ERIK T. MUELLER

COMMONSENSE REASONING

MORGAN KAUFMANN

Commonsense Reasoning

Erik T. Mueller
IBM Thomas J. Watson Research Center

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann Publishers is an imprint of Elsevier

Acquisitions Editor and Publisher	Denise Penrose
Publishing Services Manager	Simon Crump
Production Editor	Dawnmarie Simpson
Editorial Assistant	Valerie Witte
Cover Design	Hannus Design Assoc.
Composition	Cepha Imaging Pvt Ltd
Technical Illustration	Dartmouth Publishing, Inc.
Copyeditor	Julie Nemer
Proofreader	Broccoli Information Management
Indexer	Broccoli Information Management
Interior printer	The Maple-Vail Book Manufacturing Group
Cover printer	Phoenix Color

Front cover photograph, pan90475 (RM) USA, New York, New York City, Central Park, People walking in The Gates © Panoramic Images/Getty Images.

Morgan Kaufmann Publishers is an imprint of Elsevier.
500 Sansome Street, Suite 400, San Francisco, CA 94111

This book is printed on acid-free paper.

Library of Congress Cataloging-in-Publication Data
Mueller, Erik T.
Commonsense reasoning/Erik T. Mueller.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-12-369388-4 (hardcover : alk. paper)
ISBN-10: 0-12-369388-8 (hardcover : alk. paper) 1. Commonsense reasoning–Automation. 2. Artificial intelligence–Mathematics. 3. Logic, Symbolic and mathematical–Data processing. I. Title.
Q338.85.M84 2006
153.4'3–dc22

2005031664

ISBN 13: 978-0-12-369388-4
ISBN 10: 0-12-369388-8

For information on all Morgan Kaufmann publications,
visit our Web site at www.mkp.com or www.books.elsevier.com

Printed in the United States of America

06 07 08 09 10 5 4 3 2 1

Contents

PART VIII Appendices

Foreword

An eminent professor of logic once said to me, "Why do you bother devising all those little predicate calculus theories? We already know that first-order predicate calculus is capable of expressing almost anything, so what is the point?" This question typifies the attitude of a certain breed of logician, for whom the quintessence of intellectual endeavour is the study of the metalevel properties of various formalisms—their expressive power, their computational limitations, and the relationships between one formalism and another. Without doubt such work is, from an academic point of view, noble and worthwhile. So I did wonder, for several minutes, whether the eminent logician was perhaps right. But then it occurred to me that no one ever says, "Why do you bother giving your undergraduates all those little programming exercises? We already know that Turing machines can compute almost anything, so what is the point?"

The point, of course, is that the gap between believing something to be possible and knowing how to achieve it is very wide indeed. There is an art to programming, and learning how to do it well takes many years. If the eminent logician had not retired to his office before allowing me a return blow, I would have replied that what goes for programming also goes for logic: There is an art to the use of logic for knowledge representation, and learning it requires much practise. So it is surprising that more books aimed at teaching this art do not exist. Fortunately, we can now add to this small corpus the admirable volume by Erik Mueller that you are hopefully about to read. But this book is more than just a guide to building complex representations of knowledge in logic because its target is an area that might be thought of as the nemesis of artificial intelligence, namely common sense.

One of the starkest lessons of AI research in the twentieth century was that it is those aspects of human behaviour that we most take for granted that are the hardest to emulate on a computer. A two-year-old child who finds a chocolate bar hidden in his mother's bag is performing a feat of common sense that our most sophisticated AI systems would be utterly incapable of matching. It is true that we now have programs that can defeat chess grandmasters—but only at chess. To a cognitive scientist, the most remarkable thing about a chess grandmaster is that, having played a great game of chess, she can then go and make a cup of tea. The very same biological apparatus that has mastered chess had long beforehand mastered the everyday physical world of solids, liquids, gravity, surfaces, and shapes, not to mention the everyday social world of interaction with her peers.

What is the right way to approach the daunting problem of endowing computers and robots with common sense? There's no shortage of opinions on this question among AI researchers and cognitive scientists. Perhaps we should be inspired by biology. Perhaps we should imitate evolution. Perhaps we should eschew nature and instead embrace mathematical formalisms such as logic. If we were empirical scientists, there would be a right and a wrong answer, whether or not we yet knew which was which. But insofar as we are engineers, there can be many right answers. With a mere half century of thinking behind us—a very short time in the history of ideas—we still do not know how far the symbolic approach exemplified by this book can take us towards human-level artificial intelligence. But we do know that the symbolic approach makes for elegant designs with provable properties in a wide range of application areas where systems with a little extra intelligence have the edge. So Erik Mueller's book is necessary and timely, and I hope it gains the wide readership it deserves.

Murray Shanahan
Imperial College London
July 2005

Preface

Commonsense reasoning is the sort of reasoning we all perform about the everyday world. We can predict that, if a person enters a kitchen, then afterward the person will be in the kitchen. Or that, if someone who is holding a newspaper walks into a kitchen, then the newspaper will be in the kitchen. Because we make inferences such as these so easily, we might get the impression that commonsense reasoning is a simple thing. But it is very complex.

Reasoning about the world requires a large amount of knowledge about the world and the ability to use that knowledge. We know that a person cannot be in two places at once, that a person can move from one location to another by walking, and that an object moves along with a person holding it. We have knowledge about objects, events, space, time, and mental states and can use that knowledge to make predictions, explain what we observe, and plan what to do.

This book addresses the following question: How do we automate commonsense reasoning? In the last few decades, much progress has been made on this question by artificial intelligence researchers. This book provides a detailed account of this progress and a guide to automating commonsense reasoning using logic. We concentrate on one formalism, the event calculus, that incorporates many of the discoveries of the field. Although the event calculus is defined by a handful of first-order logic axioms, it enables reasoning about a wide range of commonsense phenomena.

Why Commonsense Reasoning?

Why study commonsense reasoning? The first reason for studying commonsense reasoning is practical. Automated commonsense reasoning has many applications

ranging from intelligent user interfaces and natural language processing to robotics and vision. Commonsense reasoning can be used to make computers more human-aware, easier to use, and more flexible.

The second reason for studying commonsense reasoning is scientific. Commonsense reasoning is a core capability of intelligence that supports many other high-level capabilities. The ability to understand what is happening in a story, for example, crucially involves commonsense reasoning. By studying commonsense reasoning we can gain a greater understanding of what intelligence is.

Approach

The approach to commonsense reasoning taken in this book is not shared by all researchers. My approach can be characterized by the following assumptions.

I assume, along with most cognitive scientists, that commonsense reasoning involves the use of representations and computational processes that operate on those representations.

I assume along with researchers in symbolic artificial intelligence, that these representations are symbolic.

I assume, along with researchers in logic-based artificial intelligence, that commonsense knowledge is best represented declaratively rather than procedurally.

I use the declarative language of many-sorted first-order logic.

I do not claim that the methods for commonsense reasoning presented in this book are the methods used by humans. This book presents one way of automating commonsense reasoning. How humans perform commonsense reasoning is an interesting topic, but it is not the topic of this book. There is evidence both for and against the use of logic in human reasoning.

Intended Audience

This book is intended for use by researchers and students in the areas of computer science, artificial intelligence, mathematics, and philosophy. It is also intended for use by software designers wishing to incorporate commonsense reasoning into their applications. The book can be used as a graduate-level textbook for courses on commonsense reasoning and reasoning about action and change, as well as a reference work for researchers working in these areas. It will be of interest to those using logic as their primary technique, as well as those using other techniques. This book can also be used as a supplementary graduate-level or advanced undergraduate textbook for courses on knowledge representation and artificial intelligence.

I assume the reader has some familiarity with first-order logic, although reviews of first-order logic are provided in Chapter **??** and Appendix A.

Roadmap

This book consists of 17 chapters and four appendices. The chapters are organized into seven parts.

Part **??** describes the foundations of the event calculus.

Part II deals with various commonsense phenomena. Chapter 3 discusses the effects of events. Chapter 4 discusses the triggering of events by conditions. Chapter 5 discusses the commonsense law of inertia. Chapter 6 discusses the indirect effects of events. Chapter 7 discusses continuous change. Chapter 8 discusses concurrent events. Chapter 9 discusses nondeterministic effects of events.

Part III deals with important commonsense domains. Chapter 10 presents axiomatizations of relational and metric space, and discusses reasoning about object identity, space, and time. Chapter 11 presents axiomatizations of the mental states of agents, including beliefs, goals, plans, and emotions.

Part IV discusses default reasoning.

Part V deals with programs and applications. Chapter 13 discusses the Discrete Event Calculus Reasoner program used to solve event calculus reasoning problems, and Chapter 14 discusses several real-world applications.

Part VI reviews logical and nonlogical methods for commonsense reasoning and discusses their relationship to the event calculus. Chapter 15 reviews logical methods, and Chapter 16 reviews nonlogical methods.

Part VII presents my conclusions.

Material Covered

The skills that make up human commonsense reasoning are complex, and the body of research related to it is large. Because no book can realistically cover every aspect of commonsense reasoning, a choice had to be made about what this book would cover. The coverage of this book was determined by the following considerations.

Most instances of commonsense reasoning involve action and change because action and change are pervasive aspects of the world. It is crucial for any method for commonsense reasoning to deal with action and change. Therefore, a large

part of this book is devoted to this topic. In addition to reasoning about action and change, or the domain of time, this book covers two other significant domains of commonsense reasoning: space and mental states, including emotions, goals, plans, and beliefs. This book also covers default reasoning and reasoning about object identity.

Over the last few decades, researchers have developed a number of logics for commonsense reasoning. It would take much time and space to cover all of these in detail. Hence, this book concentrates on one logic, the event calculus, which incorporates many of the features of the other logics. The reader who understands the event calculus will be well equipped to understand the other logics. They are closely related to the event calculus, and some are provably equivalent to the event calculus. Chapter 15 compares the event calculus with other logics, and detailed bibliographic notes throughout the book discuss research performed using other logics.

Several types of commonsense reasoning are not covered by this book. Reasoning under uncertainty about action and change is not covered because this is not a well-developed area. But this book does cover nondeterminism, and some initial work on the use of probability theory for reasoning about action and change is reviewed (in Section 16.3). Although there is a large published literature on machine learning, relatively little research on learning commonsense knowledge has been performed, so this is not included. The related area of analogical processing is reviewed (in Section 16.2). Although this book covers most features of the event calculus, it does not cover continuous change described using differential equations; this book does, however, cover continuous change described by closed-form expressions.

Supplemental Materials

Web Site and Reasoning Programs

The book web site at www.signiform.com/csr/ contains additional material related to this book. This includes links to event calculus reasoning programs that can be downloaded, such as the Discrete Event Calculus Reasoner program discussed in Chapter 13.

Exercises and Solutions

Exercises are provided at the end of Chapters 2 through 16. Solutions to selected exercises are provided in Appendix D. Solutions to further exercises are available online to instructors who have adopted this text. Register at www.textbooks.elsevier.com for access.

Text and Figure Acknowledgments

Portions of the book, *The Snowman*, by Raymond Briggs, courtesy of Random House.

Portions of the book, *Solving the Frame Problem*, by Murray Shanahan, courtesy of MIT Press.

Portions of the article "A Logic of Actions" by Patrick J. Hayes, in *Machine Intelligence*, Vol. 6, courtesy of Edinburgh University Press.

Portions of the article, "Event Calculus Reasoning Through Satisfiability" by Erik T. Mueller, in *Journal of Logic and Computation*, courtesy of *Journal of Logic and Computation* and Oxford University Press.

Portions of the article, "Story understanding through multi-representation model construction" by Erik T. Mueller, in *Text Meaning*: *Proceedings of the HLT-NAACL 2003 Workshop*, courtesy of Association for Computational Linguistics.

Portions of the discussion, "[Protocol of online discussion about the article A logical account of the common sense informatic situation for a mobile robot]" by Murray Shanahan, courtesy of "Electronic News Journal on Reasoning about Actions and Change."

Figure 2, "Shanahan's Circuit," reprinted from *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, by Murray Shanahan, "The ramification problem in the event calculus," p. 145, 1999, with permission from Elsevier.

Figure 4, "Thielscher's Circuit," reprinted from "Ramification and Causality," by Michael Thielscher, p. 19, 1996, with permission from Berkeley, CA: International Computer Science Institute.

This Page Intentionally Left Blank

Acknowledgments

It is my great pleasure to acknowledge the people who have contributed to this book, although space constraints do not permit me to refer to everyone by name. I am particularly indebted to the inventors of the event calculus. This book owes its existence to the pathbreaking work of Robert A. Kowalski, Marek J. Sergot, Murray Shanahan, Rob Miller, and other researchers in the field of reasoning about action and change.

I am privileged to work in the academically stimulating environment of the IBM T. J. Watson Research Center. Special thanks are due to Doug Riecken, the creator of the Commonsense Computing department in which I work. I am grateful for the support of Guruduth S. Banavar, Arthur C. Ciccolo, Thomas A. Cofino, Paul Horn, Charles W. Lickel, Alfred Spector, John Turek, Jürg von Känel, and Ellen J. Yoffa. Over the years, I have learned a great deal about formal knowledge representation from Leora Morgenstern. My research benefited immensely from many conversations with Tessa A. Lau, Daniel Oblinger, and David Canfield Smith. Other colleagues who helped in a number of ways were Rangachari Anand, Karen Appleby, Richard J. Cardone, David Ferrucci, Leiguang Gong, Robert Hoch, Xuan Liu, Mark Podlaseck, Moninder Singh, Vugranam Sreedhar, Malgorzata Stys-Budzikowska, and Christopher Welty.

I feel very fortunate to receive continuing inspiration from the faculty, staff, and students of the MIT Media Laboratory. I particularly value the support and friendship of Marvin Minsky and Push Singh, who made it possible for me to begin to work seriously in the area of commonsense reasoning by inviting me to spend 6 months as a research scientist at the Media Lab. Both Marvin and Push have helped improve my thinking about this area in innumerable ways. Other people

from the Media Lab who have helped me in my work are Barbara A. Barry, Walter Bender, Nicholas L. Cassimatis, Timothy Chklovski, Glorianna Davenport, Robert W. Lawler, Henry Lieberman, Xinyu Hugo Liu, Betty Lou McClanahan, Nicholas Negroponte, Deb Roy, Warren Sack, Oliver G. Selfridge, and Ted Selker.

Other researchers have been very generous in assisting me. Conversations with Benjamin Kuipers helped me to define a research strategy for commonsense reasoning. The excellent paper of Murray Shanahan and Mark Witkowski on event calculus planning through satisfiability sparked my interest in this technique. My work has vastly improved as a result of the helpful comments and criticism I received from those who served as anonymous referees of my papers on the event calculus. I am grateful to John McCarthy for his ongoing support. I thank Geoff Sutcliffe for his collaboration on the use of automated theorem proving for event calculus reasoning. Discussions with Srini Narayanan have also been helpful.

I also acknowledge the reviewers, Chitta Baral, Arizona Stata University; Patrick Hayes, Florida Institute for Human and Machine Cognition; Michael Thielscher, Technische Universität Dresden; Mary-Anne Williams, University of Technology, Sydney; and an anonymous reviewer, who took the time to read the entire manuscript and provide me with feedback. Vladimir Lifschitz's helpful comments significantly improved the section on the $\mathcal{C}+$ action language.

I have had the good fortune to work with an extremely talented group of people at Morgan Kaufmann and Elsevier. Special thanks go to Denise E. M. Penrose, publisher, whose strong support for this project has never faltered. The editorial and production process was a smooth and efficient experience thanks to the excellent work of Valerie Witte, editorial assistant, and of Dawnmarie E. Simpson, project manager. I also thank Julie F. Nemer for copyediting the book, and Dartmouth Publishing for producing many of the figures.

I am grateful for the support and friendship of Alexandra Guest, Anne-Marie Hainault, Karen M. Kensek, Lo-Ann Lai, Christopher A. Perez, Joseph E. Pingree, Lorenzo A. Sadun, and Uri Zernik. My thanks also go to Eyal Amir, Patrick Doherty, Jeffrey Epstein, Dov M. Gabbay, Douglas B. Lenat, Jane Spurr, and Jeff Wolfers, who assisted this project in many ways.

My greatest debt is to my family. I especially thank my parents, Diana E. Mueller and Robert E. Mueller, for their love and enthusiasm for my work. I am grateful for the love and moral support of my sister, Rachel A. G. Mueller-Lust, and brother-in-law, Andrew D. Mueller-Lust. I thank my in-laws, David Hackett Fischer and Judith H. Fischer, who read the manuscript and provided helpful feedback. My thanks also go to Ilus Lobl, John H. Fischer, Victor Lobl, Anne W. Fischer, Frederick C. Turner, and Althea W. Turner. Sophie and Nicole Fischer provided welcome distractions. Words cannot express my love and gratitude to my wife, Susanna F. Fischer, and our son, Matthew E. Mueller, for the happiness we share.

Erik T. Mueller
July 2005

CHAPTER 1

Introduction

This book is about commonsense reasoning, the sort of reasoning people perform in daily life. Here are some examples of commonsense reasoning:

1. In the living room, Lisa picked up a newspaper and walked into the kitchen. Where did the newspaper end up? It ended up in the kitchen.
2. Kate set a book on a coffee table and left the living room. When she returned, the book was gone. What happened to the book? Someone must have taken it.
3. Jamie walks to the kitchen sink, puts the stopper in the drain, turns on the faucet, and leaves the kitchen. What will happen as a result? The water level will increase until it reaches the rim of the sink. Then the water will start spilling onto the floor.
4. Kimberly turns on a fan. What will happen? The fan will start turning. What if the fan is not plugged in? Then the fan will not start turning.
5. A hungry cat saw some food on a nearby table. The cat jumped onto a chair near the table. What was the cat about to do? The cat was about to jump from the chair onto the table in order to eat the food.

This book is concerned with understanding and describing commonsense reasoning to such a level of detail that it can be *automated*, or performed automatically by a machine such as a computer. It reviews methods for commonsense reasoning and describes in detail a method for commonsense reasoning using the event calculus, an extension of first-order logic.

1.1 What Is Commonsense Reasoning?

Commonsense reasoning is a process that involves taking information about certain aspects of a scenario in the world and making inferences about other aspects of the scenario based on our *commonsense knowledge*, or knowledge of how the world works. Commonsense reasoning is essential to intelligent behavior and thought. It allows us to fill in the blanks, to reconstruct missing portions of a scenario, to figure out what happened, and to predict what might happen next. Commonsense reasoning stands in contrast to various types of expert reasoning such as economic, legal, mathematical, medical, and scientific reasoning.

1.2 Key Issues of Commonsense Reasoning

Although commonsense reasoning comes naturally to us and appears to be simple, it is actually a complex process. In this section, we examine the previously mentioned examples of commonsense reasoning in detail. We introduce fundamental concepts and point out some of the key issues that must be addressed by any method for commonsense reasoning.

Consider the first scenario.

Representation

In the living room, Lisa picked up a newspaper …

In order to automate commonsense reasoning about a scenario such as this, we must first build a representation of the scenario. A *representation* is something that resembles something else. For the purpose of automating commonsense reasoning, the representation should be a data structure or a sentence of a language defined by a formal syntax, and the representation should facilitate automated reasoning.

