rightarrow action \cup object.$$

  These function symbols are called *functional fluents*. In most applications, there will be just finitely many functional fluents, but we do not preclude the possibility of an infinite number of them. These are used to denote situation dependent functions like $age(Mary, s)$, $primeMinister(Italy, s)$, etc. Notice that functional fluents take just one argument of sort *situation*, and this is always its last argument.

Notice that only two function symbols of $\mathcal{L}_{sitcalc}$—S_0 and do—are permitted to take values in sort *situation*.

4.2 Axioms for the Situation Calculus

There is a strong analogy between the situation calculus and a part of number theory. Accordingly, we begin with a diversion into the world of natural numbers.

4.2.1 Number Theory

In 1889, Giuseppe Peano provided the first axiomatization of the natural numbers. Here, we focus on a fragment of full number theory that characterizes the successor function, and

the less-than relation. Full number theory would also have axioms characterizing addition and multiplication, but these turn out not to be relevant here. Accordingly, we introduce the following vocabulary for a second-order language (with equality):

- A single constant symbol 0.
- A unary function symbol σ (successor function).
- A binary relation symbol $<$ (the less than relation).

The axioms for this part of number theory are:

$$\sigma(x) = \sigma(y) \supset x = y,$$

$$(\forall P).P(0) \wedge (\forall x)[P(x) \supset P(\sigma(x))] \supset (\forall x)P(x),$$

$$\neg x < 0,$$

$$x < \sigma(y) \equiv x \leq y.$$

Here, $x \leq y$ is an abbreviation for $x < y \vee x = y$.

The second sentence is a second-order induction axiom; it is a way of characterizing the domain of discourse to be the *smallest* set such that

1. 0 is in the set.
2. Whenever x is in the set, so is $\sigma(x)$.

This second-order fragment of arithmetic is *categorical* (it has a unique model). Normally, textbooks on the subject describe first-order number theory, which you obtain by replacing the second-order axiom by an induction *schema* representing countably infinitely many first-order sentences, one for each instance of P obtained by replacing P by a first-order formula with one free variable. The resulting first-order fragment of arithmetic is *not* categorical; it has (infinitely) many distinct models.

This leads to the natural question: Why not use the second-order axiom instead of the axiom schema? The answer is, because second-order logic is incomplete; there is no "decent" axiomatization of second-order logic that will yield all the valid second-order sentences; the valid sentences of second-order logic are not recursively enumerable, or equivalently, there is no recursive axiomatization for second-order logic (Section 2.2.4). That being the case, why appeal to second-order logic at all? The main reason is that *semantically*, but not syntactically, it characterizes the natural numbers. We shall encounter the same phenomenon in semantically characterizing the situation calculus.

4.2.2 Foundational Axioms for Situations

We now focus on the domain of situations. The primary intuition about situations that we wish to capture axiomatically is that they are finite sequences of actions. We want also to be able to say that a certain sequence of actions precedes another. The four axioms we are

about to present capture these two properties of situations:

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \tag{4.1}$$

$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s). \tag{4.2}$$

Compare these to the first two axioms for the natural numbers.

Axiom (4.2) is a second-order induction axiom, and has the effect of limiting the sort *situation* to the smallest set containing S_0, and closed under the application of the function *do* to an action and a situation. Any model of these axioms will have as its domain of situations the smallest set $\mathcal{S}$ satisfying:

1. $\sigma_0 \in \mathcal{S}$, where σ_0 is the interpretation of S_0 in the model.
2. If $\sigma \in \mathcal{S}$, and $A \in \mathcal{A}$, then $do(A, \sigma) \in \mathcal{S}$, where $\mathcal{A}$ is the domain of actions in the model.

Notice that axiom (4.1) is a unique names axiom for situations. This, together with the induction axiom, imply that two situations will be the same iff they result from the same sequence of actions applied to the initial situation. Two situations S_1 and S_2 may be different, yet assign the same truth values to all fluents. So a situation in the situation calculus must not be identified with the set of fluents that hold in that situation, i.e with a *state*. The proper way to understand a situation is as a *history*, namely, a finite sequence of actions; two situations are equal iff they denote identical histories. This is the major reason for using the terminology "situation" instead of "state"; the latter carries with it the connotation of a "snapshot" of the world. In our formulation of the situation calculus, *situations are not snapshots, they are finite sequences of actions.* While states can repeat themselves—the same snapshot of the world can happen twice—situations cannot.

There are two more axioms, designed to capture the concept of a subhistory:

$$\neg s \sqsubset S_0, \tag{4.3}$$

$$s \sqsubset do(a, s') \equiv s \sqsubseteq s', \tag{4.4}$$

where $s \sqsubseteq s'$ is an abbreviation for $s \sqsubset s' \vee s = s'$.[1] Here, the relation $\sqsubset$ provides an ordering on situations; $s \sqsubset s'$ means that the action sequence s' can be obtained from the sequence s by adding one or more actions to the front of s.[2] These axioms also have their

---

1 Unlike $\sqsubset$, $\sqsubseteq$ is *not* a predicate symbol of $\mathcal{L}_{sitcalc}$; it is an *external* notation—a convenient shorthand—standing for the situation calculus formula that it abbreviates. So whenever you encounter an external expression of the form $s \sqsubseteq s'$, you are to mentally replace it with the legitimate situation calculus formula $s \sqsubset s' \vee s = s'$ for which it is an abbreviation. A good way to view abbreviations in logic is as macros that expand, wherever they are used, into their definitions. Logicians often appeal to such macros as a way of keeping a language, and an axiomatization, to a bare minimum. That is what we are doing here by treating $\sqsubseteq$ as an abbreviation. First, it need not be included in $\mathcal{L}_{sitcalc}$; secondly, an axiom $s \sqsubseteq s' \equiv s \sqsubset s' \vee s = s'$ need not be included among the foundational axioms of the situation calculus.

2 Readers familiar with the programming language LISP will have noticed that in the situation calculus, the constant S_0 is just like NIL, and *do* acts like *cons*. Situations are simply *lists* of primitive actions. For example, the situation term $do(C, do(B, do(A, S_0)))$ is simply an alternative syntax for the LISP list $(C\ B\ A)$ (=

analogues in the last two axioms of the preceding fragment of number theory.

The above four axioms are *domain independent*. They will provide the basic properties of situations in any domain specific axiomatization of particular fluents and actions. Henceforth, call them Σ. It takes a little bit of proving, but one can show that the situations in any model of Σ can be represented by a tree. Figure 4.1 shows one such tree, for a model M of Σ with n individuals $\alpha_1, \ldots, \alpha_n$ in M's domain of actions.

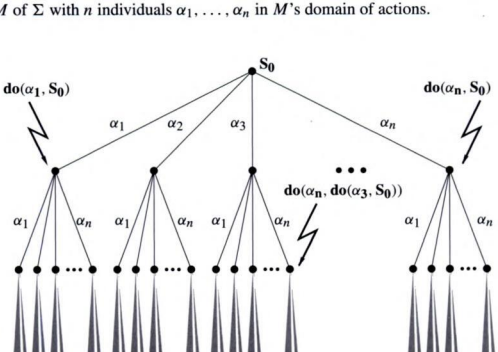


Figure 4.1: The tree of situations for a model with n actions.

We have abused notation slightly in this figure; strictly speaking, do should be do^M, and S_0 should be S_0^M. Nothing in this figure should suggest to the reader that there can be only finitely many actions in a model of Σ. There are models of Σ with action domain of any

$cons(C, cons(B, cons(A, NIL)))$). Notice that to obtain the action *history* corresponding to this term, namely the performance of action A, followed by B, followed by C, we read this list from right to left. Therefore, when situation terms are read from right to left, the relation $s \sqsubset s'$ means that situation s is a proper subhistory of the situation s'. The situation calculus induction axiom (4.2) is simply the induction principle for lists: If the empty list has property P and if, whenever list s has property P so does $cons(a, s)$, then all lists have property P.

cardinality.