Objects, Properties, Events, and Time

Several fundamental entities must be represented. First, we must represent objects in the world and agents such as persons and animals; we must represent Lisa, the newspaper, and the living room. Second, we must represent properties of the world that change over time; we need to represent the time-varying locations of Lisa and the newspaper. Third, we must represent events or actions that occur in the world; we need to represent the event of Lisa picking up the newspaper. Fourth, we must represent time; we must represent that Lisa picked up the newspaper when she and the newspaper were in the living room.

Object Identity

We must represent the identities of objects; we must represent the fact that Lisa and the newspaper are not the same object.

Reasoning

Having formed a representation of the scenario, we can then perform commonsense *reasoning* or *inference*. Because our goal is automation, the method of reasoning should be expressed as an algorithm or formal rule that takes representations as input and produces representations as output.

Representations of Commonsense Knowledge

We must construct representations of commonsense knowledge that can be used by the reasoning method to reason about this scenario as well as other scenarios.

Effects of Events

We must be able to represent and reason about the effects of events on world properties. We must be able to reason from a specific event and general knowledge about the effects of events to the specific effects of the specific event. We should be able to represent that, if a person picks up an object, then the person will be holding that object. Given that Lisa picked up the newspaper, and this piece of commonsense knowledge, we should be able to infer that Lisa was then holding the newspaper.

Context-Sensitive Effects

We must be able to represent and reason about the context-sensitive effects of events. We should be able to represent that, if a person picks up a slippery object and is not careful, then the person will not be holding the object.

Nondeterministic Effects

We must also be able to represent and reason about events with nondeterministic effects. We should be able to represent that if a person picks up a slippery object, then the person may or may not be holding the object.

Concurrent Events

We must be able to represent and reason about concurrent events. We should be able to represent that certain concurrent events are impossible; for example, a person cannot walk into two rooms simultaneously. We must be able to reason about concurrent events with cumulative or canceling effects. For example, if a shopping cart is pushed, it moves forward. If it is pulled, it moves backward. But if it is simultaneously pulled and pushed, then it moves neither forward nor backward; instead, it spins around.

Space

...and walked into the kitchen.

In order to automate commonsense reasoning, we must be able to deal with space. We must represent the knowledge that, after a person walks into a room, the person will be in that room. From this knowledge and the fact that Lisa walked into the kitchen, we should be able to infer that afterward Lisa was in the kitchen.

Indirect Effects

Where did the newspaper end up? It ended up in the kitchen.

In order to make this inference, we must be able to reason about the indirect effects or ramifications of events. We know that, if a person is holding an object, then the object moves along with the person.

Next, we consider the second scenario.

Kate set a book on a coffee table ...

We must represent the effect of setting an object on another object, and we should be able to reason that, after Kate set the book on the coffee table, the book was on the coffee table.

Preconditions

We should also be able to infer that before Kate set the book on the table, she was holding the book and she was near the table; we must be able to represent and reason about the preconditions of actions or events. We need to represent two preconditions of a person placing an object onto a surface: (1) the person must be holding the object, and (2) the person must be near the surface.

...and left the living room.

We should be able to infer that after Kate left the living room, she was no longer in the living room.

Commonsense Law of Inertia

We should also be able to infer that the book was still on the table in the living room after she left.

When she returned, ...

We should be able to infer that after Kate returned, the book was probably still on the table in the living room. That is, unless a person or animal moved the book, or some natural phenomenon such as an earthquake occurred, the book was where Kate left it. This property of the commonsense world, that things tend to stay the same unless affected by some event, is known as the *commonsense law of inertia*.

But we learn that the book was no longer in the living room:

…the book was gone.

In this case we should be able to infer that someone took the book out of the room (or a natural phenomenon occurred):

What happened to the book? Someone must have taken it.

Next, we consider the third scenario.

Delayed Effects and Continuous Change

Jamie walks to the kitchen sink, puts the stopper in the drain, turns on the faucet, and leaves the kitchen.

We have so far seen that it is necessary for us to be able to represent and reason about the immediate effects of events, such as putting a stopper in a drain and turning on a faucet. Thus, we should be able to infer that the stopper is in the drain, the faucet is running, and the sink is filling. In addition, we should be able to represent and reason about the delayed effects of events:

What will happen as a result? The water level will increase until it reaches the rim of the sink. Then the water will start spilling onto the floor.

Making these inferences involves representing and reasoning about continuous change. We should be able to represent that if a faucet is turned on with the stopper in place, then the water level will increase with time.

Release from the Commonsense Law of Inertia

Recall that the commonsense law of inertia states that things stay the same unless affected by some event. But notice that the water level continues to change after the event of turning on the faucet. Therefore we must be able to represent that, after the faucet is turned on, the water level is released from the commonsense law of inertia and is permitted to vary. We must further represent that the water level is proportional to the time elapsed since the faucet was turned on.

Triggered Events

In order to reason about this scenario, we must also be able to represent and reason about triggered events. The water level does not increase endlessly. When a sink is filling and the water reaches the rim of the sink, the sink will overflow. We should be able to represent and reason that when a sink overflows, the water starts spilling onto the floor and the water level stops increasing. At this point, the water level will again be subject to the commonsense law of inertia.

Consider the fourth scenario.

Default Reasoning

When we perform commonsense reasoning, we rarely have complete information. We are unlikely to know the state of affairs down to the last detail, everything about the events that are occurring, or everything about the way the world works. Therefore, when we perform commonsense reasoning, we must jump to conclusions. Yet, if new information becomes available that invalidates those conclusions, then we must also be able to take them back. Reasoning in which we reach conclusions and retract those conclusions when warranted is known as *default reasoning*. In the fourth scenario,

Kimberly turns on a fan. What will happen? The fan will start turning.

How can the method for commonsense reasoning conclude that the fan will start turning? In fact, the fan might not start turning if the fan is broken, if the fan is not plugged in, and so on. The world is filled with exceptions such as these. The method must be able to assume that things are as normal and conclude that the fan will start turning. If it is later learned that the fan is not plugged in, then the conclusion should be revised:

What if the fan is not plugged in? Then the fan will not start turning.

Two special cases of default reasoning are required for reasoning about events. First, although we are told that a fan is turned on, we do not know what other events occur. We do not know whether, for example, some other person is simultaneously attempting to turn off the fan. The method for commonsense reasoning must assume that this is not the case; that is, it must be assumed by default that unexpected events do not occur.

Second, although we know that a fan will start turning after it is turned on, we do not know what the other results of turning on the fan might be. Perhaps turning on the fan also unlocks a nearby door. The method for commonsense reasoning must assume by default that events do not have unexpected effects.

Next, we consider the fifth scenario.

Mental States

A hungry cat saw some food on a nearby table.

We must represent the piece of commonsense knowledge that if an agent has an unsatisfied goal, then the agent will form a plan to achieve that goal. In this case, if an animal has the goal to eat and has the belief that food is nearby, then the animal will form the plan to go to the food and eat it.

The cat jumped onto a chair near the table.

We must further represent that agents act on their plans. We should be able to infer that jumping onto a chair is part of the cat's plan to eat.

What was the cat about to do? The cat was about to jump from the chair onto the table in order to eat the food.

Based on the knowledge that agents act on their plans, we should be able to infer that the cat will complete the plan. After the cat eats the food, we should infer that the plan is completed. We may also then infer that the goal to eat is satisfied.

Reasoning Types

A method for automated commonsense reasoning must support several types of commonsense reasoning. The first is *temporal projection* or *prediction*, in which we start with an initial state and some events and then reason about the state that results from the events. The examples of Lisa walking into the kitchen, the kitchen sink overflowing, and the cat eating the food all involve temporal projection. The second type of reasoning is *abduction*, in which we start with an initial state and a final state and then reason about the events that lead from the initial state to the final state. The example of Kate's book disappearing involves abduction. The third type of reasoning is *postdiction*, in which we start with some events that lead to a state and then reason about the state prior to the events. If we are told that Lisa picked up a newspaper and was then holding the newspaper, we may reason that Lisa was not previously holding the newspaper.

1.2.1 Summary

Any method for automated commonsense reasoning must address the following.

Representation. The method must represent scenarios in the world and must represent commonsense knowledge about the world.

Commonsense entities. The method must represent objects, agents, time-varying properties, events, and time.

Commonsense domains. The method must represent and reason about time, space, and mental states. The method must deal with object identity.

Commonsense phenomena. The method must address the commonsense law of inertia, release from the commonsense law of inertia, concurrent events with cumulative and canceling effects, context-sensitive effects, continuous change, delayed effects, indirect effects, nondeterministic effects, preconditions, and triggered events.

Reasoning. The method must specify processes for reasoning using representations of scenarios and representations of commonsense knowledge. The method must support default reasoning, temporal projection, abduction, and postdiction.

1.3 Brief History of Commonsense Reasoning

Artificial intelligence researchers have been trying to invent ways of automating commonsense reasoning since the inception of the field in 1956. Work on commonsense reasoning can be divided into two categories: logical and nonlogical. Logical methods are reviewed in detail in Chapter 15, and nonlogical methods are reviewed in Chapter 16. In this section, we present a brief history of work on logical and nonlogical methods.

1.3.1 Logical Methods

In 1958, John McCarthy proposed using logic to give computer programs common sense. In the 1960s, he and Patrick J. Hayes introduced the situation calculus, a logical formalism for commonsense reasoning. In the 1970s, the crucial role of defaults in commonsense reasoning was recognized, and researchers began to formalize methods for default reasoning. Important formalisms for default reasoning such as circumscription and default logic appeared around 1980.

Taking their inspiration from the situation calculus, Robert Kowalski and Marek Sergot introduced the event calculus in 1986. In the late 1980s, several other logical formalisms began to appear, including the features and fluents framework, action languages, and the fluent calculus.

Since the early 1990s, logic-based commonsense reasoning has been the focus of intense activity. Researchers proposed a number of benchmark problems designed to expose issues of commonsense reasoning not yet addressed by the available formalisms. This led to a considerable evolution of the formalisms.

In this book, we use a version of the event calculus developed in the 1990s by Murray Shanahan and Rob Miller and a version that is equivalent for integer time, called the discrete event calculus. The event calculus has benefited enormously

Table 1.1 Benchmark problems leading to the addition of features to the event calculus

Benchmark problem	*Commonsense phenomena*	*Event calculus features added*
Bus ride scenario (Kartha, 1994)	Nondeterministic effects	Disjunctive event axioms (Shanahan, 1997b)
Chessboard scenario due to Raymond Reiter (Kartha and Lifschitz, 1994)	Nondeterministic effects	Determining fluents (Shanahan, 1997b)
Commuter scenario (Shanahan, 1999a)	Compound events	Three-argument *Happens* (Shanahan, 1999a)
Kitchen sink scenario (Shanahan, 1990)	Continuous change, triggered events	Trajectory axioms, trigger axioms (Shanahan, 1990)
Russian turkey scenario (Sandewall, 1994)	Nondeterministic effects	Release axioms (Shanahan, 1997b)
Soup bowl scenario (Gelfond, Lifschitz, and Rabinov, 1991)	Concurrent events	Cumulative effect axioms (R. Miller and Shanahan, 1999)
Stolen car scenario (Kautz, 1986)	Explanation	Abduction (Shanahan, 1997b)
Stuffy room scenario (Ginsberg and Smith, 1988a)	Indirect effects	State constraints, primitive and derived fluents (Shanahan, 1997b)
Thielscher's circuit (Thielscher, 1996)	Indirect effects	Causal constraints (Shanahan, 1999b)
Walking turkey scenario due to Matthew L. Ginsberg (Baker, 1991)	Indirect effects	Effect constraints (Shanahan, 1997b)
Yale shooting scenario (Hanks and McDermott, 1987)	Commonsense law of inertia	Circumscription, forced separation (Shanahan, 1997b)

from the investigation of benchmark problems. Table 1.1 shows some of the benchmark problems that led to the addition of features to the event calculus.

1.3.2 Nonlogical Methods

Starting in the early 1970s, Roger Schank, Robert Abelson, and their colleagues and students developed a number of knowledge structures and inference methods for use in natural language understanding systems. A notation known as conceptual dependency (CD) was proposed for representing actions and states. Knowledge structures called scripts were introduced to represent stereotypical

activities such as attending a birthday party. A taxonomy of human plans and goals was developed, and representations for the themes of stories were introduced.

Starting in the late 1970s, researchers working in the area of qualitative reasoning developed techniques for automated reasoning about physical mechanisms such as bathtubs, clocks, electrical circuits, pressure regulators, sliding blocks, and water tanks. Physical devices are described using a modeling language, and simulation algorithms are used to perform reasoning. These techniques are useful for commonsense reasoning in the physical domain. Some of the techniques have been recast in first-order logic.

Beginning in the early 1980s, researchers developed methods for analogical processing. These methods can be used to find an analogy between a familiar domain and a novel domain and then to use the analogy to generate candidate inferences about the novel domain. Analogical processing is not a complete method for commonsense reasoning, because candidate inferences must still be evaluated and repaired using other commonsense reasoning techniques.

Probability theory has a long history and is well suited to default reasoning. Since the late 1980s, some work has been performed on using probabilistic reasoning for reasoning about action and change, but this approach is not as well developed as the logical approach. The logical and probabilistic approaches are closely related, and the integration of logic and probability theory is an active area of research.

Starting in the early 1970s, the society of mind theory was developed by Marvin Minsky and his colleagues and students. This theory views human common sense as a vast collection of skills involving multiple representation and reasoning techniques. Unlike most other approaches, this approach places a great emphasis on procedural representations of knowledge and on the ways that procedures can monitor and influence one another.

1.4 The Event Calculus

The event calculus addresses all the key issues of commonsense reasoning described in Section 1.2. Using the event calculus we can represent commonsense knowledge, represent scenarios, and use the knowledge to reason about the scenarios.

1.4.1 Events, Fluents, and Timepoints

The basic notions of the event calculus are as follows. An *event* represents an event or action that may occur in the world, such as a person picking up a glass. We use the words *event* and *action* interchangeably. A *fluent* represents a time-varying property of the world, such as the location of a physical object. A *timepoint* represents an instant of time, such as 9:30 AM Greenwich Mean Time on November 13, 2007. The event calculus uses *linear time*, in which time is considered to be

a line, rather than the *branching time* of the situation calculus, in which time is considered to be a tree.

An event may occur or happen at a timepoint. A fluent has a truth value at a timepoint or over a timepoint interval; the possible truth values are true and false. After an event occurs, the truth values of the fluents may change. We have commonsense knowledge about the effects of events on fluents. Specifically, we have knowledge about events that initiate fluents and events that terminate fluents. For example, we know that the event of picking up a glass initiates the fluent of holding the glass and that the event of setting down a glass terminates the fluent of holding the glass. We represent these notions in first-order logic as follows.

$HoldsAt(f, t)$ represents that fluent f is true at timepoint t.

$Happens(e, t)$ represents that event e occurs at timepoint t.

$Initiates(e, f, t)$ represents that, if event e occurs at timepoint t, then fluent f will be true after t.

$Terminates(e, f, t)$ represents that, if event e occurs at timepoint t, then fluent f will be false after t.

1.4.2 A Simple Example

Here is a simple example of how the event calculus works. We use a simplified version of the event calculus, consisting of the following single axiom:

$$(Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg \exists e, t\,(Happens(e, t) \wedge t_1 < t \wedge t < t_2 \wedge Terminates(e, f, t))) \Rightarrow HoldsAt(f, t_2) \quad (1.1)$$

This axiom states that if an event occurs and the event is known to initiate a particular fluent, then that fluent will be true from just after the moment the event occurs, until and including the moment an event occurs that is known to terminate the fluent.

Now let us see how this axiom can be used to solve a simple commonsense reasoning problem, using one event and one fluent. The event $WakeUp(p)$ represents that person p wakes up, and the fluent $Awake(p)$ represents that person p is awake. Our commonsense knowledge consists of the following.

If a person wakes up, then the person will be awake:

$$Initiates(e, f, t) \Leftrightarrow \exists p\,(e = WakeUp(p) \wedge f = Awake(p)) \quad (1.2)$$

If a person falls asleep, then the person will no longer be awake:

$$Terminates(e, f, t) \Leftrightarrow \exists p\,(e = FallAsleep(p) \wedge f = Awake(p)) \quad (1.3)$$

Now suppose we have the following scenario. Nathan is not awake at timepoint 0:

$$\neg HoldsAt(Awake(Nathan), 0) \tag{1.4}$$

The only known event is that Nathan wakes up at timepoint 1:

$$Happens(e, t) \Leftrightarrow e = WakeUp(Nathan) \wedge t = 1 \tag{1.5}$$

From this commonsense knowledge and scenario, and event calculus axiom (1.1), we can deduce that Nathan is awake at timepoint 3. That is, we can prove

$$(1.1) \wedge (1.2) \wedge (1.3) \wedge (1.4) \wedge (1.5) \models HoldsAt(Awake(Nathan), 3)$$

The proof runs as follows. From (1.3) and (1.5), we have $\neg\exists e, t\,(Happens(e, t) \wedge 1 < t \wedge t < 3 \wedge Terminates(e, Awake(Nathan), t))$. From this, (1.1), (1.2), (1.5), and $1 < 3$, we have $HoldsAt(Awake(Nathan), 3)$, as required.

Adding commonsense knowledge and event occurrences to this formalization requires us to modify (1.2), (1.3), and (1.5). Later, in Sections 2.6 and 2.7, we show how circumscription allows us to add commonsense knowledge and event occurrences simply by adding axioms.

1.4.3 Automated Event Calculus Reasoning

In the logic-based approach to commonsense reasoning, knowledge is represented *declaratively* as logical formulas rather than *procedurally* as computer code. Using a declarative knowledge representation has two main advantages. First, the same knowledge can be used for different types of commonsense reasoning such as temporal projection, abduction, and postdiction. If a procedural knowledge representation is used, knowledge must often be duplicated for each type of commonsense reasoning. Second, using a declarative knowledge representation allows us to use the latest, off-the-shelf, automated, theorem-proving techniques to solve reasoning problems. If a procedural knowledge representation is used, reasoning techniques must often be built from scratch or reinvented.

Of course, entailment in first-order logic is *undecidable*: There is no algorithm that, given arbitrary formulas of first-order logic ψ and π, will eventually respond "yes" if ψ entails π and "no" if ψ does not entail π. First-order logic entailment is only *semidecidable*: There are algorithms that, given arbitrary formulas of first-order logic ψ and π, will respond "yes" if ψ entails π; if ψ does not entail π, the algorithms may eventually respond "no" or may never terminate. It turns out, however, that many real-world reasoning problems in the event calculus can be solved efficiently by computer programs.

Several programs that perform automated reasoning in the event calculus have been constructed, as listed in Table 1.2. The Discrete Event Calculus Reasoner is

Table 1.2 Event calculus reasoning programs

Description	*Technique*	*Reasoning types*
Event calculus planner (Shanahan, 2000a, 2000b)	Abductive logic programming	Abduction
Event calculus planner (Shanahan and Witkowski, 2004)	Propositional satisfiability	Abduction
Discrete Event Calculus Reasoner (Mueller, 2004a, 2004b)	Propositional satisfiability	Deduction, abduction, postdiction, model finding
Discrete event calculus theorem prover (Mueller and Sutcliffe, 2005a, 2005b; Sutcliffe and Suttner, 2005)	First-order logic automated theorem proving	Deduction

discussed in detail in Chapter 13. These programs rely on various solvers and provers, namely, logic programming languages, satisfiability (SAT) solvers, and first-order automated theorem provers. Improving the efficiency of these solvers and provers is a major focus of activity. As better solvers and provers are developed, they can be plugged into event calculus reasoning programs.