4.2.3 Some Consequences of the Foundational Axioms

The following are some logical consequences of the foundational axioms Σ:

$S_0 \neq do(a, s)$,

$do(a, s) \neq s$,

Existence of a predecessor: $s = S_0 \vee (\exists a, s')s = do(a, s')$,

$S_0 \sqsubseteq s$,

Transitivity: $s_1 \sqsubset s_2 \wedge s_2 \sqsubset s_3 \supset s_1 \sqsubset s_3$,

Anti-reflexivity: $\neg s \sqsubset s$,

Unique names: $s_1 \sqsubset s_2 \supset s_1 \neq s_2$,

Anti-symmetry: $s \sqsubset s' \supset \neg s' \sqsubset s$,

$\neg do(a, s) \sqsubseteq s$,

$s \sqsubseteq s' \wedge s' \sqsubseteq s \supset s = s'$.

The Principle of Double Induction

$$
\begin{aligned}
&(\forall R).R(S_0, S_0) \ \wedge \\
&\quad [(\forall a, s).R(s, s) \supset R(do(a, s), do(a, s))] \ \wedge \\
&\quad [(\forall a, s, s').s \sqsubseteq s' \wedge R(s, s') \supset R(s, do(a, s'))] \\
&\qquad \supset (\forall s, s').s \sqsubseteq s' \supset R(s, s').
\end{aligned}
$$

4.2.4 Executable Situations

A situation is a finite sequence of actions. There are no constraints on the actions entering into such a sequence, so that it may not be possible to actually execute these actions one after the other. For example, suppose the precondition for performing the action $putdown(x)$ in situation s is that the agent is holding x: $holding(x, s)$. Suppose also that in situation S_0, the agent is not holding A: $\neg holding(A, S_0)$. Then $do(putdown(A), S_0)$ is a perfectly good situation, but it is not executable; the precondition for performing $putdown(A)$ is violated in S_0. Moreover, $do(pickup(B), do(putdown(A), S_0))$ is not an executable situation either. In fact, no situation whose first action is $putdown(A)$ is executable. Similarly, an action A_1 may be executable in S_0, but the action A_2 may not be possible in $do(A_1, S_0)$, in which case no sequence of actions beginning with these two would be executable. We emphasize that action sequences in which an action violates its precondition are perfectly good situations. They simply are not physically realizable; they

- Grades are functional: No matter how the database evolves, no one may have two different grades for the same course in the same database situation.
- No one's salary may decrease during the evolution of the database.

The concept of an integrity constraint is intimately connected with that of database *evolution*. No matter how the database evolves, the constraint must be true in all database futures. Therefore, in order to make formal sense of integrity constraints, we need a prior theory of database evolution. How do databases change? One way (not the only way) is via predefined update *transactions*, as formally specified using the situation calculus in the last chapter. We shall assume that transactions provide the only mechanism for such database changes. Therefore, we can appeal to our earlier situation calculus theory of database transactions (Section 3.4) in defining integrity constraints and their role in maintaining database integrity. We shall identify a possible future of a database with a situation, and represent integrity constraints as first-order sentences, universally quantified over situations.

- No one ever has two different grades for the same course in any database situation:

$$(\forall s, st, c, g, g').grade(st, c, g, s) \wedge grade(st, c, g', s) \supset g = g'. \tag{4.6}$$

- Salaries never decrease in any executable situation:

$$\begin{aligned}(\forall s, s', p, \$, \$').executable(s') \wedge s \sqsubseteq s' \wedge sal(p, \$, s) \wedge sal(p, \$', s') \\ \supset \$ \leq \$'.\end{aligned} \tag{4.7}$$

We can now define what we mean by a database satisfying its constraints.