The SAT approach is particularly effective. A SAT solver takes as input a set of Boolean variables and a propositional formula over those variables and produces as output zero or more models or satisfying truth assignments, truth assignments for the variables such that the formula is true. SAT solvers take a propositional formula in conjunctive normal form: a conjunction of clauses where each clause is a disjunction of literals and where each literal is a variable or a negated variable. In order to use a SAT solver to solve an event calculus problem, formulas of first-order logic must be transformed into formulas of the propositional calculus. This is accomplished by restricting the problem to a finite universe. Although entailment in the propositional calculus is decidable, it is *NP-complete*, or believed in the worst case, to take a number of steps that is exponential on the size of the problem. But, in practice, real-world SAT problems consisting of well over 10,000 variables can be solved efficiently.

Bibliographic Notes

People have long sought to describe and capture commonsense reasoning. Logic was developed to characterize valid reasoning (Kneale and Kneale, 1962, pp. 738–739). The advent of computers led to the field of artificial intelligence and to a call by McCarthy (1959) to use logic to build computer programs with common sense. Computer science and artificial intelligence drove the development

of new logics (Gabbay and Guenthner, 2001, p. vii). Crevier (1993), Russell and Norvig (2003, pp. 16–27), and McCorduck (2004) provide histories of the field of artificial intelligence.

Book-length treatments of commonsense reasoning are provided by Hobbs and Moore (1985), Minsky (1986), E. Davis (1990), Lenat and Guha (1990), Lifschitz (1990a), Thanassas (1992), and Reiter (2001). Books on the related area of knowledge representation are by Reichgelt (1991), Baral (2003), and Brachman and Levesque (2004). Textbooks on artificial intelligence are by Nilsson (1998) and Russell and Norvig (2003).

The ability to perform commonsense reasoning starts to develop early in life. Several-month-old infants are able to reason using pieces of commonsense knowledge such as that a moving object must follow an unbroken path over time, an object cannot pass through another object, the parts of an object move together, and unsupported objects fall (Baillargeon, 1995; Spelke, Vishton, and von Hofsten, 1995).

Thagard (1996) reviews the basic notions of representations and computational processes that operate on those representations. Our list of key issues of commonsense reasoning follows those of other researchers. McCarthy (1984b, pp. 131–135) presents important aspects of commonsense capability. He considers reasoning about action and change to be a central aspect, writing that "the most salient commonsense knowledge concerns situations that change in time as a result of events" (p. 131). McCarthy also mentions other key aspects: knowledge about knowledge; knowledge about objects, space, beliefs, goals, intentions, and commonsense physics; logical inference; obtaining facts by observation; and default reasoning. For E. Davis (1990), the important areas are plausible reasoning, quantities and measurements, time, space, physics, minds, plans and goals, and society.

The assumptions of logic-based artificial intelligence are elaborated by Nilsson (1991). Whether logic is the right approach to commonsense reasoning has been hotly debated (Minsky, 1974, 1986, 1991b; Hayes, 1977; Kolata, 1982; R. C. Moore, 1982; Israel, 1985; McDermott, 1987; McCarthy and Lifschitz, 1987; Ginsberg, 1991; Nilsson, 1991; Birnbaum, 1991). The following advantages of logic have been pointed out:

- Logic can be used to represent any domain (R. C. Moore, 1982, p. 430).
- Via model theory, logic provides an account of the meaning of logical formulas (Hayes, 1977, pp. 559–561; Nilsson, 1991, pp. 34–40).
- Logic allows the representation of incomplete information (R. C. Moore, 1995, p. 7).

The following alleged disadvantages of logic have been pointed out:

- Logic focuses on deductive reasoning, and not all reasoning is deductive (McDermott, 1987, pp. 151–152; Birnbaum, 1991, pp. 59, 70–71). But

note that deduction is not the only type of reasoning that can be performed with logic; other types of reasoning such as abduction can be performed (Shanahan, 1997b, pp. 32–33).

- Logic is preoccupied with consistency, and anything can be deduced from a contradiction (Minsky, 1974, pp. 76–78; Hewitt, 1987, pp. 185–186; Birnbaum, 1991, p. 63). But note that logics have been developed without this property, such as paraconsistent logic (Priest, 2002) and active logic (Elgot-Drapkin and Perlis, 1990; Elgot-Drapkin, Kraus, Miller, Nirkhe, and Perlis, 1999). Also note that a logical theory can be revised by subtracting axioms as well as adding them (Hayes, 1979, pp. 54–55; (Israel, 1980, p. 101); 1985, pp. 436–437). See also the discussion of Bibel and Nicolas (1989, pp. 18–22).
- Logical reasoning is computationally inefficient (Minsky, 1974, p. 76; Birnbaum, 1991, p. 72). But note that the efficiency of theorem-proving systems is constantly being improved (Sutcliffe and Suttner, 2003; Le Berre and Simon, 2004).

The role of logic in human reasoning has been vigorously debated. Henle (1962), Braine (1978), Rips (1983, 1994), Braine and O'Brien (1998), and others argue that humans use a *mental logic* in which inference rules similar to those of the natural deduction of formal logic are applied and present experimental evidence supporting this theory. Other researchers disagree. Johnson-Laird (1983, 1993) proposes and presents experimental evidence for a *mental models* approach in which humans reason by building, updating, and evaluating models in the mind.

McCarthy (1963, 1968) and McCarthy and Hayes (1969) introduced the situation calculus. Kowalski and Sergot (1986) introduced the original event calculus within the framework of logic programming (Kowalski, 1979). The event calculus was reformulated in classical first-order logic by Shanahan (1995a, 1996, 1997b, 1999a, 1999b). R. Miller and Shanahan (1999, 2002) introduced several alternative formulations of the classical logic event calculus. Mueller (2004a) introduced the discrete event calculus. An introduction to the classical logic event calculus is provided by Shanahan (1999a). Symposia on logical formalizations of commonsense reasoning are regularly held (McIlraith, Peppas, and Thielscher, 2005), as are conferences on knowledge representation and reasoning (Dubois, Welty, and Williams, 2004).

Lifschitz (1989) created a list of commonsense reasoning benchmark problems, following a suggestion by John McCarthy. E. Davis (1990, pp. 4–12) presents a methodology for formalization of commonsense reasoning based on the use of benchmark problems. Sandewall (1994) proposes a systematic methodology for assessing entailment methods, which we discuss in the Bibliographic Notes of Chapter 15. McCarthy (1998a) argues that the field of artificial intelligence needs an equivalent to drosophilae, the fruit flies biologists use to study genetic mutations because of their short generation time. A list of unsolved benchmark

problems and a few solved ones is maintained by Leora Morgenstern (Morgenstern and Miller, 2004).

Conceptual dependency, scripts, plans, goals, and themes are discussed by Schank and Abelson (1977), Schank and Riesbeck (1981), and Dyer (1983). Bibliographic notes for nonlogical methods are provided in Chapter 16.

The kitchen sink scenario is from Shanahan (1990; 1997b, pp. 326–329; 1999a, pp. 426–428). This scenario can be traced back to Siklóssy and Dreussi (1973, pp. 426, 429) and Hendrix (1973, pp. 149, 159–167), who used the example of filling a bucket with water. McDermott (1982, pp. 129–133, 135–138) used the example of water flowing into a tank, and Hayes (1985, pp. 99–103) used the example of filling a bath.

The shopping cart example is from Shanahan (1997b, pp. 302–304). The hungry cat scenario is from Winikoff, Padgham, Harland, and Thangarajah (2002).

The distinction between declarative and procedural (or imperative) knowledge representation is discussed by McCarthy (1959, pp. 79–80), Winograd (1975), Hayes (1977), Winston (1977, pp. 390–392), and Genesereth and Nilsson (1987, pp. 2–4). Winograd (1975, p. 186) and McCarthy (1988, p. 299) suggest that declarative representations are more flexible. They point out that a fact represented declaratively can be used for many purposes, even unanticipated ones, whereas a fact represented procedurally has to be represented differently for each purpose.

The undecidability of first-order logic entailment, satisfiability, and validity is due to Church (1936/2004) and Turing (1936/2004). The semidecidability of first-order logic entailment, unsatisfiability, and validity is due to Gödel (1930/1967). It is useful to keep in mind the following relationships among entailment, satisfiability, and validity:

- ψ entails π if and only if $\psi \wedge \neg\pi$ is unsatisfiable.
- ψ entails π if and only if $\neg\psi \vee \pi$ is valid.
- ψ entails π if and only if $\psi \Rightarrow \pi$ is valid.
- ψ is valid if and only if $\neg\psi$ is unsatisfiable.

Automated theorem proving is treated in detail by Robinson and Voronkov (2001a, 2001b). SAT solvers are discussed by Du, Gu, and Pardalos (1997). Automated theorem-proving system competitions (Sutcliffe and Suttner, 2003) and SAT system competitions (Le Berre and Simon, 2004) are regularly held. The use of SAT solving for planning was proposed by Kautz and Selman (1992, 1996). The use of SAT solving in the event calculus was introduced by Shanahan and Witkowski (2004). Runtime statistics for the solution of some event calculus reasoning problems using SAT are given by Mueller (2003, 2004b, 2004c). NP-completeness is discussed by Garey and Johnson (1979). The NP-completeness of propositional satisfiability was proved by Cook (1971) and Levin (1973). The growth in the capabilities of SAT solvers is discussed by Selman, Kautz, and McAllester (1997), Kautz and Selman (2003), and Dechter (2003, pp. 186–188).

PART I

Foundations

This Page Intentionally Left Blank

CHAPTER 2

The Event Calculus

In this chapter, we present the foundations of the event calculus, a formalism for commonsense reasoning. We review first-order logic and describe some notational conventions. We discuss the basics of the event calculus and we present axiomatizations of the event calculus (EC) and the discrete event calculus (DEC). We discuss how to choose which axiomatization to use. We present reification, which is needed to represent statements about events and fluents in first-order logic. We discuss unique names axioms, conditions, circumscription, and domain descriptions, and we describe the types of reasoning that can be performed using the event calculus.

2.1 First-Order Logic

The event calculus is based on first-order logic, which consists of a syntax, semantics, and proof theory. The version of first-order logic used in this book is described in detail in Appendix A. Here we provide a summary.

2.1.1 Syntax of First-Order Logic

A language $\mathcal{L}$ of first-order logic is specified by disjoint sets of predicate symbols, function symbols, constants, and variables. Each predicate and function symbol has an arity. If the arity of a symbol is n, then we say that the symbol is n-ary.

A *term* is a constant, a variable, or, recursively, $\phi(\tau_1, \ldots, \tau_n)$, where ϕ is an n-ary function symbol and $\tau_1, \ldots, \tau_n$ are terms.

An *atom* is $\rho(\tau_1, \ldots, \tau_n)$, where ρ is an n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ are terms, or $\tau_1 = \tau_2$, where τ_1 and τ_2 are terms.

A *formula* is an atom, or, recursively, $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \Rightarrow \beta$, $\alpha \Leftrightarrow \beta$, $\exists v_1, \ldots, v_n\, \alpha$, or $\forall v_1, \ldots, v_n\, \alpha$, where α and β are formulas and $v_1, \ldots, v_n$ are variables.

The *scope* of the quantifier $\exists v$ in the formula $\exists v\, \alpha$ is α, and the scope of $\forall v$ in $\forall v\, \alpha$ is α. An occurrence of a variable in a formula that is within the scope of a quantifier is *bound*; otherwise, it is *free*. A *sentence* is a formula that contains no free occurrences of variables.

For example, suppose we specify a language with the predicate symbols *Walk* (arity 3), *Happy* (arity 1), and *Sad* (arity 1); the function symbol *Cos* (arity 1); the constants *Lisa*, *Kitchen*, *LivingRoom*, 0, and 1; and the variable a. Then 0, $Cos(0)$, and $Cos(Cos(0))$ are terms; $Walk(Lisa, Kitchen, LivingRoom)$ and $Cos(0) = 1$ are atoms; and $Happy(a) \Rightarrow \neg Sad(a)$ is a formula. The following are sentences:

$$Walk(Lisa, Kitchen, LivingRoom)$$
$$Cos(0) = 1$$
$$\forall a\, (Happy(a) \Rightarrow \neg Sad(a))$$

2.1.2 Semantics of First-Order Logic

The semantics of a language $\mathcal{L}$ of first-order logic defines the meaning of a formula of $\mathcal{L}$ as the set of models of the formula or the set of structures in which the formula is true. For example, one model of the formula $P(A, B) \wedge P(B, C)$ is the structure consisting of the following:

- the domain $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$
- the mapping from constants to elements of the domain $A \mapsto \mathbf{A}$, $B \mapsto \mathbf{B}$, $C \mapsto \mathbf{C}$
- the mapping from predicate symbols to relations $P \mapsto \{\langle \mathbf{A}, \mathbf{B} \rangle, \langle \mathbf{B}, \mathbf{C} \rangle\}$

The set $\{\langle \mathbf{A}, \mathbf{B} \rangle, \langle \mathbf{B}, \mathbf{C} \rangle\}$ is called the *extension* of P in the structure. If a formula π is true in all models of a formula ψ, then we write $\psi \models \pi$ and say that ψ *entails* π.

2.1.3 Proof Theory

The *proof theory* defines a proof of a formula π given a formula ψ as a sequence of formulas such that

- The last formula is π.
- Each formula is either a logical axiom, ψ, or a formula derived from previous formulas using an inference rule.

An example of an inference rule is *modus ponens*, which states that, from α and $\alpha \Rightarrow \beta$, we may derive β. If a proof of a formula π, given a formula ψ exists, then we write $\psi \vdash \pi$ and say that ψ is *provable* from π.

2.1.4 Many-Sorted First-Order Logic

The event calculus uses an extension of first-order logic called many-sorted first-order logic. We specify a set of sorts, and for each sort, a possibly empty set of subsorts. We specify the sort of each constant, variable, and function symbol and the sort of each argument position of each predicate and function symbol. Every term, atom, formula, and sentence must satisfy the sort specifications.

For example, suppose we specify the following. We have agent, person, and room sorts. The person sort is a subsort of the agent sort. The sort of *Lisa* is the person sort, and the sorts of *Kitchen* and *LivingRoom* are the room sort. The sort of the first argument position of *Walk* is the agent sort, and the sorts of the second and third argument positions of *Walk* are the room sort. Then the following is a formula and sentence:

$$Walk(Lisa, Kitchen, LivingRoom)$$

We assume a real number sort and appropriate functions and predicates such as *Plus* ($+$) and *LessThan* ($<$). See Appendix A for details.

2.1.5 Notational Conventions

In this book we use several notational conventions.

Case Conventions

Predicate symbols, function symbols, and nonnumeric constants start with an uppercase letter. Examples of predicate symbols are *Walk* and *InRoom*, examples of function symbols are *Distance* and *Cos*, and examples of constants are *Lisa*, *Nathan*, -4, 1, and π. Variables start with a lowercase letter. Examples of variables are a, b, b_1, and b_2.

Implicit Universal Quantification

Free variables are implicitly universally quantified. For example, $P(x, y, z) \wedge \exists u\, R(u, x)$ is an abbreviation for $\forall x, y, z\, (P(x, y, z) \wedge \exists u\, R(u, x))$.

Conjunctions and Disjunctions

The expression $\bigwedge_{i=1}^{n} \Gamma_i$ stands for $\Gamma_1 \wedge \cdots \wedge \Gamma_n$, and $\bigvee_{i=1}^{n} \Gamma_i$ stands for $\Gamma_1 \vee \cdots \vee \Gamma_n$.

Running Exclusive OR (XOR) Notation

The expression $\alpha_1 \dot{\vee} \alpha_2 \dot{\vee} \alpha_3$ means that exactly one of α_1, α_2, and α_3 is true, which is equivalent neither to $(\alpha_1 \dot{\vee} \alpha_2) \dot{\vee} \alpha_3$ nor to $\alpha_1 \dot{\vee} (\alpha_2 \dot{\vee} \alpha_3)$. In general, $\alpha_1 \dot{\vee} \cdots \dot{\vee} \alpha_n$ stands for the conjunction of $\alpha_1 \vee \cdots \vee \alpha_n$ and $\alpha_i \Rightarrow \neg\alpha_j$ for every $i, j \in \{1, \ldots, n\}$ such that $i \neq j$. Thus, $\alpha_1 \dot{\vee} \alpha_2 \dot{\vee} \alpha_3$ stands for

$$\begin{aligned}(\alpha_1 \vee \alpha_2 \vee \alpha_3) \wedge \\ (\alpha_1 \Rightarrow \neg\alpha_2) \wedge (\alpha_1 \Rightarrow \neg\alpha_3) \wedge \\ (\alpha_2 \Rightarrow \neg\alpha_1) \wedge (\alpha_2 \Rightarrow \neg\alpha_3) \wedge \\ (\alpha_3 \Rightarrow \neg\alpha_1) \wedge (\alpha_3 \Rightarrow \neg\alpha_2)\end{aligned}$$

Definitions of Abbreviations

The notation $\Gamma_1 \stackrel{\text{def}}{\equiv} \Gamma_2$ defines Γ_1 as an abbreviation for Γ_2. That is, $\Gamma_1 \stackrel{\text{def}}{\equiv} \Gamma_2$ means that all occurrences of the expression Γ_1 are to be replaced with the expression Γ_2.

2.2 Event Calculus Basics

This section discusses the sorts and predicates of the event calculus, and the possible states of a fluent.

2.2.1 Event Calculus Sorts

The event calculus uses the following sorts:

- an event sort, with variables e, e_1, e_2, ...
- a fluent sort, with variables f, f_1, f_2, ...
- a timepoint sort, which is a subsort of the real number sort, with variables $t, t_1, t_2, \ldots$

2.2.2 Event Calculus Predicates

The predicates of the event calculus are as follows.

Happens(e, t): Event e happens or occurs at timepoint t.

HoldsAt(f, t): Fluent f is true at timepoint t. If $\neg$*HoldsAt*(f, t), then we say that f is false at t.

ReleasedAt(f, t): Fluent f is released from the commonsense law of inertia at timepoint t. If $\neg$*ReleasedAt*(f, t), then we say that f is not released from the commonsense law of inertia at t. The commonsense law of inertia states that a fluent's truth value persists unless the fluent is affected by an event. When a fluent is released from this law, its truth value can fluctuate. We discuss the commonsense law of inertia in detail in Chapter 5.

Initiates(e, f, t): Event e initiates fluent f at timepoint t. If e occurs at t, then f will be true and not released from the commonsense law of inertia after t. If *Happens*(e, t) and *Initiates*(e, f, t), then we say that f is initiated by an event e that occurs at t.

Terminates(e, f, t): Event e terminates fluent f at timepoint t. If e occurs at t, then f will be false and not released from the commonsense law of inertia after t. If *Happens*(e, t) and *Terminates*(e, f, t), then we say that f is terminated by an event e that occurs at t.

Releases(e, f, t): Event e releases fluent f at timepoint t. If e occurs at t, then fluent f will be released from the commonsense law of inertia after t. If *Happens*(e, t) and *Releases*(e, f, t), then we say that f is released by an event e that occurs at t.

Trajectory(f_1, t_1, f_2, t_2): If fluent f_1 is initiated by an event that occurs at timepoint t_1, and $t_2 > 0$, then fluent f_2 will be true at timepoint $t_1 + t_2$.

AntiTrajectory(f_1, t_1, f_2, t_2): If fluent f_1 is terminated by an event that occurs at timepoint t_1, and $t_2 > 0$, then fluent f_2 will be true at timepoint $t_1 + t_2$.

2.2.3 States of a Fluent

In any given model, a fluent f can be in one of the following four states at a timepoint t:

true and released $HoldsAt(f, t) \wedge ReleasedAt(f, t)$

true and not released $HoldsAt(f, t) \wedge \neg ReleasedAt(f, t)$

false and released $\neg HoldsAt(f, t) \wedge ReleasedAt(f, t)$

false and not released $\neg HoldsAt(f, t) \wedge \neg ReleasedAt(f, t)$

2.3 Event Calculus Axiomatizations

This section presents and describes two axiomatizations: EC and DEC. We give a short explanation of each axiom or definition as it is presented. Much of the remainder of the book is devoted to exploring the behavior and uses of these axioms and definitions in detail.

2.3.1 The (Continuous) Event Calculus

EC consists of 17 axioms and definitions. We divide them into several groups.

Clipped, Declipped, Stopped, and Started

Some abbreviations relating to the initiation and termination of fluents are defined.

DEFINITION EC1

A fluent is ***clipped*** between timepoints t_1 and t_2 if and only if the fluent is terminated by some event that occurs at or after t_1 and before t_2.

$$Clipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\,(Happens(e, t) \wedge t_1 \leq t < t_2 \wedge Terminates(e, f, t))$$

DEFINITION EC2

A fluent is ***declipped*** between timepoints t_1 and t_2 if and only if the fluent is initiated by some event that occurs at or after t_1 and before t_2.

$$Declipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\,(Happens(e, t) \wedge t_1 \leq t < t_2 \wedge Initiates(e, f, t))$$

DEFINITION EC3

A fluent is ***stopped*** between timepoints t_1 and t_2 if and only if the fluent is terminated by some event that occurs after t_1 and before t_2.

$$StoppedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\,(Happens(e, t) \wedge t_1 < t < t_2 \wedge Terminates(e, f, t))$$

DEFINITION EC4

A fluent is ***started*** between timepoints t_1 and t_2 if and only if the fluent is initiated by some event that occurs after t_1 and before t_2.

$$StartedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\,(Happens(e, t) \wedge t_1 < t < t_2 \wedge Initiates(e, f, t))$$

Trajectory and AntiTrajectory

The behavior of trajectories is defined.

AXIOM EC5

If $Trajectory(f_1, t_1, f_2, t_2)$, $0 < t_2$, f_1 is initiated by some event that occurs at t_1, and f_1 is not stopped between t_1 and $t_1 + t_2$, then f_2 is true at $t_1 + t_2$.

$$\begin{gathered}(Happens(e, t_1) \wedge Initiates(e, f_1, t_1) \wedge 0 < t_2 \wedge \\ Trajectory(f_1, t_1, f_2, t_2) \wedge \neg StoppedIn(t_1, f_1, t_1 + t_2)) \Rightarrow \\ HoldsAt(f_2, t_1 + t_2)\end{gathered}$$

AXIOM EC6

If $AntiTrajectory(f_1, t_1, f_2, t_2)$, $0 < t_2$, f_1 is terminated by some event that occurs at t_1, and f_1 is not started between t_1 and $t_1 + t_2$, then f_2 is true at $t_1 + t_2$.

$$(Happens(e, t_1) \wedge Terminates(e, f_1, t_1) \wedge 0 < t_2 \wedge AntiTrajectory(f_1, t_1, f_2, t_2) \wedge \neg StartedIn(t_1, f_1, t_1 + t_2)) \Rightarrow HoldsAt(f_2, t_1 + t_2)$$

Inertia of HoldsAt

The commonsense law of inertia is enforced for the truth values of fluents.

DEFINITION EC7

A fluent ***persists*** between timepoints t_1 and t_2 if and only if the fluent is not released from the commonsense law of inertia at any timepoint after t_1 and at or before t_2.

$$PersistsBetween(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \neg\exists t\,(ReleasedAt(f, t) \wedge t_1 < t \leq t_2)$$

DEFINITION EC8

A fluent is ***released*** between timepoints t_1 (inclusive) and t_2 if and only if the fluent is released by some event that occurs at or after t_1 and before t_2.

$$ReleasedBetween(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\,(Happens(e, t) \wedge t_1 \leq t < t_2 \wedge Releases(e, f, t))$$

AXIOM EC9

If a fluent is true at timepoint t_1 and the fluent persists and is not clipped between t_1 and some later timepoint t_2, then the fluent is true at t_2.

$$(HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge PersistsBetween(t_1, f, t_2) \wedge \neg Clipped(t_1, f, t_2)) \Rightarrow HoldsAt(f, t_2)$$

AXIOM EC10

If a fluent is false at timepoint t_1 and the fluent persists and is not declipped between t_1 and some later timepoint t_2, then the fluent is false at t_2.

$$(\neg HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge PersistsBetween(t_1, f, t_2) \wedge \neg Declipped(t_1, f, t_2)) \Rightarrow \neg HoldsAt(f, t_2)$$

Inertia of ReleasedAt

A form of inertia is also enforced for *ReleasedAt*.

AXIOM EC11

If a fluent is released from the commonsense law of inertia at timepoint t_1 and the fluent is neither clipped nor declipped between t_1 and some later timepoint t_2, then the fluent is released from the commonsense law of inertia at t_2.

$$(ReleasedAt(f, t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1, f, t_2) \wedge \neg Declipped(t_1, f, t_2)) \Rightarrow ReleasedAt(f, t_2)$$

AXIOM EC12

If a fluent is not released from the commonsense law of inertia at timepoint t_1 and the fluent is not released between t_1 (inclusive) and some later timepoint t_2, then the fluent is not released from the commonsense law of inertia at t_2.

$$(\neg ReleasedAt(f, t_1) \wedge t_1 < t_2 \wedge \neg ReleasedBetween(t_1, f, t_2)) \Rightarrow \neg ReleasedAt(f, t_2)$$

Influence of Events on Fluents

Event occurrences influence the states of fluents.

DEFINITION EC13

A fluent is ***released*** between timepoints t_1 (not inclusive) and t_2 if and only if the fluent is released by some event that occurs after t_1 and before t_2.

$$ReleasedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\, (Happens(e, t) \wedge t_1 < t < t_2 \wedge Releases(e, f, t))$$

AXIOM EC14

If a fluent is initiated by some event that occurs at timepoint t_1 and the fluent is neither stopped nor released between t_1 (not inclusive) and some later timepoint t_2, then the fluent is true at t_2.

$$(Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StoppedIn(t_1, f, t_2) \wedge \neg ReleasedIn(t_1, f, t_2)) \Rightarrow HoldsAt(f, t_2)$$

AXIOM EC15

If a fluent is terminated by some event that occurs at timepoint t_1 and the fluent is neither started nor released between t_1 (not inclusive) and some later timepoint t_2, then the fluent is false at t_2.

$$(Happens(e, t_1) \wedge Terminates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StartedIn(t_1, f, t_2) \wedge \neg ReleasedIn(t_1, f, t_2)) \Rightarrow \neg HoldsAt(f, t_2)$$

AXIOM EC16

If a fluent is released by some event that occurs at timepoint t_1 and the fluent is neither stopped nor started between t_1 and some later timepoint t_2, then the fluent is released from the commonsense law of inertia at t_2.

$$(Happens(e, t_1) \wedge Releases(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StoppedIn(t_1, f, t_2) \wedge \neg StartedIn(t_1, f, t_2)) \Rightarrow ReleasedAt(f, t_2)$$

AXIOM EC17

If a fluent is initiated or terminated by some event that occurs at timepoint t_1 and the fluent is not released between t_1 (not inclusive) and some later timepoint t_2, then the fluent is not released from the commonsense law of inertia at t_2.

$$(Happens(e, t_1) \wedge (Initiates(e, f, t_1) \vee Terminates(e, f, t_1)) \wedge t_1 < t_2 \wedge \neg ReleasedIn(t_1, f, t_2)) \Rightarrow \neg ReleasedAt(f, t_2)$$

We use EC to mean the conjunction of the event calculus axioms EC5, EC6, EC9, EC10, EC11, EC12, EC14, EC15, EC16, and EC17. The definitions EC1, EC2, EC3, EC4, EC7, EC8, and EC13 are incorporated into EC.

Note that we distinguish between definitions and axioms. EC1, for example, is called a definition and not an axiom because *Clipped* is not a predicate symbol of the first-order language. EC1 merely defines $Clipped(t_1, f, t_2)$ as an abbreviation for $\exists e, t\,(Happens(e, t) \wedge t_1 \leq t < t_2 \wedge Terminates(e, f, t))$. An axiomatization that supports event occurrences with duration is presented in Appendix C.

2.3.2 The Discrete Event Calculus

DEC restricts the timepoint sort to the integers. It consists of 12 axioms and definitions.

Stopped and Started

The definitions of *StoppedIn* and *StartedIn* are the same as in EC.

DEFINITION DEC1

$$StoppedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\,(Happens(e, t) \wedge t_1 < t < t_2 \wedge Terminates(e, f, t))$$

DEFINITION DEC2

$$StartedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\,(Happens(e, t) \wedge t_1 < t < t_2 \wedge Initiates(e, f, t))$$

Trajectory and AntiTrajectory

The axioms for trajectories are the same as in EC.

AXIOM DEC3

$$(Happens(e, t_1) \wedge Initiates(e, f_1, t_1) \wedge 0 < t_2 \wedge Trajectory(f_1, t_1, f_2, t_2) \wedge \neg StoppedIn(t_1, f_1, t_1 + t_2)) \Rightarrow HoldsAt(f_2, t_1 + t_2)$$

AXIOM DEC4

$$(Happens(e, t_1) \wedge Terminates(e, f_1, t_1) \wedge 0 < t_2 \wedge AntiTrajectory(f_1, t_1, f_2, t_2) \wedge \neg StartedIn(t_1, f_1, t_1 + t_2)) \Rightarrow HoldsAt(f_2, t_1 + t_2)$$

Inertia of HoldsAt

The commonsense law of inertia is enforced for *HoldsAt*.

AXIOM DEC5

If a fluent is true at timepoint t, the fluent is not released from the commonsense law of inertia at $t + 1$, and the fluent is not terminated by any event that occurs at t, then the fluent is true at $t + 1$.

$$(HoldsAt(f, t) \wedge \neg ReleasedAt(f, t + 1) \wedge \neg \exists e\,(Happens(e, t) \wedge Terminates(e, f, t))) \Rightarrow HoldsAt(f, t + 1)$$

AXIOM DEC6

If a fluent is false at timepoint t, the fluent is not released from the commonsense law of inertia at $t + 1$, and the fluent is not initiated by any event that occurs at t, then the fluent is false at $t + 1$.

$$(\neg HoldsAt(f, t) \wedge \neg ReleasedAt(f, t + 1) \wedge \neg \exists e\,(Happens(e, t) \wedge Initiates(e, f, t))) \Rightarrow \neg HoldsAt(f, t + 1)$$

Inertia of ReleasedAt

Inertia is enforced for *ReleasedAt*.

AXIOM DEC7

If a fluent is released from the commonsense law of inertia at timepoint t and the fluent is neither initiated nor terminated by any event that occurs at t, then the fluent is released from the commonsense law of inertia at $t + 1$.

$$(ReleasedAt(f, t) \wedge \neg\exists e\,(Happens(e, t) \wedge (Initiates(e, f, t) \vee Terminates(e, f, t)))) \Rightarrow ReleasedAt(f, t + 1)$$

AXIOM DEC8

If a fluent is not released from the commonsense law of inertia at timepoint t and the fluent is not released by any event that occurs at t, then the fluent is not released from the commonsense law of inertia at $t + 1$.

$$(\neg ReleasedAt(f, t) \wedge \neg\exists e\,(Happens(e, t) \wedge Releases(e, f, t))) \Rightarrow \neg ReleasedAt(f, t + 1)$$

Influence of Events on Fluents

Event occurrences influence the states of fluents.

AXIOM DEC9

If a fluent is initiated by some event that occurs at timepoint t, then the fluent is true at $t + 1$.

$$(Happens(e, t) \wedge Initiates(e, f, t)) \Rightarrow HoldsAt(f, t + 1)$$

AXIOM DEC10

If a fluent is terminated by some event that occurs at timepoint t, then the fluent is false at $t + 1$.

$$(Happens(e, t) \wedge Terminates(e, f, t)) \Rightarrow \neg HoldsAt(f, t + 1)$$

AXIOM DEC11

If a fluent is released by some event that occurs at timepoint t, then the fluent is released from the commonsense law of inertia at $t + 1$.

$$(Happens(e, t) \wedge Releases(e, f, t)) \Rightarrow ReleasedAt(f, t + 1)$$

AXIOM DEC12

If a fluent is initiated or terminated by some event that occurs at timepoint t, then the fluent is not released from the commonsense law of inertia at $t + 1$.

$$(Happens(e, t) \wedge (Initiates(e, f, t) \vee Terminates(e, f, t))) \Rightarrow \neg ReleasedAt(f, t + 1)$$

We use DEC to mean the conjunction of the discrete event calculus axioms DEC3 through DEC12. The definitions DEC1 and DEC2 are incorporated into DEC. DEC is logically equivalent to EC if we restrict the timepoint sort to the integers. A proof of the equivalence of DEC and EC for integer time is provided in Appendix B.

2.3.3 Choosing between the Event Calculus and the Discrete Event Calculus

In this book we freely alternate between EC and DEC. We offer the following guidance for choosing between EC and DEC when we are attempting to solve a given commonsense reasoning problem:

- DEC can be used in most cases (as can EC).
- If integer time is sufficient to model the problem, then DEC can be used.
- If continuous time is required to model the problem, then EC must be used.
- If continuous change is being modeled, then EC must be used, although DEC can be used to model the discrete version of continuous change known as gradual change. For details, see Chapter 7.
- If the truth values of fluents are to be determined at every timepoint in $\{0, 1, 2, \ldots, n\}$ for some $n \geq 0$, then it is convenient to use DEC, which proceeds timepoint by timepoint except in the case of gradual change.
- If the truth values of fluents are to be determined only at a few timepoints in $\{0, 1, 2, \ldots, n\}$, then it is convenient to use EC, which facilitates the application of the commonsense law of inertia to a span of timepoints.

It should be pointed out that the discrete event calculus restricts the timepoint sort only to the integers. Other sorts are not restricted in this fashion. Notably, the real number sort may be used in DEC (but not in the Discrete Event Calculus Reasoner implementation of DEC).

2.4 Reification

In first-order logic, we express the atemporal proposition that Nathan is in the living room using an atom such as

$$InRoom(Nathan, LivingRoom)$$

In the event calculus, we wish to write a temporal proposition, such as

$$Holds(InRoom(Nathan, LivingRoom), 1) \tag{2.1}$$

But for (2.1) to be a formula of first-order logic, $InRoom(Nathan, LivingRoom)$ must be a term and not an atom.

We therefore use the technique of *reification*, which in general consists of making a formula of a first-order language $\mathcal{L}_1$ into a term of another first-order language $\mathcal{L}_2$. We make atoms of $\mathcal{L}_1$ into terms of $\mathcal{L}_2$. We treat a predicate symbol in $\mathcal{L}_1$ such as *InRoom* as a function symbol in $\mathcal{L}_2$ whose sort is fluent. Then in $\mathcal{L}_2$, $InRoom(Nathan, LivingRoom)$ is a term.

Similarly we would like to represent temporal propositions involving events such as

$$Happens(PickUp(Nathan, Glass), 1)$$

We treat *PickUp* as a function symbol whose sort is event. Then $PickUp(Nathan, Glass)$ is a term.

DEFINITION 2.1 A ***fluent term*** is a term whose sort is fluent, an ***event term*** is a term whose sort is event, and a ***timepoint term*** is a term whose sort is timepoint.

2.4.1 Unique Names Axioms

Consider the following event terms:

$$PickUp(Nathan, Glass)$$
$$SetDown(Ryan, Bowl)$$

Nothing says that these do not denote the same event. But we would like them to denote distinct events, because *PickUp* and *SetDown* represent different types of actions. Further, consider the following event terms:

$$PickUp(Nathan, Glass1)$$
$$PickUp(Nathan, Glass2)$$

Again, nothing says that these do not denote the same event. In fact, they *might* denote the same event if *Glass1* and *Glass2* denote the same physical object.

We formalize our intuitions about when fluent and event terms denote distinct objects using *unique names axioms*. These are defined using the following U notation.

DEFINITION 2.2 If $\phi_1, \ldots, \phi_k$ are function symbols, then $U[\phi_1, \ldots, \phi_k]$ is an abbreviation for the conjunction of the formulas

$$\phi_i(x_1, \ldots, x_m) \neq \phi_j(y_1, \ldots, y_n)$$

where m is the arity of ϕ_i, n is the arity of ϕ_j, and $x_1, \ldots, x_m$, and $y_1, \ldots, y_n$ are distinct variables such that the sort of x_l is the sort of the lth argument

position of ϕ_i and the sort of y_l is the sort of the lth argument position of ϕ_j, for each $1 \leq i < j \leq k$, and the conjunction of the formulas

$$\phi_i(x_1, \ldots, x_m) = \phi_i(y_1, \ldots, y_m) \Rightarrow x_1 = y_1 \wedge \cdots \wedge x_m = y_m$$

where m is the arity of ϕ_i and $x_1, \ldots, x_m$, and $y_1, \ldots, y_m$ are distinct variables such that the sort of x_l and y_l is the sort of the lth argument position of ϕ_i, for each $1 \leq i \leq k$.

DEFINITION 2.3 If $\phi_1, \ldots, \phi_n$ are function symbols, then $U[\phi_1, \ldots, \phi_n]$ is a ***unique names axiom***.

2.5 Conditions

We use conditions within many types of event calculus formulas such as action precondition axioms, effect axioms, state constraints, and trigger axioms.

DEFINITION 2.4 If τ_1 and τ_2 are terms, then $\tau_1 < \tau_2$, $\tau_1 \leq \tau_2$, $\tau_1 = \tau_2$, $\tau_1 \geq \tau_2$, $\tau_1 > \tau_2$, and $\tau_1 \neq \tau_2$ are ***comparisons***.

DEFINITION 2.5 A ***condition*** is defined inductively as follows:

- A comparison is a condition.
- If β is a fluent term and τ is a timepoint term, then $HoldsAt(\beta, \tau)$ and $\neg HoldsAt(\beta, \tau)$ are conditions.
- If γ_1 and γ_2 are conditions, then $\gamma_1 \wedge \gamma_2$ and $\gamma_1 \vee \gamma_2$ are conditions.
- If ν is a variable and γ is a condition, then $\exists \nu\, \gamma$ is a condition.
- Nothing else is a condition.

(Inductive definitions are explained in Appendix A.)