Definition 4.3.1: Constraint Satisfaction
A database *satisfies* an integrity constraint IC iff

$$Database \models IC.$$

So, for example, to establish that a database satisfies the integrity constraint (4.6), we must prove that the database entails it. This sentence has the typical syntactic form calling for an inductive proof, using the induction axiom (4.2), with induction hypothesis $P(s)$ as:

$$(\forall st, c, g, g').grade(st, c, g, s) \wedge grade(st, c, g', s) \supset g = g'.$$

4.3.1 Some Examples of Inductive Proofs

Example 4.3.2: Consider an electric circuit with two switches, Sw_1 and Sw_2, and an action $toggle(sw)$ that opens the switch sw if it is closed, and closes it if it is open. The circuit is connected to a light that changes its state from on to off and off to on whenever one of the switches is toggled.

$$open(sw, do(a, s)) \equiv \neg open(sw, s) \wedge a = toggle(sw) \vee$$

$$open(sw, s) \wedge a \neq toggle(sw).$$

$$light(do(a, s)) \equiv \neg light(s) \wedge [a = toggle(Sw_1) \vee a = toggle(Sw_2)] \vee light(s) \wedge a \neq toggle(Sw_1) \wedge a \neq toggle(Sw_2).$$

The property we wish to prove of this setting is:

$$(\forall s).light(s) \equiv [open(Sw_1, s) \equiv open(Sw_2, s)]. \quad (4.8)$$

In other words, it will always be the case that the light will be on iff both switches are open or closed (= not open) together. This sentence has the right syntactic form for an application of the induction principle (4.2) for arbitrary situations. So, take $P(s)$ to be:

$$light(s) \equiv [open(Sw_1, s) \equiv open(Sw_2, s)].$$

Assume further that (4.8) is true of the initial situation:

$$light(S_0) \equiv [open(Sw_1, S_0) \equiv open(Sw_2, S_0)].$$

The proof is long and tedious, but is otherwise straightforward. It requires the fact that $Sw_1 \neq Sw_2$, as well as a unique names axioms for the action $toggle(sw)$.

Example 4.3.3: We want to prove that no person can carry himself in any executable situation:

$$(\forall s).executable(s) \supset (\forall x)\neg carrying(x, x, s). \quad (4.9)$$

Assume the following action precondition axiom for *pickup*, and successor state axiom for *carrying*:

$$Poss(pickup(x, y), s) \equiv x \neq y \wedge \neg carrying(x, y, s).$$

$$carrying(x, y, do(a, s)) \equiv a = pickup(x, y) \vee carrying(x, y, s) \wedge a \neq putdown(x, y).$$

Assume further that initially, no one is carrying himself:

$$\neg carrying(x, x, S_0).$$

The sentence (4.9) has the right syntactic form for a proof by induction on executable situations (Section 4.2.5), with $P(s)$ as $\neg carrying(x, x, s)$. The proof is straightforward. Notice that there are *non-executable* situations in which someone *is* carrying himself. For example, it is trivial to derive $carrying(Sam, Sam, do(pickup(Sam, Sam), S_0))$. The situation $do(pickup(Sam, Sam), S_0)$ is simply not physically realizable; it is a ghost situation. So the restriction in (4.9) to executable situations is essential. This was not the case for the previous example.

Example 4.3.4: We want to prove that salaries never decrease over any executable situa-

tion:

$$(\forall s, s', p, \$, \$').executable(s') \wedge s \sqsubseteq s' \wedge sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$

Assume the following background axioms:

- To change a person's salary, the new salary must be greater than the old:

$$Poss(changeSal(p, \$), s) \equiv (\exists \$').sal(p, \$', s) \wedge \$' < \$.$$

- Successor state axiom for *sal*:

$$sal(p, \$, do(a, s)) \equiv a = changeSal(p, \$) \vee sal(p, \$, s) \wedge (\forall \$')a \neq changeSal(p, \$').$$

- Initially, the relation *sal* is functional in its second argument:

$$sal(p, \$, S_0) \wedge sal(p, \$', S_0) \supset \$ = \$'.$$

- *Unique names axiom* for *changeSal*:

$$changeSal(p, \$) = changeSal(p', \$') \supset p = p' \wedge \$ = \$'.$$

The sentence to be proved is logically equivalent to:

$$(\forall s, s').executable(s') \wedge s \sqsubseteq s' \supset (\forall p, \$, \$').sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$