2.6 Circumscription

In the event calculus, we use logical formulas such as the following to describe the effects of events and the events that occur:

$$Initiates(SwitchOn, LightOn, t) \tag{2.2}$$

$$Terminates(SwitchOff, LightOn, t) \tag{2.3}$$
$$Happens(SwitchOn, 3) \tag{2.4}$$

But these formulas say nothing about which effects events do not have and about which events do not occur. For example, the following could also be the case:

$$Initiates(SwitchOn, WaterBoiling, t) \tag{2.5}$$
$$Happens(SwitchOff, 6) \tag{2.6}$$

As mentioned in Chapter 1, we must be able to assume by default that there are no such unexpected effects and that no such unexpected events occur. To do this, the event calculus uses a technique introduced by John McCarthy called circumscription. Specifically, the event calculus uses circumscription to minimize the extension of predicates such as *Happens*, *Initiates*, and *Terminates*.

The circumscription of *Initiates* in (2.2), written $CIRC[(2.2); Initiates]$, is

$$(e = SwitchOn \wedge f = LightOn) \Leftrightarrow Initiates(e, f, t) \tag{2.7}$$

$CIRC[(2.3); Terminates]$ is

$$(e = SwitchOff \wedge f = LightOn) \Leftrightarrow Terminates(e, f, t) \tag{2.8}$$

$CIRC[(2.4); Happens]$ is

$$(e = SwitchOn \wedge t = 3) \Leftrightarrow Happens(e, t) \tag{2.9}$$

A formal definition of circumscription is provided in Appendix A.

Nonmonotonic Reasoning

First-order logic entailment is *monotonic*: If $\psi \models \pi$, then $\psi \wedge \psi' \models \pi$ for every ψ'; also, if $\psi \vdash \pi$, then $\psi \wedge \psi' \vdash \pi$ for every ψ'. Circumscription allows us to perform *nonmonotonic reasoning*: If the circumscription of ψ entails π, then it is not necessarily the case that the circumscription of $\psi \wedge \psi'$ entails π.

Suppose that the only known event occurrence is given by (2.4). Then from (2.7), (2.8), (2.9), and other appropriate axioms, we can conclude that the light is on at timepoint 7:

$$HoldsAt(LightOn, 7)$$

Now suppose that, in addition to (2.4), the event (2.6) is known to occur. The circumscription $CIRC[(2.4) \wedge (2.6); Happens]$ is

$$(e = SwitchOn \wedge t = 3) \vee (e = SwitchOff \wedge t = 6) \Leftrightarrow Happens(e, t)$$

From this, (2.7), and (2.8), we can no longer conclude that the light is on at timepoint 7. In fact, we can conclude that the light is *not* on at that timepoint.

2.6.1 Computing Circumscription

In general, computing circumscription is difficult. In many cases, however, we can compute circumscription by applying the following two theorems. The first theorem provides a method for computing circumscription using predicate completion.

THEOREM 2.1

Let ρ be an n-ary predicate symbol and $\Delta(x_1, \ldots, x_n)$ be a formula whose only free variables are $x_1, \ldots, x_n$. If $\Delta(x_1, \ldots, x_n)$ does not contain ρ, then the basic circumscription $CIRC[\forall x_1, \ldots, x_n\ (\Delta(x_1, \ldots, x_n) \Rightarrow \rho(x_1, \ldots, x_n)); \rho]$ is equivalent to $\forall x_1, \ldots, x_n\ (\Delta(x_1, \ldots, x_n) \Leftrightarrow \rho(x_1, \ldots, x_n))$.

Proof See the proof of Proposition 2 of Lifschitz (1994). ■

Thus, we may compute circumscription of ρ in a formula by (1) rewriting the formula in the form

$$\forall x_1, \ldots, x_n\ (\Delta(x_1, \ldots, x_n) \Rightarrow \rho(x_1, \ldots, x_n))$$

where $\Delta(x_1, \ldots, x_n)$ does not contain ρ, and (2) applying Theorem 2.1.

The second theorem provides a method for computing parallel circumscription or the circumscription of several predicates. First a definition is required.

DEFINITION 2.6

A formula Δ is ***positive relative to a predicate symbol*** ρ if and only if all occurrences of ρ in Δ are in the range of an even number of negations in an equivalent formula obtained by eliminating $\Rightarrow$ and $\Leftrightarrow$ from Δ. We eliminate $\Rightarrow$ from a formula by replacing all instances of $(\alpha \Rightarrow \beta)$ with $(\neg\alpha \vee \beta)$. We eliminate $\Leftrightarrow$ from a formula by replacing all instances of $(\alpha \Leftrightarrow \beta)$ with $((\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha))$. (See Exercise 3.)

THEOREM 2.2

Let $\rho_1, \ldots, \rho_n$ be predicate symbols and Δ be a formula. If Δ is positive relative to every ρ_i, then the parallel circumscription $CIRC[\Delta; \rho_1, \ldots, \rho_n]$ is equivalent to the conjunction of the basic circumscriptions $\bigwedge_{i=1}^{n} CIRC[\Delta; \rho_i]$.

Proof See the proof of Proposition 14 of Lifschitz (1994). ■

2.6.2 Example: Circumscription of Happens

Let $\Delta = Happens(E1, T1) \wedge Happens(E2, T2)$. We compute $CIRC[\Delta; Happens]$ by rewriting Δ as the logically equivalent formula

$$(e = E1 \wedge t = T1) \vee (e = E2 \wedge t = T2) \Rightarrow Happens(e, t)$$

and then applying Theorem 2.1, which gives

$$(e = E1 \wedge t = T1) \vee (e = E2 \wedge t = T2) \Leftrightarrow Happens(e, t)$$

2.6.3 Example: Circumscription of Initiates

Let $\Sigma = Initiates(E1(x), F1(x), t) \wedge Initiates(E2(x,y), F2(x,y), t)$. We compute $CIRC[\Sigma; Initiates]$ by rewriting Σ as

$$\begin{gathered}\exists x\,(e = E1(x) \wedge f = F1(x)) \vee \\ \exists x, y\,(e = E2(x,y) \wedge f = F2(x,y)) \Rightarrow \\ Initiates(e, f, t)\end{gathered}$$

and then applying Theorem 2.1, which gives

$$\begin{gathered}\exists x\,(e = E1(x) \wedge f = F1(x)) \vee \\ \exists x, y\,(e = E2(x,y) \wedge f = F2(x,y)) \Leftrightarrow \\ Initiates(e, f, t)\end{gathered}$$

Numerous examples of computing circumscription are given in the remainder of this book.

2.7 Domain Descriptions

We solve commonsense reasoning problems by creating an event calculus domain description. A *domain description* consists of the following:

- a collection of axioms describing the commonsense domain or domains of interest, called an *axiomatization*
- observations of world properties at various times
- a narrative of known world events

In general, we should aim for elaboration-tolerant axiomatizations that can easily be extended to handle new scenarios or phenomena. John McCarthy defines a formalism as *elaboration tolerant* to the degree that it is easy to extend knowledge represented using the formalism. Elaboration-tolerant formalisms allow an axiomatization to be extended through the addition of new axioms rather than by performing surgery on existing axioms. Circumscription helps provide elaboration tolerance in the event calculus because it enables the incorporation of new event effects and occurrences by adding axioms. The types of axioms we may use to describe commonsense domains are summarized in Table 2.1 and described in detail in the rest of this book.

Observations consist of *HoldsAt* and *ReleasedAt* formulas.

Table 2.1 Event calculus formulas used for commonsense reasoning[a]

Formula	*Form*	*Defined in*
Unique names axiom [Ω]	$U[\phi_1, \ldots, \phi_n]$	Section 2.4.1
Observation [Γ]	$HoldsAt(\beta, \tau)$ or $ReleasedAt(\beta, \tau)$	Section 2.7
Event occurrence formula [Δ_1]	$Happens(\alpha, \tau)$	Section 2.7
Temporal ordering formula [Δ_1]	$\tau_1 < \tau_2$, $\tau_1 \leq \tau_2$, $\tau_1 = \tau_2$, ...	Section 2.7
Positive effect axiom [Σ]	$\gamma \Rightarrow Initiates(\alpha, \beta, \tau)$	Section 3.1
Negative effect axiom [Σ]	$\gamma \Rightarrow Terminates(\alpha, \beta, \tau)$	Section 3.1
Fluent precondition axiom [Σ]	$\gamma \Rightarrow \pi(\alpha, \beta, \tau)$	Section 3.3.1
Action precondition axiom [Ψ]	$Happens(\alpha, \tau) \Rightarrow \gamma$	Section 3.3.2
State constraint [Ψ]	γ_1, $\gamma_1 \Rightarrow \gamma_2$, or $\gamma_1 \Leftrightarrow \gamma_2$	Section 3.4
Trigger axiom [Δ_2]	$\gamma \Rightarrow Happens(\alpha, \tau)$	Section 4.1
Release axiom [Σ]	$\gamma \Rightarrow Releases(\alpha, \beta, \tau)$	Section 5.3
Effect constraint [Σ]	$\gamma \wedge \pi_1(\alpha, \beta_1, \tau) \Rightarrow \pi_2(\alpha, \beta_2, \tau)$	Section 6.4
Causal constraint [Δ_2]	$\sigma(\beta, \tau) \wedge \pi_1(\beta_1, \tau) \wedge \cdots \wedge \pi_n(\beta_n, \tau) \Rightarrow Happens(\beta, \tau)$	Section 6.5
Trajectory axiom [Π]	$\gamma \Rightarrow Trajectory(\beta_1, \tau_1, \beta_2, \tau_2)$	Section 7.1
Antitrajectory axiom [Π]	$\gamma \Rightarrow AntiTrajectory(\beta_1, \tau_1, \beta_2, \tau_2)$	Section 7.2
Event occurrence constraint [Ψ]	$Happens(\alpha_1, \tau) \wedge \gamma \Rightarrow (\neg)Happens(\alpha_2, \tau)$	Section 8.1.2
Positive cumulative effect axiom [Σ]	$\gamma \wedge Happens(\alpha_1, \tau) \wedge \cdots \wedge Happens(\alpha_n, \tau) \Rightarrow Initiates(\alpha, \beta, \tau)$	Section 8.2
Negative cumulative effect axiom [Σ]	$\gamma \wedge Happens(\alpha_1, \tau) \wedge \cdots \wedge Happens(\alpha_n, \tau) \Rightarrow Terminates(\alpha, \beta, \tau)$	Section 8.2
Disjunctive event axiom [Δ_2]	$Happens(\alpha, \tau) \Rightarrow Happens(\alpha_1, \tau) \vee \cdots \vee Happens(\alpha_n, \tau)$	Section 9.2
Cancellation axiom [Θ]	$\gamma \Rightarrow Ab_i(\ldots, \tau)$	Section 12.3

[a] $\alpha, \alpha_1, \ldots, \alpha_n$ = event terms; $\beta, \beta_1, \ldots, \beta_n$ = fluent terms, $\gamma, \gamma_1, \gamma_2$ = conditions, $\pi, \pi_1, \ldots, \pi_n$ = *Initiates* or *Terminates*, σ = *Stopped* or *Started*, τ, τ_1, τ_2 = timepoint terms, $\phi_1, \ldots, \phi_n$ = function symbols.

DEFINITION 2.7 If β is a fluent term and τ is a timepoint term, then $HoldsAt(\beta, \tau)$ and $ReleasedAt(\beta, \tau)$ are ***observations***.

A narrative consists of event occurrence formulas and temporal ordering formulas that constrain their order.

DEFINITION 2.8 If α is an event term and τ is a timepoint term, then $Happens(\alpha, \tau)$ is an ***event occurrence formula***.

DEFINITION 2.9 If τ_1 and τ_2 are timepoint terms, then $\tau_1 < \tau_2$, $\tau_1 \leq \tau_2$, $\tau_1 = \tau_2$, $\tau_1 \geq \tau_2$, $\tau_1 > \tau_2$, and $\tau_1 \neq \tau_2$ are ***timepoint comparisons***.

DEFINITION 2.10 A ***temporal ordering formula*** is a conjunction of timepoint comparisons.

We now define a domain description.

DEFINITION 2.11 An event calculus ***domain description*** is given by

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge$$
$$CIRC[\Delta_1 \wedge \Delta_2; Happens] \wedge$$
$$CIRC[\Theta; Ab_1, \ldots, Ab_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge \mathrm{E}$$

where

- Σ is a conjunction of positive effect axioms, negative effect axioms, release axioms, effect constraints, positive cumulative effect axioms, and negative cumulative effect axioms.
- Δ_1 is a conjunction of event occurrence formulas and temporal ordering formulas (the narrative).
- Δ_2 is a conjunction of trigger axioms, causal constraints, and disjunctive event axioms.
- Θ is a conjunction of cancellation axioms containing $Ab_1, \ldots, Ab_n$.
- Ω is a conjunction of unique names axioms.
- Ψ is a conjunction of state constraints, action precondition axioms, and event occurrence constraints.
- Π is a conjunction of trajectory axioms and antitrajectory axioms.
- Γ is a conjunction of observations.
- E is a conjunction of event calculus axioms such as EC or DEC.

The axiomatization consists of Σ, Δ_2, Θ, Ω, Ψ, Π, and E, the observations consist of Γ, and the narrative consists of Δ_1. We write Δ to mean the conjunction of Δ_1 and Δ_2.

Event calculus reasoning is nonmonotonic, because event calculus domain descriptions make use of circumscription.

2.7.1 Example: Sleep

Let us revisit the example in Section 1.4.2. We use an agent sort with the variable a. The event $WakeUp(a)$ represents that agent a wakes up, the event $FallAsleep(a)$ represents that agent a falls asleep, and the fluent $Awake(a)$ represents that agent a is awake.

Our axiomatization consists of several axioms. We use a positive effect axiom to represent that, if an agent wakes up, then the agent will be awake:

$$Initiates(WakeUp(a), Awake(a), t) \tag{2.10}$$

We use a negative effect axiom to represent that, if an agent falls asleep, then the agent will no longer be awake:

$$Terminates(FallAsleep(a), Awake(a), t) \tag{2.11}$$

We have the following observations and narrative. At timepoint 0, Nathan is not awake and this fact is subject to the commonsense law of inertia:

$$\neg HoldsAt(Awake(Nathan), 0) \tag{2.12}$$
$$\neg ReleasedAt(Awake(Nathan), 0) \tag{2.13}$$

We have a narrative consisting of a single event occurrence. At timepoint 1, Nathan wakes up:

$$Happens(WakeUp(Nathan), 1) \tag{2.14}$$

Given these axioms and the conjunction of axioms EC, we can show that Nathan will be awake at, say, timepoint 3. The Halmos symbol ■ is used to indicate the end of a proof.

PROPOSITION 2.1 Let $\Sigma = (2.10) \wedge (2.11)$, $\Delta = (2.14)$, $\Omega = U[WakeUp, FallAsleep]$, and $\Gamma = (2.12) \wedge (2.13)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Gamma \wedge \mathrm{EC} \models HoldsAt(Awake(Nathan), 3)$$

Proof From $CIRC[\Sigma; Initiates, Terminates, Releases]$, Theorem 2.2, and Theorem 2.1, we have

$$(Initiates(e,f,t) \Leftrightarrow \exists a\,(e = WakeUp(a) \wedge f = Awake(a))) \wedge \quad (2.15)$$
$$(Terminates(e,f,t) \Leftrightarrow \exists a\,(e = FallAsleep(a) \wedge f = Awake(a))) \wedge$$
$$\neg Releases(e,f,t)$$

From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have

$$Happens(e,t) \Leftrightarrow (e = WakeUp(Nathan) \wedge t = 1) \quad (2.16)$$

From (2.16) and EC3, we have $\neg StoppedIn(1, Awake(Nathan), 3)$. From (2.16) and EC13, we have $\neg ReleasedIn(1, Awake(Nathan), 3)$. From (2.16), (2.15), $1 < 3$, $\neg StoppedIn(1, Awake(Nathan), 3)$, $\neg ReleasedIn(1, Awake(Nathan), 3)$, and EC14, we have $HoldsAt(Awake(Nathan), 3)$. ■

We can also show that Nathan will be awake at timepoint 3 using the conjunction of axioms DEC.

PROPOSITION 2.2 Let $\Sigma = (2.10) \wedge (2.11)$, $\Delta = (2.14)$, $\Omega = U[WakeUp, FallAsleep]$, and $\Gamma = (2.12) \wedge (2.13)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge$$
$$\Omega \wedge \Gamma \wedge \text{DEC} \models HoldsAt(Awake(Nathan), 3).$$

Proof From $CIRC[\Sigma; Initiates, Terminates, Releases]$, Theorem 2.2, and Theorem 2.1, we have

$$(Initiates(e,f,t) \Leftrightarrow \exists a\,(e = WakeUp(a) \wedge f = Awake(a))) \wedge \quad (2.17)$$
$$(Terminates(e,f,t) \Leftrightarrow \exists a\,(e = FallAsleep(a) \wedge f = Awake(a))) \wedge$$
$$\neg Releases(e,f,t)$$

From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have

$$(e = WakeUp(Nathan) \wedge t = 1) \Leftrightarrow Happens(e,t) \quad (2.18)$$

Using DEC, we proceed one timepoint at a time. From (2.17), we have $\neg\exists e\,(Happens(e,0) \wedge Releases(e, Awake(Nathan), 0))$. From this, (2.13), and DEC8, we have

$$\neg ReleasedAt(Awake(Nathan), 1) \quad (2.19)$$

From (2.17), (2.18), and DEC9, we have

$$HoldsAt(Awake(Nathan), 2) \tag{2.20}$$

From (2.17), we have $\neg\exists e\,(Happens(e, 1) \wedge Releases(e, Awake(Nathan), 1))$. From this, (2.19), and DEC8, we have

$$\neg ReleasedAt(Awake(Nathan), 2) \tag{2.21}$$

From (2.18), we have $\neg\exists e\,(Happens(e, 2) \wedge Terminates(e, Awake(Nathan), 2))$. From this, (2.20), (2.21), and DEC5, we have $HoldsAt(Awake(Nathan), 3)$. ■

2.7.2 Inconsistency

Inconsistency can arise in a domain description in a number of ways. Here are some common ways in which inconsistency can arise using the EC and DEC axioms.

Simultaneously initiating and terminating a fluent produces inconsistency. For example, given

$$Initiates(E(o), F(o), t)$$
$$Terminates(E(o), F(o), t)$$
$$Happens(E(O1), 0)$$

we can show both $HoldsAt(F(O1), 1)$ and $\neg HoldsAt(F(O1), 1)$.

Simultaneously releasing and initiating or terminating a fluent produces inconsistency. For example, from

$$Releases(E(o), F(o), t)$$
$$Initiates(E(o), F(o), t)$$
$$Happens(E(O1), 0)$$

we can show both $ReleasedAt(F(O1), 1)$ and $\neg ReleasedAt(F(O1), 1)$.

Inconsistency can arise from the use of effect axioms and state constraints. For example, given

$$Initiates(E(o), F1(o), t) \tag{2.22}$$
$$HoldsAt(F1(o), t) \Leftrightarrow HoldsAt(F2(o), t) \tag{2.23}$$
$$\neg HoldsAt(F2(O1), 0) \tag{2.24}$$
$$\neg ReleasedAt(F2(O1), 1) \tag{2.25}$$
$$Happens(E(O1), 0) \tag{2.26}$$

we can show both $\neg HoldsAt(F2(O1), 1)$, from (2.24), (2.25), and other appropriate axioms, and $HoldsAt(F2(O1), 1)$, from (2.26), (2.22), (2.23), and other appropriate axioms.

2.8 Reasoning Types

Several types of reasoning may be performed using the event calculus, such as deduction, abduction, postdiction, and model finding. The Discrete Event Calculus Reasoner program discussed in Chapter 13 can perform all these reasoning types. For the purposes of the definitions in this section, let Σ, Δ_1, Δ_2, Θ, Ω, Ψ, Π, Γ, and E be formulas as defined in Section 2.7, Δ be a conjunction of Δ_1 and Δ_2, and Γ' be a conjunction of observations.

2.8.1 Deduction and Temporal Projection

Deduction or *temporal projection* consists of determining the state that results from performing a sequence of actions. For example, given that a fan is set on a table and turned on, deduction or temporal projection determines that the fan is on the table and turning.

DEFINITION 2.12

A ***deduction*** or ***temporal projection problem*** consists of determining whether it is the case that

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge CIRC[\Theta; Ab_1, \ldots, Ab_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge \mathrm{E} \models \Gamma'$$

Propositions 2.1 and 2.2 are deduction problems. From the fact that Nathan woke up, we can deduce that afterward he was awake. Deduction can be performed by the Discrete Event Calculus Reasoner as discussed in Section 13.3.1.

2.8.2 Abduction and Planning

Abduction consists of determining what events might lead from an initial state to a final state. Similarly, *planning* consist of generating a sequence of actions to bring about a final state, called a *goal*, starting from an initial state. Suppose that the initial state is that Nathan is in his bedroom on the second floor and that the goal or final state is that Nathan is outside his house. Abduction or planning will produce a sequence of actions in which Nathan walks out of his bedroom, walks downstairs, opens the front door, and walks outside.

DEFINITION 2.13

Let $\Phi =$

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta_1 \wedge \Delta_2; Happens] \wedge CIRC[\Theta; Ab_1, \ldots, Ab_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge \mathrm{E}$$

An ***abduction*** or ***planning problem*** consists of taking Σ, Δ_2, Θ, Ω, Ψ, Π, Γ, Γ' (the goal) and E as input and producing as output zero or more Δ_1 (called ***plans*** for Γ') such that Φ is consistent and $\Phi \models \Gamma'$.

2.8.3 Example: Sleep Abduction

Given that Nathan was not awake and then he was awake, we can abduce that he woke up. Abduction can be performed using Shanahan's event calculus planner, which is written in Prolog. We create a file called `sleep.pl` containing the following:

```
axiom(initiates(wake_up(X),awake(X),T),[]).

axiom(terminates(fall_asleep(X),awake(Y),T),[]).

axiom(initially(neg(awake(nathan))),[]).

abducible(dummy).

executable(wake_up(X)).

executable(fall_asleep(X)).
```

We start Prolog and load the event calculus planner and `sleep.pl`:

```
Welcome to SWI-Prolog (Version 5.0.10)

Copyright (c) 1990-2002 University of Amsterdam.

SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
software, and you are welcome to redistribute it under certain
conditions.

Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- load_files(['gensym.pl','planner42.txt','sleep.pl']).
```

We then issue the following query:

```
?- abdemo([holds_at(awake(nathan),t)],R).
```

The event calculus planner produces the following plan:

```
R = [[happens(wake_up(nathan), t1, t1)], [before(t1, t)]]

Yes

?-
```

Abduction can also be performed by the Discrete Event Calculus Reasoner as discussed in Section 13.3.2.

2.8.4 Postdiction

Postdiction consists of determining the initial state given a sequence of events and a final state. For example, given that Nathan threw a ball and the ball was flying toward the wall, postdiction determines that Nathan was previously holding the ball.

DEFINITION 2.14

Let $\Phi =$

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge CIRC[\Theta; Ab_1, \ldots, Ab_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge \mathrm{E}$$

A ***postdiction problem*** consists of taking Σ, Δ, Θ, Ω, Ψ, Π, E, and Γ' as input, and producing as output zero or more Γ such that Φ is consistent and $\Phi \models \Gamma'$.

For example, postdiction allows us to determine that Nathan was asleep before he woke up. We add the following action precondition axiom:

$$Happens(WakeUp(a), t) \Rightarrow \neg HoldsAt(Awake(a), t) \tag{2.27}$$

We do not know whether Nathan was asleep at timepoint 1. We know that Nathan woke up at timepoint 1:

$$Happens(WakeUp(Nathan), 1)$$

From this and (2.27), we conclude that Nathan was asleep at timepoint 1:

$$\neg HoldsAt(Awake(Nathan), 1)$$

Postdiction can be performed by the Discrete Event Calculus Reasoner as discussed in Section 13.3.3.

2.8.5 Model Finding

Model finding consists of generating models from states and events. For example, given that Nathan went to sleep and sleeping occurs at home or at a hotel, model finding produces models in which Nathan sleeps at home and models in which Nathan sleeps at a hotel.

DEFINITION 2.15

A ***model finding problem*** consists of taking Σ, Δ, Θ, Ω, Ψ, Π, Γ, and E as input, and producing as output zero or more structures $\mathcal{M}$ such that

$$\mathcal{M} \models CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge CIRC[\Theta; Ab_1, \ldots, Ab_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge E$$

Model finding can be performed by the Discrete Event Calculus Reasoner as discussed in Section 13.3.4.

Bibliographic Notes

Event Calculus Basics and Axioms

McCarthy (1963) defined a *fluent* as "a predicate or function whose argument is a situation." He borrowed the term from Newton (1670/1969, pp. 72–73), who used it to mean a time-varying quantity. See Guicciardini's (1999) description of Newton's fluents and fluxions. The running exclusive or (xor) notation is from Hayes (1985, p. 72) and E. Davis (1990, p. 32). Bibliographic notes for first-order logic are provided in Appendix A.

The conjunction of axioms EC and associated definitions are taken from a paper by R. Miller and Shanahan (1999), which was later revised and expanded (R. Miller and Shanahan, 2002). The conjunction of axioms EC consists of axioms and definitions from the following sections of both versions of the paper:

- Section 2, which provides the basic classical logic axiomatization of the event calculus
- Section 3.2, which revises the axioms and definitions of Section 2 for a version of the event calculus in which initiating and terminating a fluent at the same time produces inconsistency
- Section 3.5, which adds axioms to those of Section 2 to support gradual change
- Section 3.7, which revises the axioms and definitions of Section 2 for a version of the event calculus in which fluents may be released from the commonsense law of inertia

The conjunction of axioms DEC was introduced by Mueller (2004a). R. Miller and Shanahan (2002, p. 452) provide a list of the many formulations of the event calculus in forms of logic programming, in classical logic, in modal logic, and as action languages.

Figures 2.1 and 2.2 show the evolution of the classical logic event calculus. The forced separation version of the event calculus is the first version that largely resembles the version used in this book. It is called forced separation because the axioms of the event calculus are outside the scope of any circumscription, a technique that can be traced back to the filtered preferential entailment or filtering of Sandewall (1989b, 1994). Table 2.2 shows the evolution of the axiom that determines when a fluent is true.

Reification

McCarthy (1987) defines reification as "making objects out of sentences and other entities" (p. 1034). McCarthy (1979) introduces reification in order to reason about knowledge and belief in first-order logic. He introduces terms to represent concepts such as "'*Mike's telephone number*' in the sentence '*Pat knows Mike's telephone number*'" (p. 129). He uses the convention that concept symbols start with an uppercase letter; thus, *Mike* represents the concept of Mike, *mike* represents Mike himself, *Telephone*(*Mike*) represents the concept of Mike's telephone number, *telephone*(*mike*) represents Mike's telephone number itself, *know*(*pat*, *Telephone*(*Mike*)) represents that Pat knows Mike's telephone number, and *dial*(*pat*, *telephone*(*mike*)) represents that Pat dials Mike's telephone number. The notion of reification was present in the original situation calculus of McCarthy and Hayes (1969). The reified notation $raining(x)(s)$ and the nonreified notation $raining(x, s)$ are both given as ways of representing that the fluent $raining(x)$ is true in situation s (p. 478). The action notation $opens(sf, k)$ is used within an axiom schema (p. 480). Hayes (1971, p. 511) considers symbols such as MOVE and CLIMB to represent functions that return actions. Kowalski (1979) introduces into the situation calculus the notation $Holds(f, s)$ (p. 134), which represents that fluent f is true in situation f.

Lifschitz (1987a, pp. 48–50) introduces techniques for reification in the situation calculus. These techniques are incorporated into the event calculus by Shanahan (1995a, p. 257; 1997b, pp. 37–38, 58). Lifschitz uses a many-sorted logic with sorts for actions, fluents, situations, and truth values. He uses the predicate $holds(f, s)$ (p. 39) to represent that fluent f is true in situation s. He introduces fluent function symbols to represent functions that return fluents, and action function symbols to represent functions that return actions. For example, $paint(Block_1, Blue)$ is an action term in which *paint* is an action function symbol (p. 48).

Following Reiter's (1980a) introduction of "axioms specifying that all constants are pairwise distinct" (p. 248), and a proposal of McCarthy (1986, pp. 96–97), Lifschitz (1987a, pp. 44, 50) introduces the U notation for defining unique names axioms. A system that assumes unique names axioms is said to use the unique

1. Original logic programming version of the event calculus (Kowalski and Sergot, 1986)
2. Simplified version of the event calculus (Kowalski, 1986, 1992); introduces *Happens*
3. Simplified version of the event calculus (Eshghi, 1988)
4. Simplified event calculus (Shanahan, 1989)
5. Simplified event calculus (Shanahan, 1990, sec. 1)
    (a) Continuous change (sec. 5)
6. Simplified event calculus (Shanahan, 1997b, sec. 13.1)
    (a) Continuous change (sec. 13.3)
7. Circumscriptive event calculus (Shanahan, 1995a, sec. 3; Shanahan, 1997b, sec. 14.3); first classical logic version of the event calculus
    (a) State constraints and ramifications (Shanahan, 1995a, sec. 7; Shanahan, 1997b, sec. 15.1)
    (b) Nondeterministic effects (Shanahan, 1995a, sec. 8; Shanahan, 1997b, sec. 15.2)
    (c) Release from commonsense law of inertia (Shanahan, 1997b, sec. 15.3)
    (d) Concurrent events (Shanahan, 1995a, sec. 9; Shanahan, 1997b, sec. 15.4)
    (e) Continuous change (Shanahan, 1995a, sec 10; Shanahan, 1997b, sec. 15.5)
8. Event calculus using forced separation (Shanahan, 1996, sec. 2); includes *Releases* but no $Initially_N$
    (a) Continuous change (sec. 3)
9. Event calculus using forced separation (R. Miller and Shanahan, 1996, sec. 3); includes *Releases*, *InitialisedTrue*, and *InitialisedFalse*
    (a) Continuous change described using differential equations (sec. 3); treats continuously varying parameters and discontinuities
10. Event calculus using forced separation (Shanahan, 1997b, sec. 16.3)
    (a) State constraints and ramifications (pp. 323–325)
    (b) Continuous change (sec. 16.4)

Figure 2.1 Evolution of the classical logic event calculus (1986–1997).

names assumption (Genesereth and Nilsson, 1987, p. 120). Pirri and Reiter (1999) provide a rich formalization of the situation calculus. They use a many-sorted second-order logic (p. 326), action function symbols to represent functions that return actions (p. 327), and unique names axioms for action function symbols (p. 331). They use a nonreified notation for fluents (p. 327).

1. Event calculus (Shanahan, 1998a, 2004); slightly different set of axioms; equivalent to event calculus using forced separation with continuous change
2. Event calculus (Shanahan, 1999b); makes extensions to the forced separation event calculus to deal with the ramification problem
    (a) State constraints (sec. 2)
    (b) Effect constraints (sec. 3)
    (c) Causal constraints (sec. 4); adds *Initiated*, *Terminated*, *Started*, and *Stopped*
3. Full event calculus (Shanahan, 1999a, sec. 3.1); forced separation version with three-argument *Happens* (R. Miller and Shanahan, 1994) to represent event occurrences with duration
    (a) Causal constraints (Shanahan, 1999a, sec. 4.2)
    (b) Concurrent events and continuous change (sec. 5); adds *Cancels*, *Cancelled*, and *Trajectory* predicates
4. Classical logic event calculus (R. Miller and Shanahan, 1999); forced separation version with more variations
    (a) Narrative information and planning (sec. 2.3)
    (b) Nonnegative time (sec. 3.1)
    (c) Initiating and terminating a fluent at the same time produces inconsistency (sec. 3.2)
    (d) Action preconditions (sec. 3.3)
    (e) Frame and nonframe fluents (sec. 3.4)
    (f) Continuous change (sec. 3.5)
    (g) Event occurrences with duration (sec. 3.6)
    (h) Release from commonsense law of inertia (sec. 3.7)
    (i) Mathematical modeling (sec. 3.8)
5. Discrete event calculus (Mueller, 2004a)

Figure 2.2 Evolution of the classical logic event calculus (1998–2004).

In addition to the situation calculus and the event calculus, several other logics have been proposed that use reified fluents or events. McDermott (1982) introduces a temporal logic for reasoning about action in which (*T s p*) represents that fact *p* is true in state *s* (p. 108) and (*Occ s1 s2 e*) represents that event *e* occurs over the interval specified by *s1* and *s2* (p. 110). A fact such as (*ON A B*) denotes the set of states in which *A* is on *B*, and an event such as (*MOVE A B*) denotes the set of intervals over which *A* is moved to *B*. Shoham (1987, pp. 96–99;

Table 2.2 Evolution of the *HoldsAt* axiom

Version	*HoldsAt Axiom*
Original event calculus	$Holds(after(e\ u))\ if\ Initiates(e\ u)$
(Kowalski and Sergot, 1986)	
Event calculus	$HoldsAt(r\ n)\ if\ Happens(e)\ and$
(Kowalski, 1986, 1992)	$Initiates(e\ r)\ and\ e < n\ and$
	$not\ \exists e^*\ [Happens(e^*)\ and$
	$Terminates(e^*\ r)\ and\ e < e^*\ and\ e^* < n]$
Event calculus	$holds(f, e2) \leftarrow initiates(e1, f),$
(Eshghi, 1988)	$e1 < e2,\ persists(f, e1, e3),\ e2 \leq e3$
Simplified event calculus	$holds\text{-}at(P, T)\ if$
(Shanahan, 1989)	$happens(E)\ and\ E < T\ and$
	$initiates(E, P)\ and\ not\ clipped(E, P, T)$
Simplified event calculus	$holds\text{-}at(P, T2)\ if$
(Shanahan, 1990, sec. 1)	$happens(E)\ and$
	$time(E, T1)\ and\ T1 < T2\ and$
	$initiates(E, P)\ and\ not\ clipped(T1, P, T2)$
Simplified event calculus	$HoldsAt(f, t2) \leftarrow$
(Shanahan, 1997b, sec. 13.1)	$Happens(a, t1) \wedge$
	$Initiates(a, f, t1) \wedge t1 < t2 \wedge$
	$not\ Clipped(t1, f, t2)$
Circumscriptive event calculus	$HoldsAt(p, t) \leftrightarrow$
(Shanahan, 1995a, sec. 3)	$\forall s\,[State(t, s) \wedge HoldsIn(p, s)]$
Event calculus	$HoldsAt(f, t2) \leftarrow$
(Shanahan, 1996, sec. 2)	$Happens(a, t1) \wedge$
	$Initiates(a, f, t1) \wedge t1 < t2 \wedge$
	$\neg Clipped(t1, f, t2)$
Full event calculus	$HoldsAt(f, t3) \leftarrow$
(Shanahan, 1999a, sec. 3.1)	$Happens(a, t1, t2) \wedge$
	$Initiates(a, f, t1) \wedge t2 < t3 \wedge$
	$\neg Clipped(t1, f, t3)$
Discrete event calculus	$Happens(e, t) \wedge Initiates(e, f, t) \Rightarrow$
(Mueller, 2004a)	$HoldsAt(f, t + 1)$

1988, pp. 43–46) introduces a logic in which $TRUE(t_1, t_2, r(a_1, \ldots, a_n))$ represents that $r(a_1, \ldots, a_n)$ is true during the interval specified by t_1 and t_2. Bacchus, Tenenberg, and Koomen (1991, pp. 100–102) compare reified and nonreified temporal logics. Ma and Knight (2001) review the use of reification in several temporal logics.

Time Intervals

Allen (1983) introduces a temporal representation based on time intervals. He defines a number of relations that may hold between intervals of time, such as *before*, *equal*, *meets*, *overlaps*, *during*, *starts*, and *finishes*. Allen (1984) introduces an interval-based temporal logic in which $HOLDS(p, t)$ represents that property p is true during interval t (p. 128) and $OCCUR(e, t)$ represents that event e occurs over interval t (pp. 132–133). The property p may contain logical connectives and quantifiers (p. 130).

In the event calculus, fluents are true at timepoints rather than time intervals. Following Shoham (1987, p. 94) and E. Davis (1990, p. 150), we may define a time interval as the set of timepoints that fall in the interval. Let the function $Start(i)$ represent the start timepoint of interval i, the function $End(i)$ represent the end timepoint of interval i, and the predicate $In(t, i)$ represent that timepoint t falls in the interval i. We define In in terms of $Start$ and End:

$$In(t, i) \Leftrightarrow Start(i) < t < End(i)$$

We may then represent that a fluent f is true over an interval i:

$$In(t, i) \Rightarrow HoldsAt(f, t)$$

E. Davis (1990, p. 153) shows that Allen's (1983) relations over intervals can be defined in terms of $Start$ and End. For example:

$$Before(i_1, i_2) \Leftrightarrow End(i_1) < Start(i_2)$$
$$Equal(i_1, i_2) \Leftrightarrow (Start(i_1) = Start(i_2) \wedge End(i_1) = End(i_2))$$
$$Meets(i_1, i_2) \Leftrightarrow End(i_1) = Start(i_2)$$

Nonmonotonic Reasoning

Minsky (1974, p. 75) criticized the monotonicity of logic, which led (Minsky, 1991b, p. 378; Israel, 1985) to the development of nonmonotonic reasoning methods (Bobrow, 1980). Minsky (1974) used the word *monotonicity* after a suggestion from Vaughan R. Pratt (McDermott and Doyle, 1980, p. 44). Previously, Hayes (1973) called this the "extension property" (p. 46), and Minsky (1961) noted that "in a mathematical domain a theorem, once proved, remains true when one

2.9 Exercises

2.1 List at least five models of the formula $P(A, B) \wedge P(B, C)$.

2.2 Suppose $Awake(a)$ and $Asleep(a)$ are functions. What conjunction of formulas does the expression $U[Awake, Asleep]$ stand for?

2.3 Give some examples of formulas that are positive relative to a predicate Q and some that are not positive relative to Q.

2.4 Compute the circumscription of R in the conjunction of the following formulas:

$$P(x) \wedge Q(x) \Rightarrow R(x)$$
$$R(A)$$
$$R(B)$$

2.5 (Research Problem) Investigate the combination of the event calculus and the contexts of McCarthy (1987, 1993) and Guha (1992). Introduce a sort for contexts and a predicate $ist(c, f)$ that represents that formula f is true in context c. See also the proposal of F. Giunchiglia and Ghidini (1998).

2.6 (Research Problem) Investigate how contexts could be used to facilitate the use of multiple representations in the event calculus.