This has the right syntactic form for an application of the double induction principle for executable situations of Section 4.2.5, with $R(s, s')$:

$$(\forall p, \$, \$').sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$

The rest of the proof is straightforward. As was the case in the previous example, the restriction to executable situations in the current example is essential; it is trivial to derive $sal(Mary, 100, do(changeSal(Mary, 100, S_0))$, even when initially, $Mary$'s salary is 200: $sal(Mary, 200, S_0)$.

4.3.2 State Constraints

The previous discussion of inductive proofs in the situation calculus dealt with two qualitatively different classes of sentences whose proofs required induction: Sentences universally quantified over *all* situations, executable or not, and sentences universally quantified over executable situations only. Sentences (4.6) and (4.8) are examples of the first kind, while (4.7) and (4.9) are of the second. Sentences of the first kind use the simple induction axiom (4.2) of the foundational axioms for the situation calculus, or the principle of double induction over all situations derived in Section 4.2.3, and *do not make use of action precondition axioms*. The second class of sentences require induction (simple or double) over executable situations, as derived in Section 4.2.5, and typically do require the information

in action precondition axioms to make the proofs by induction go through. Both classes of sentences describe *global* properties of the background theory of actions, global because these are properties that must be true of *all* (executable) situations. Our perspective on such sentences is that they are *inductive invariants* of the background axioms, i.e. they must be provable, normally using induction, from these axioms.

Both kinds of sentences are examples of what are called *state constraints* in the literature on theories of actions; database researchers generally call them *integrity constraints*. By whatever name, they are a source of deep theoretical and practical difficulties in modeling dynamical systems, and we do not treat this topic in any depth in this book. Appendix B contains a brief discussion that clarifies how sentences of the first and second kinds are intimately connected to the frame and qualification problems (Section 3.1.3), respectively.

4.4 Basic Theories of Action

Recall that Σ denotes the four foundational axioms for situations. We now consider some metamathematical properties of these axioms when combined with a specification of the initial situation, successor state and action precondition axioms, and unique names axioms for actions. Such a collection of axioms will be called a *basic theory of actions*. First we must be more precise about what counts as successor state and action precondition axioms.

Definition 4.4.1: The Uniform Formulas
Let σ be a term of sort *situation*. Inductively define the concept of a term of $\mathcal{L}_{sitcalc}$ that is *uniform in* σ as follows:

1. Any term that does not mention a term of sort *situation* is uniform in σ.
2. If g is an n-ary non-fluent function symbol, and $t_1, \ldots, t_n$ are terms that are uniform in σ and whose sorts are appropriate for g, then $g(t_1, \ldots, t_n)$ is uniform in σ.
3. If f is an $(n+1)$-ary functional fluent symbol, and $t_1, \ldots, t_n$ are terms that are uniform in σ and whose sorts are appropriate for f, then $f(t_1, \ldots, t_n, \sigma)$ is uniform in σ.

The formulas of $\mathcal{L}_{sitcalc}$ that are *uniform in* σ are inductively defined by:

1. Any formula that does not mention a term of sort *situation* is uniform in σ.
2. When F is an $(n+1)$-ary relational fluent and $t_1, \ldots, t_n$ are terms uniform in σ whose sorts are appropriate for F, then $F(t_1, \ldots, t_n, \sigma)$ is a formula uniform in σ.
3. If U_1 and U_2 are formulas uniform in σ, so are $\neg U_1$, $U_1 \wedge U_2$ and $(\exists v)U_1$ provided v is a variable not of sort *situation*.

Thus, a formula of $\mathcal{L}_{sitcalc}$ is uniform in σ iff it does not mention the predicates $Poss$ or $\sqsubset$, it does not quantify over variables of sort *situation*, it does not mention equality

on situations, and whenever it mentions a term of sort *situation* in the situation argument position of a fluent, then that term is σ.