PART II

Commonsense Phenomena

This Page Intentionally Left Blank

CHAPTER 3

The Effects of Events

Events change the state of the world. When an event occurs, some properties of the world that are false become true and some properties of the world that are true become false. This chapter is concerned with the representation of the effects of events on world properties. We discuss positive and negative effect axioms, commonly used effect axiom idioms, preconditions, and state constraints.

3.1 Positive and Negative Effect Axioms

In the event calculus, the effects of events are described by two predicates. The predicate $Initiates(\alpha, \beta, \tau)$ represents that, if an event α occurs at timepoint τ, then fluent β will be true after τ. The predicate $Terminates(\alpha, \beta, \tau)$ represents that, if an event α occurs at timepoint τ, then fluent β will be false after τ. We represent the effects of events using effect axioms.

The changes produced by an event may depend on the context in which the event occurs; an event may have one effect in one context and another effect in another context. Therefore, effect axioms contain conditions representing contexts.

DEFINITION 3.1

If γ is a condition representing the context, α is an event term, β is a fluent term, and τ is a timepoint term, then

$$\gamma \Rightarrow Initiates(\alpha, \beta, \tau)$$

is a ***positive effect axiom***. This represents that, if γ is true and α occurs at τ, then β will be true after τ.

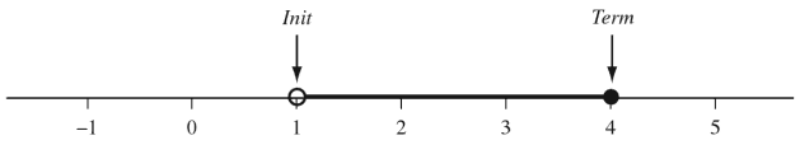

Figure 3.1 Truth value of a fluent in EC.

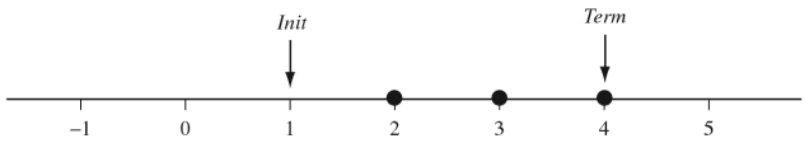

Figure 3.2 Truth value of a fluent in DEC.

DEFINITION 3.2

If γ is a condition representing the context, α is an event term, β is a fluent term, and τ is a timepoint term, then

$$\gamma \Rightarrow Terminates(\alpha, \beta, \tau)$$

is a ***negative effect axiom***. This represents that, if γ is true and α occurs at τ, then β will be false after τ.

Using the conjunction of axioms EC, a fluent is true (false) for times greater than the time of the initiating (terminating) event. Using the conjunction of axioms DEC, a fluent is true (false) starting one timepoint after the time of the initiating (terminating) event. For example, suppose we have *Initiates*(*Init*, *Fluent*, *t*), *Happens*(*Init*, 1), *Terminates*(*Term*, *Fluent*, *t*), and *Happens*(*Term*, 4). Figure 3.1 shows when *Fluent* is true using the conjunction of axioms EC; Figure 3.2 shows when *Fluent* is true using DEC.

3.1.1 Example: Telephone

We perform complex commonsense reasoning about the effects of events whenever we use a telephone. The behavior of a phone is highly context-sensitive.

$$HoldsAt(Connected(p_1, p_2), t) \Rightarrow \\ Initiates(SetDown(a, p_1), Disconnected(p_2), t) \tag{3.18}$$

$$HoldsAt(Connected(p_1, p_2), t) \Rightarrow \\ Terminates(SetDown(a, p_1), Connected(p_1, p_2), t) \tag{3.19}$$

Similarly, if phone p_1 is connected to phone p_2 and an agent sets down p_2, then p_2 will be idle, p_1 will be disconnected, and p_1 will no longer be connected to p_2:

$$HoldsAt(Connected(p_1, p_2), t) \Rightarrow \\ Initiates(SetDown(a, p_2), Idle(p_2), t) \tag{3.20}$$

$$HoldsAt(Connected(p_1, p_2), t) \Rightarrow \\ Initiates(SetDown(a, p_2), Disconnected(p_1), t) \tag{3.21}$$

$$HoldsAt(Connected(p_1, p_2), t) \Rightarrow \\ Terminates(SetDown(a, p_2), Connected(p_1, p_2), t) \tag{3.22}$$

If an agent sets down a phone that is disconnected, then the phone will be idle and the phone will no longer be disconnected:

$$HoldsAt(Disconnected(p), t) \Rightarrow \\ Initiates(SetDown(a, p), Idle(p), t) \tag{3.23}$$

$$HoldsAt(Disconnected(p), t) \Rightarrow \\ Terminates(SetDown(a, p), Disconnected(p), t) \tag{3.24}$$

Let us now use this axiomatization to solve a particular reasoning problem. We start by specifying some observations. At timepoint 0, all phones are idle, no phones have a dial tone or busy signal, no phones are ringing other phones, no phones are connected to other phones, and no phones are disconnected:

$$HoldsAt(Idle(p), 0) \tag{3.25}$$

$$\neg HoldsAt(DialTone(p), 0) \tag{3.26}$$

$$\neg HoldsAt(BusySignal(p), 0) \tag{3.27}$$

$$\neg HoldsAt(Ringing(p_1, p_2), 0) \tag{3.28}$$

$$\neg HoldsAt(Connected(p_1, p_2), 0) \tag{3.29}$$

$$\neg HoldsAt(Disconnected(p), 0) \tag{3.30}$$

We specify that fluents are never released from the commonsense law of inertia:

$$\neg ReleasedAt(f, t) \tag{3.31}$$

We specify a narrative. One agent picks up the phone and dials another agent, and then the other agent answers:

$$Happens(PickUp(Agent1, Phone1), 0) \tag{3.32}$$
$$Happens(Dial(Agent1, Phone1, Phone2), 1) \tag{3.33}$$
$$Happens(PickUp(Agent2, Phone2), 2) \tag{3.34}$$

We can then show that the two agents will be connected.

PROPOSITION 3.1 Let Σ be the conjunction of (3.1) through (3.24). Let $\Delta = (3.32) \wedge (3.33) \wedge (3.34)$. Let $\Omega = U[PickUp, SetDown, Dial] \wedge U[Idle, DialTone, Ringing, BusySignal, Connected, Disconnected]$. Let Γ be the conjunction of (3.25) through (3.31). Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Gamma \wedge DEC \models HoldsAt(Connected(Phone1, Phone2), 3)$$

Proof From $CIRC[\Sigma; Initiates, Terminates, Releases]$, Theorem 2.2, and Theorem 2.1, we have

$$Initiates(e, f, t) \Leftrightarrow \tag{3.35}$$
$$\exists a, p\,(e = PickUp(a, p) \wedge f = DialTone(p) \wedge HoldsAt(Idle(p), t)) \vee$$
$$\exists a, p\,(e = SetDown(a, p) \wedge f = Idle(p) \wedge HoldsAt(DialTone(p), t)) \vee$$
$$\exists a, p_1, p_2\,(e = Dial(a, p_1, p_2) \wedge$$
$$f = Ringing(p_1, p_2) \wedge$$
$$HoldsAt(DialTone(p_1), t) \wedge$$
$$HoldsAt(Idle(p_2), t)) \vee$$
$$\exists a, p_1, p_2\,(e = Dial(a, p_1, p_2) \wedge$$
$$f = BusySignal(p_1) \wedge$$
$$HoldsAt(DialTone(p_1), t) \wedge$$
$$\neg HoldsAt(Idle(p_2), t)) \vee$$
$$\exists a, p\,(e = SetDown(a, p) \wedge f = Idle(p) \wedge HoldsAt(BusySignal(p), t)) \vee$$
$$\exists a, p_1, p_2\,(e = SetDown(a, p_1) \wedge$$
$$f = Idle(p_1) \wedge$$
$$HoldsAt(Ringing(p_1, p_2), t)) \vee$$
$$\exists a, p_1, p_2\,(e = SetDown(a, p_1) \wedge$$
$$f = Idle(p_2) \wedge$$

$$
\begin{aligned}
&HoldsAt(Ringing(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\, (e = PickUp(a, p_2) \wedge \\
&f = Connected(p_1, p_2) \wedge \\
&HoldsAt(Ringing(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\, (e = SetDown(a, p_1) \wedge \\
&f = Idle(p_1) \wedge \\
&HoldsAt(Connected(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\, (e = SetDown(a, p_1) \wedge \\
&f = Disconnected(p_2) \wedge \\
&HoldsAt(Connected(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\, (e = SetDown(a, p_2) \wedge \\
&f = Idle(p_2) \wedge \\
&HoldsAt(Connected(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\, (e = SetDown(a, p_2) \wedge \\
&f = Disconnected(p_1) \wedge \\
&HoldsAt(Connected(p_1, p_2), t)) \vee \\
&\exists a, p\, (e = SetDown(a, p) \wedge f = Idle(p) \wedge HoldsAt(Disconnected(p), t))
\end{aligned}
$$

$$
\begin{aligned}
&Terminates(e, f, t) \Leftrightarrow \qquad (3.36) \\
&\exists a, p\, (e = PickUp(a, p) \wedge f = Idle(p) \wedge HoldsAt(Idle(p), t)) \vee \\
&\exists a, p\, (e = SetDown(a, p) \wedge f = DialTone(p) \wedge HoldsAt(DialTone(p), t)) \vee \\
&\exists a, p_1, p_2\, (e = Dial(a, p_1, p_2) \wedge \\
&f = DialTone(p_1) \wedge \\
&HoldsAt(DialTone(p_1), t) \wedge \\
&HoldsAt(Idle(p_2), t)) \vee \\
&\exists a, p_1, p_2 \\
&(e = Dial(a, p_1, p_2) \wedge \\
&f = Idle(p_2) \wedge \\
&HoldsAt(DialTone(p_1), t) \wedge \\
&HoldsAt(Idle(p_2), t)) \vee \\
&\exists a, p_1, p_2 \\
&(e = Dial(a, p_1, p_2) \wedge \\
&f = DialTone(p_1) \wedge \\
&HoldsAt(DialTone(p_1), t) \wedge \\
&\neg HoldsAt(Idle(p_2), t)) \vee \\
&\exists a, p\, (e = SetDown(a, p) \wedge f = BusySignal(p) \wedge HoldsAt(BusySignal(p), t)) \vee
\end{aligned}
$$

$$\begin{aligned}
&\exists a, p_1, p_2\,(e = SetDown(a, p_1)\ \wedge f = Ringing(p_1, p_2) \wedge \\
&HoldsAt(Ringing(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\,(e = PickUp(a, p_2) \wedge \\
&f = Ringing(p_1, p_2) \wedge \\
&HoldsAt(Ringing(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\,(e = SetDown(a, p_1) \wedge \\
&f = Connected(p_1, p_2) \wedge \\
&HoldsAt(Connected(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\,(e = SetDown(a, p_2) \wedge \\
&f = Connected(p_1, p_2) \wedge \\
&HoldsAt(Connected(p_1, p_2), t)) \vee \\
&\exists a, p\,(e = SetDown(a, p) \wedge \\
&f = Disconnected(p) \wedge \\
&HoldsAt(Disconnected(p), t))
\end{aligned}$$

$$\neg Releases(e, f, t) \tag{3.37}$$

From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have

$$\begin{aligned}
&Happens(e, t) \Leftrightarrow \\
&(e = PickUp(Agent1, Phone1) \wedge t = 0) \vee \\
&(e = Dial(Agent1, Phone1, Phone2) \wedge t = 1) \vee \\
&(e = PickUp(Agent2, Phone2) \wedge t = 2)
\end{aligned} \tag{3.38}$$

From (3.32) (which follows from (3.38)), (3.25), (3.1) (which follows from (3.35)), and DEC9, we have

$$HoldsAt(DialTone(Phone1), 1) \tag{3.39}$$

From (3.38) and (3.36), we have $\neg\exists e\,(Happens(e, 0) \wedge Terminates(e, Idle(Phone2), 0))$. From this, (3.25), (3.31), and DEC5, we have

$$HoldsAt(Idle(Phone2), 1) \tag{3.40}$$

From (3.33) (which follows from (3.38)), (3.39), (3.40), (3.5) (which follows from (3.35)), and DEC9, we have $HoldsAt(Ringing(Phone1, Phone2), 2)$. From this, (3.34) (which follows from (3.38)), (3.15) (which follows from (3.35)), and DEC9, we have $HoldsAt(Connected(Phone1, Phone2), 3)$. ■

3.2 Effect Axiom Idioms

In this section, we present several commonly used idioms involving effect axioms.

Setting and Resetting

One event sets a fluent; another event resets the fluent. If o is set, then o will be on, whereas, if o is reset, then o will no longer be on:

$$Initiates(Set(o), On(o), t)$$
$$Terminates(Reset(o), On(o), t)$$

Flipping

An event flips the truth value of a fluent. If o is not on and o is flipped, then o will be on, but, if o is on and o is flipped, then o will no longer be on:

$$\neg HoldsAt(On(o), t) \Rightarrow Initiates(Flip(o), On(o), t)$$
$$HoldsAt(On(o), t) \Rightarrow Terminates(Flip(o), On(o), t)$$

Selection

An event selects from among a number of values. If the value is v_1 and the value v_2 is selected, then the value will be v_2 and will no longer be v_1:

$$Initiates(Select(o, v_2), Value(o, v_2), t)$$

$$HoldsAt(Value(o, v_1), t) \wedge v_1 \neq v_2 \Rightarrow$$
$$Terminates(Select(o, v_2), Value(o, v_1), t)$$

We may wish to represent explicitly that a value is changed from one value to another:

$$Initiates(Change(o, v_1, v_2), Value(o, v_2), t)$$

$$HoldsAt(Value(o, v_1), t) \wedge v_1 \neq v_2 \Rightarrow$$
$$Terminates(Change(o, v_1, v_2), Value(o, v_1), t)$$

By contraposition this is equivalent to $\neg\gamma \Rightarrow \neg Happens(\alpha, \tau)$. Thus, if an event occurs whose action precondition is not true, then inconsistency arises.

Action precondition axioms provide an elaboration-tolerant way of expressing qualifications. Whenever we wish to add a qualification, we may simply add an action precondition axiom. Fluent precondition axioms can also be made elaboration tolerant by using default reasoning, as discussed in Section 12.4.

3.3.3 Example: Walking through a Door

Suppose that in order for an agent to walk through a door, the agent must be near the door:

$$Happens(WalkThroughDoor(a, d), t) \Rightarrow HoldsAt(Near(a, d), t) \tag{3.41}$$

Suppose further that Nathan is not near a door and walks through a door:

$$\neg HoldsAt(Near(Nathan, Door), 1) \tag{3.42}$$
$$Happens(WalkThroughDoor(Nathan, Door), 1) \tag{3.43}$$

We then get inconsistency.

PROPOSITION 3.2 The conjunction of (3.41), (3.42), and (3.43) is inconsistent.

Proof From (3.43) and (3.41), we have $HoldsAt(Near(Nathan, Door), 1)$, which contradicts (3.42). ■

3.4 State Constraints

Some properties of the world follow other properties in a lawlike fashion. We represent relationships that hold among properties over all timepoints using state constraints.

DEFINITION 3.5 If γ_1 and γ_2 are conditions, then γ_1, $\gamma_1 \Rightarrow \gamma_2$, and $\gamma_1 \Leftrightarrow \gamma_2$ are ***state constraints***.

Table 3.1 shows some typical state constraints involving one, two, and three or more fluents. In this section, we describe some sample uses of state constraints.

Irreflexive and Antisymmetric Relation

Suppose we have a fluent $On(o_1, o_2)$, which represents that an object o_1 is on top of an object o_2. We can use state constraints to specify that On denotes an irreflexive

Table 3.1 Typical state constraints

State constraint	*Axiom*
Reflexive relation	$HoldsAt(R(a,a),t)$
Irreflexive relation	$\neg HoldsAt(R(a,a),t)$
Symmetric relation	$HoldsAt(R(a,b),t) \Rightarrow HoldsAt(R(b,a),t)$
Antisymmetric relation	$HoldsAt(R(a,b),t) \wedge a \neq b \Rightarrow$ $\neg HoldsAt(R(b,a),t)$
Transitive relation	$HoldsAt(R(a,b),t) \wedge HoldsAt(R(b,c),t) \Rightarrow$ $HoldsAt(R(a,c),t)$
Intransitive relation	$HoldsAt(R(a,b),t) \wedge HoldsAt(R(b,c),t) \Rightarrow$ $\neg HoldsAt(R(a,c),t)$
Trichotomous relation	$HoldsAt(R(a,b),t) \dot{\vee} HoldsAt(R(b,a),t) \dot{\vee} a = b$
Total relation	$\exists b\ HoldsAt(R(a,b),t)$
Functional relation	$HoldsAt(R(a,b),t) \wedge HoldsAt(R(a,c),t) \Rightarrow b = c$
Surjective relation	$\exists a\ HoldsAt(R(a,b),t)$
Injective relation	$HoldsAt(R(a,c),t) \wedge HoldsAt(R(b,c),t) \Rightarrow a = b$
Negation	$HoldsAt(R(a_1,\ldots,a_n),t) \Leftrightarrow$ $\neg HoldsAt(S(a_1,\ldots,a_n),t)$
Converse	$HoldsAt(R(a,b),t) \Leftrightarrow HoldsAt(S(b,a),t)$
Composite	$HoldsAt(R(a,b),t) \wedge HoldsAt(S(b,c),t) \Leftrightarrow$ $HoldsAt(T(a,c),t)$
Union	$HoldsAt(R(a_1,\ldots,a_n),t) \Leftrightarrow$ $HoldsAt(S_1(a_1,\ldots,a_n),t) \vee \cdots \vee$ $HoldsAt(S_k(a_1,\ldots,a_n),t)$
Intersection	$HoldsAt(R(a_1,\ldots,a_n),t) \Leftrightarrow$ $HoldsAt(S_1(a_1,\ldots,a_n),t) \wedge \cdots \wedge$ $HoldsAt(S_k(a_1,\ldots,a_n),t)$
Exactly one	$HoldsAt(R_1(a_1,\ldots,a_n),t) \dot{\vee} \cdots \dot{\vee}$ $HoldsAt(R_k(a_1,\ldots,a_n),t)$

and antisymmetric relation. That is, an object can never be on top of itself:

$$\neg HoldsAt(On(o,o),t)$$

and, if one object o_1 is on top of another object o_2, then o_2 cannot also be on top of o_1:

$$HoldsAt(On(o_1,o_2),t) \wedge o_1 \neq o_2 \Rightarrow \neg HoldsAt(On(o_2,o_1),t)$$

Functional and Total Relation

Consider a fluent $At(o, l)$, which represents that object o is located at location l. We may specify that At denotes a functional and total relation. An object is in at most one location at a time:

$$HoldsAt(At(o, l_1), t) \wedge HoldsAt(At(o, l_2), t) \Rightarrow l_1 = l_2$$

and, at all times, every object has a location:

$$\exists l \; HoldsAt(At(o, l), t)$$

State constraints can be used to address the qualification problem, as demonstrated in the following example: Let $Occupies(p, s)$ represent that a chess piece p occupies a square s of a chessboard. We wish to represent that it is not possible to move a piece onto a square that is already occupied. We may do this by specifying that *Occupies* denotes an injective relation. That is, at most one piece occupies a square at a time:

$$HoldsAt(Occupies(p_1, s), t) \wedge HoldsAt(Occupies(p_2, s), t) \Rightarrow p_1 = p_2$$

A state constraint used to represent a qualification that prevents an event from occurring is called a *qualification constraint*. State constraints provide an elaboration-tolerant way of expressing qualifications because, whenever we wish to add a qualification, we may simply add a state constraint.