Example 4.4.2: When f, g are functional fluents, F is a relational fluent, and P is a non-fluent predicate, the following is uniform in σ:

$$(\forall x).x = f(g(x,\sigma),\sigma) \wedge (\exists y)F(g(A,\sigma),y,\sigma) \supset \\ \neg P(x,B) \vee P(f(f(x,\sigma),\sigma),g(x,\sigma)).$$

No formula that mentions $Poss$ or $\sqsubset$ is uniform in any situation term σ. The following are not uniform in σ:

$$holding(x, do(pickup(x),\sigma)), \quad do(a,\sigma) \neq \sigma, \quad (\exists s)holding(x,s),$$
$$resigned(primeMinister(Canada, do(elect(p,\sigma)),\sigma)).$$

Definition 4.4.3: Action Precondition Axiom
An action precondition axiom of $\mathcal{L}_{sitcalc}$ is a sentence of the form:

$$Poss(A(x_1,\cdots,x_n),s) \equiv \Pi_A(x_1,\cdots,x_n,s),$$

where A is an n-ary action function symbol, and $\Pi_A(x_1,\cdots,x_n,s)$ is a formula that is uniform in s and whose free variables are among $x_1,\cdots,x_n,s$.

The uniformity requirement on Π_A ensures that the preconditions for the executability of the action A are determined by the current situation s.

Definition 4.4.4: Successor State Axiom

1. A successor state axiom for an $(n+1)$-ary relational fluent F is a sentence of $\mathcal{L}_{sitcalc}$ of the form:

   $$F(x_1,\ldots,x_n,do(a,s)) \equiv \Phi_F(x_1,\ldots,x_n,a,s), \tag{4.10}$$

   where $\Phi_F(x_1,\ldots,x_n,a,s)$ is a formula uniform in s, all of whose free variables are among $a,s,x_1,\ldots,x_n$. Notice that we do not assume that successor state axioms have the exact syntactic form as those obtained earlier by combining the ideas of Davis/Haas/Schubert and Pednault. The discussion there was meant to motivate one way that successor state axioms of the form (4.10) might arise, but nothing in the development that follows depends on that earlier approach.
   As for action precondition axioms, the uniformity of Φ_F guarantees that the truth value of F in the successor situation $do(a,s)$ is determined entirely by the current situation s. In systems and control theory, this is called the *Markov property*.
2. A successor state axiom for an $(n+1)$-ary functional fluent f is a sentence of $\mathcal{L}_{sitcalc}$

"This book is a masterful integration of several decades of work on first-order logic, situation calculus, logic programming, and semantics of time and knowledge. The result is a unified, well-thought-out, and systematic approach to dynamical systems that spans much of modern computer science and AI." — Johan van Benthem, Professor of Logic, Stanford University and University of Amsterdam

"This book describes a thoroughly developed logic- and situation-calculus-based system for problem solving and planning. Its emphasis on theory is especially important for the ambitious student who wants to look beyond immediate applications toward the goal of human-level artificial intelligence." — John McCarthy, Professor Emeritus of Computer Science, Stanford University

"Reiter's new book, *Knowledge in Action*, offers the first systematic account of the logical approach to cognitive robotics, a field that he and his colleagues have developed over the past decade. The unique feature of this approach rests in its capacity to admit specifications in the form of meaningful knowledge fragments, to piece those fragments together by logical and probabilistic inferences, and to use those inferences to guide both manipulative and perceptual actions by programmable agents. A must for anyone concerned with the foundations of commonsense knowledge or the design of autonomous dynamical systems." — Judea Pearl, Department of Computer Science, University of California Los Angeles

"This outstanding work should be on the shelf of anyone concerned with logical control of physical processes." — Anil Nerode, Goldwin Smith Professor of Mathematics, Cornell University

The MIT Press · Massachusetts Institute of Technology · Cambridge, Massachusetts 02142 · http://mitpress.mit.edu

Cover illustration: *The mirrored universe* by Flora Pirri

0-262-18218-1