Negation

We may wish to specify that one fluent represents the negation of another fluent. An example is the fact that a device is off if and only if it is not on:

$$HoldsAt(Off(d), t) \Leftrightarrow \neg HoldsAt(On(d), t)$$

Intersection

A light is lit if and only if it is on and not broken:

$$HoldsAt(Lit(l), t) \Leftrightarrow HoldsAt(On(l), t) \wedge \neg HoldsAt(Broken(l), t)$$

Exactly One

At all times, a person is either lying, sitting, or standing:

$$\begin{aligned} &HoldsAt(Lying(p), t) \; \dot{\vee} \\ &HoldsAt(Sitting(p), t) \; \dot{\vee} \\ &HoldsAt(Standing(p), t) \end{aligned} \tag{3.44}$$

3.4.1 Example: Telephone Revisited

An important use of state constraints is to simplify the specification of initial conditions. We can use state constraints to tighten up the telephone axiomatization given in Section 3.1.1.

We add several axioms. A phone cannot be ringing itself:

$$\neg HoldsAt(Ringing(p, p), t) \tag{3.45}$$

If phone p_1 is ringing phone p_2, then p_2 cannot be ringing p_1:

$$HoldsAt(Ringing(p_1, p_2), t) \wedge p_1 \neq p_2 \Rightarrow \neg HoldsAt(Ringing(p_2, p_1), t) \tag{3.46}$$

A phone cannot be connected to itself:

$$\neg HoldsAt(Connected(p, p), t) \tag{3.47}$$

If phone p_1 is connected to phone p_2, then p_2 cannot be connected to p_1:

$$HoldsAt(Connected(p_1, p_2), t) \wedge p_1 \neq p_2 \Rightarrow \neg HoldsAt(Connected(p_2, p_1), t) \tag{3.48}$$

At any time, a phone either is idle, has a dial tone, has a busy signal, is ringing another phone, is being rung by another phone, is connected to another phone, or is disconnected:

$$\begin{aligned} &HoldsAt(Idle(p), t) \dot{\vee} \\ &HoldsAt(DialTone(p), t) \dot{\vee} \\ &HoldsAt(BusySignal(p), t) \dot{\vee} \\ &\exists p_1\, HoldsAt(Ringing(p, p_1), t) \dot{\vee} \\ &\exists p_1\, HoldsAt(Ringing(p_1, p), t) \dot{\vee} \\ &\exists p_1\, HoldsAt(Connected(p, p_1), t) \dot{\vee} \\ &\exists p_1\, HoldsAt(Connected(p_1, p), t) \dot{\vee} \\ &HoldsAt(Disconnected(p), t) \end{aligned} \tag{3.49}$$

These state constraints simplify specification of initial conditions. For example, from $HoldsAt(Idle(Phone1), 0)$, $HoldsAt(Idle(Phone2), 0)$, (3.45), (3.46), (3.47),

(3.48), and (3.49), we have all of the following:

$$\neg HoldsAt(DialTone(Phone1), 0)$$
$$\neg HoldsAt(BusySignal(Phone1), 0)$$
$$\neg HoldsAt(Ringing(Phone1, Phone1), 0)$$
$$\neg HoldsAt(Ringing(Phone1, Phone2), 0)$$
$$\neg HoldsAt(Connected(Phone1, Phone1), 0)$$
$$\neg HoldsAt(Connected(Phone1, Phone2), 0)$$
$$\neg HoldsAt(Disconnected(Phone1), 0)$$
$$\neg HoldsAt(DialTone(Phone2), 0)$$
$$\neg HoldsAt(BusySignal(Phone2), 0)$$
$$\neg HoldsAt(Ringing(Phone2, Phone2), 0)$$
$$\neg HoldsAt(Ringing(Phone2, Phone1), 0)$$
$$\neg HoldsAt(Connected(Phone2, Phone2), 0)$$
$$\neg HoldsAt(Connected(Phone2, Phone1), 0)$$
$$\neg HoldsAt(Disconnected(Phone2), 0)$$

Therefore, we no longer have to specify these initial conditions explicitly.

Bibliographic Notes

GPS

An early problem-solving program was GPS (Newell and Simon, 1961). GPS uses subgoaling to find a sequence of operators that transforms an object from an initial state into a goal state. The subgoal to transform an object a into an object b is achieved as follows:

1. If a and b are the same, return with success.
2. Find a difference d between a and b.
3. Invoke subgoal to reduce d between a and b, producing a'.
4. Recursively invoke subgoal to transform a' into b.

The subgoal to reduce d between a and b is achieved as follows:

1. Select relevant operator (action) o.
2. Invoke subgoal to apply o to a, producing a'.

Preconditions

The distinction between action preconditions and fluent preconditions is from R. Miller and Shanahan (2002, p. 464). Baral (1995) makes a similar distinction between an "executability condition of an action" and "preconditions of effects" (p. 2017). Our representation of an action precondition axiom is from Shanahan and Witkowski (2004). This representation must be used with caution when solving abduction or planning problems, as pointed out by R. Miller and Shanahan (2002, p. 465): If the initial situation is not completely specified, then *Happens*(*event*, *time*) becomes a plan for achieving the precondition.

State Constraints

McCarthy and Hayes (1969, p. 478) give the following transitive law in the situation calculus:

$$\forall x . \forall y . \forall z . \forall s . in(x, y, s) \wedge in(y, z, s) \supset in(x, z, s)$$

Green (1969) introduces a kind of axiom that represents "an implication that holds for a fixed state" (p. 78). State constraints (Genesereth and Nilsson, 1987, p. 267) are also called "domain constraints" (Ginsberg and Smith, 1987a, p. 237); E. Davis (1990) calls state constraints "state coherence axioms" (p. 193). Such constraints in the event calculus are discussed by Shanahan (1995a, pp. 255, 262; 1996, p. 685; 1997b, pp. 11, 39–40, 275, 285–286, 323–324; 1999a, pp. 417–419). Reiter (2001, pp. 401–406) discusses the treatment of state constraints in the situation calculus. Doherty, Gustafsson, Karlsson, and Kvarnström (1998, p. 16) discuss domain constraints in temporal action logics. Gustafsson and Doherty (1996) call state constraints that mention multiple timepoints "transition constraints" (p. 92). They give an example that in the event calculus is represented as

$$\neg HoldsAt(Alive(a), t) \Rightarrow \neg HoldsAt(Alive(a), t + 1)$$

Exercises

3.1 Write an axiom to formalize that a person who eats is no longer hungry.

3.2 Write an axiom to formalize the following. If two agents are in the same room, the first agent is listening to the second agent, and the first agent tells the second agent a fact, then the second agent will know that fact.

3.3 Using the axiom written in Exercise 3.2, prove that, if Nathan and Ryan are in the same room, Ryan is listening to Nathan, and Nathan tells Ryan a particular fact, then Ryan will know that fact.

3.4 Formalize that if an agent is outside, it is cold outside, and the agent is not wearing a coat, then the agent is cold. Incorporate other weather conditions such as rain. Include axioms for putting on and taking off a coat.

3.5 Formalize the opening of a book to a particular page number and the closing of a book.

3.6 Formalize the formation and dissolution of interpersonal relationships such as friendship and marriage (Schank and Abelson, 1977; Dyer, 1983).

3.7 Simple axioms for waking up and falling asleep are given in Section 2.7.1. Create a more detailed formalization of the human sleep cycle. Incorporate getting out of bed, getting tired, lying in bed, and waiting to fall asleep.

3.8 Formalize lighting and putting out a fire.

3.9 State constraint (3.44) says that a person is either lying, sitting, or standing. Add appropriate event predicates, fluent predicates, and axioms to formalize lying down on something, sitting down on something, and standing up.

3.10 Write state constraints relating various expressions for time of day, such as daytime, nighttime, morning, afternoon, and evening.

3.11 Are there any bugs in the formalization of a telephone in Sections 3.1.1 and 3.4.1? Consider the following scenarios:

- Two agents dial one another simultaneously.
- One agent dials another agent at the same instant that the other agent picks up the phone.

3.12 Prove Proposition 3.1 using the conjunction of axioms EC instead of the conjunction of axioms DEC.

CHAPTER 4

The Triggering of Events

So far we have addressed that when an event occurs, a fluent changes its truth value. What about the opposite? That is, when a fluent changes its truth value, an event occurs; more generally, when a particular condition becomes true, an event occurs. We call such an event a *triggered event*. An example of a triggered event is a ball bouncing off of a wall when it reaches the wall. This chapter addresses the triggering of events in response to conditions; we also discuss triggered fluents.

4.1 Trigger Axioms

We specify when a triggered event occurs using a trigger axiom.

DEFINITION 4.1

If γ is a condition, α is an event term, and τ is a timepoint term, then

$$\gamma \Rightarrow Happens(\alpha, \tau)$$

is a ***trigger axiom***.

4.1.1 Example: Alarm Clock

Whenever we use an alarm clock, we perform commonsense reasoning about triggered events. The alarm going off is a triggered event, and we can formalize the operation of an alarm clock using trigger axioms.

We start with some effect axioms. If a clock's alarm time is t_1 and an agent sets the clock's alarm time to t_2, then the clock's alarm time will be t_2 and will no longer be t_1:

$$HoldsAt(AlarmTime(c, t_1), t) \wedge t_1 \neq t_2 \Rightarrow \\ Initiates(SetAlarmTime(a, c, t_2), AlarmTime(c, t_2), t) \tag{4.1}$$

$$HoldsAt(AlarmTime(c, t_1), t) \wedge t_1 \neq t_2 \Rightarrow \\ Terminates(SetAlarmTime(a, c, t_2), AlarmTime(c, t_1), t) \tag{4.2}$$

If an agent turns on a clock's alarm, then it will be on:

$$Initiates(TurnOnAlarm(a, c), AlarmOn(c), t) \tag{4.3}$$

If an agent turns off a clock's alarm, then it will no longer be on:

$$Terminates(TurnOffAlarm(a, c), AlarmOn(c), t) \tag{4.4}$$

If an alarm starts beeping, then it will be beeping:

$$Initiates(StartBeeping(c), Beeping(c), t) \tag{4.5}$$

If an agent turns off a clock's alarm, then the clock will no longer be beeping:

$$Terminates(TurnOffAlarm(a, c), Beeping(c), t) \tag{4.6}$$

We have a state constraint that says that a clock has a unique alarm time at any given time:

$$HoldsAt(AlarmTime(c, t_1), t) \wedge HoldsAt(AlarmTime(c, t_2), t) \Rightarrow t_1 = t_2 \tag{4.7}$$

Now we use a trigger axiom. If a clock's alarm time is the present moment and the alarm is on, then the clock starts beeping:

$$HoldsAt(AlarmTime(c, t), t) \wedge HoldsAt(AlarmOn(c), t) \Rightarrow \\ Happens(StartBeeping(c), t) \tag{4.8}$$

Let us use the following observations and narrative. At timepoint 0, the alarm is not on, the alarm is not beeping, and the alarm time is set to 10:

$$\neg HoldsAt(AlarmOn(Clock), 0) \tag{4.9}$$

$$\neg HoldsAt(Beeping(Clock), 0) \tag{4.10}$$

$$HoldsAt(AlarmTime(Clock, 10), 0) \tag{4.11}$$

$$\neg ReleasedAt(f, t) \tag{4.12}$$

At timepoint 0, Nathan sets the alarm clock for timepoint 2; and at timepoint 1, he turns on the alarm:

$$Happens(SetAlarmTime(Nathan, Clock, 2), 0) \tag{4.13}$$

$$Happens(TurnOnAlarm(Nathan, Clock), 1) \tag{4.14}$$

We can then show that the alarm clock will be beeping at timepoint 3.

PROPOSITION 4.1

Let $\Sigma = (4.1) \wedge (4.2) \wedge (4.3) \wedge (4.4) \wedge (4.5) \wedge (4.6)$, $\Delta = (4.8) \wedge (4.13) \wedge (4.14)$, $\Omega = U[SetAlarmTime, TurnOnAlarm, TurnOffAlarm, StartBeeping] \wedge U[AlarmTime, AlarmOn, Beeping]$, $\Psi = (4.7)$, and $\Gamma = (4.9) \wedge (4.10) \wedge (4.11) \wedge (4.12)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Psi \wedge \Gamma \wedge \mathrm{DEC} \models HoldsAt(Beeping(Clock), 3)$$

Proof From $CIRC[\Sigma; Initiates, Terminates, Releases]$, Theorem 2.2, and Theorem 2.1, we have

$$\begin{aligned}
&Initiates(e,f,t) \Leftrightarrow && (4.15)\\
&\exists a, c, t_1, t_2\ (e = SetAlarmTime(a,c,t_2)\ \wedge\\
&f = AlarmTime(c,t_2)\ \wedge\\
&HoldsAt(AlarmTime(c,t_1),t)\ \wedge\\
&t_1 \neq t_2)\ \vee\\
&\exists a, c\ (e = TurnOnAlarm(a,c) \wedge f = AlarmOn(c))\ \vee\\
&\exists c\ (e = StartBeeping(c) \wedge f = Beeping(c))
\end{aligned}$$

$$\begin{aligned}
&Terminates(e,f,t) \Leftrightarrow && (4.16)\\
&\exists a, c, t_1, t_2\ (e = SetAlarmTime(a,c,t_2)\ \wedge\\
&f = AlarmTime(c,t_1)\ \wedge\\
&HoldsAt(AlarmTime(c,t_1),t)\ \wedge\\
&t_1 \neq t_2)\ \vee\\
&\exists a, c\ (e = TurnOffAlarm(a,c) \wedge f = AlarmOn(c))\ \vee\\
&\exists a, c\ (e = TurnOffAlarm(a,c) \wedge f = Beeping(c))\\
&\neg Releases(e,f,t) && (4.17)
\end{aligned}$$

From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have

$$\begin{aligned}
&Happens(e,t) \Leftrightarrow && (4.18)\\
&\exists c\ \big(e = StartBeeping(c)\ \wedge
\end{aligned}$$

account, then a service fee is charged to the account:

$$\begin{gathered} HoldsAt(Balance(a,x),t) \wedge \\ x < MinimumBalance(a) \wedge \\ \neg HoldsAt(ServiceFeeCharged(a),t) \Rightarrow \\ Happens(ChargeServiceFee(a),t) \end{gathered} \tag{4.27}$$

When a service fee is charged to an account, a note is made of this fact so that the account is not repeatedly charged:

$$Initiates(ChargeServiceFee(a), ServiceFeeCharged(a), t) \tag{4.28}$$

This is reset once each month:

$$EndOfMonth(t) \Rightarrow Happens(MonthlyReset(a), t) \tag{4.29}$$

$$Terminates(MonthlyReset(a), ServiceFeeCharged(a), t) \tag{4.30}$$

If a service fee is charged to an account, then the balance of the account decreases by the amount of the service fee:

$$\begin{gathered} HoldsAt(Balance(a,x),t) \Rightarrow \\ Initiates(ChargeServiceFee(a), Balance(a, x - ServiceFee(a)), t) \end{gathered} \tag{4.31}$$

$$\begin{gathered} HoldsAt(Balance(a,x),t) \Rightarrow \\ Terminates(ChargeServiceFee(a), Balance(a,x), t) \end{gathered} \tag{4.32}$$

Let us use the following observations and narrative about two bank accounts. Initially, a service fee has not been charged to the first account, the balance in both accounts is 1000, the minimum balance of the first account is 500, and the service fee of the first account is 5:

$$\neg HoldsAt(ServiceFeeCharged(Account1), 0) \tag{4.33}$$

$$HoldsAt(Balance(Account1, 1000), 0) \tag{4.34}$$

$$HoldsAt(Balance(Account2, 1000), 0) \tag{4.35}$$

$$MinimumBalance(Account1) = 500 \tag{4.36}$$

$$ServiceFee(Account1) = 5 \tag{4.37}$$

$$\neg ReleasedAt(f, t) \tag{4.38}$$

Two transfers are made from the first account to the second account. A transfer of 200 is made and then a transfer of 400 is made:

$$Happens(Transfer(Account1, Account2, 200), 0) \tag{4.39}$$

$$Happens(Transfer(Account1, Account2, 400), 1) \tag{4.40}$$

We can show that, after these transfers, the balance in the first account will be 395.

PROPOSITION 4.2

Let $\Sigma = (4.22) \wedge (4.23) \wedge (4.24) \wedge (4.25) \wedge (4.28) \wedge (4.30) \wedge (4.31) \wedge (4.32)$, $\Delta = (4.27) \wedge (4.29) \wedge (4.39) \wedge (4.40)$, $\Omega = U[Transfer, ChargeServiceFee, MonthlyReset] \wedge U[Balance, ServiceFeeCharged]$, $\Psi = (4.26)$, and $\Gamma = (4.33) \wedge (4.34) \wedge (4.35) \wedge (4.36) \wedge (4.37) \wedge (4.38)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Psi \wedge \Gamma \wedge DEC \models HoldsAt(Balance(Account1, 395), 3)$$

Proof From $CIRC[\Sigma; Initiates, Terminates, Releases]$, Theorem 2.2, and Theorem 2.1, we have

$$\begin{aligned}
&Initiates(e, f, t) \Leftrightarrow \qquad (4.41)\\
&\exists a_1, a_2, x_1, x_2, x_3\ (e = Transfer(a_1, a_2, x_3) \wedge\\
&f = Balance(a_2, x_2 + x_3) \wedge\\
&HoldsAt(Balance(a_1, x_1), t) \wedge\\
&HoldsAt(Balance(a_2, x_2), t) \wedge\\
&x_3 > 0 \wedge x_1 \geq x_3) \vee\\
&\exists a_1, a_2, x_1, x_2, x_3\ (e = Transfer(a_1, a_2, x_3) \wedge\\
&f = Balance(a_1, x_1 - x_3) \wedge\\
&HoldsAt(Balance(a_1, x_1), t) \wedge\\
&HoldsAt(Balance(a_2, x_2), t) \wedge\\
&x_3 > 0 \wedge x_1 \geq x_3) \vee\\
&\exists a\ (e = ChargeServiceFee(a) \wedge f = ServiceFeeCharged(a)) \vee\\
&\exists a, x\ (e = ChargeServiceFee(a) \wedge\\
&f = Balance(a, x - ServiceFee(a)) \wedge\\
&HoldsAt(Balance(a, x), t))
\end{aligned}$$

$$\begin{aligned}
&Terminates(e, f, t) \Leftrightarrow \qquad (4.42)\\
&\exists a_1, a_2, x_1, x_2, x_3\ (e = Transfer(a_1, a_2, x_3) \wedge\\
&f = Balance(a_2, x_2) \wedge\\
&HoldsAt(Balance(a_1, x_1), t) \wedge\\
&HoldsAt(Balance(a_2, x_2), t) \wedge\\
&x_3 > 0 \wedge x_1 \geq x_3) \vee
\end{aligned}$$

$$\begin{aligned}
&\exists a_1, a_2, x_1, x_2, x_3\,(e = Transfer(a_1, a_2, x_3) \wedge \\
&f = Balance(a_1, x_1) \wedge \\
&HoldsAt(Balance(a_1, x_1), t) \wedge \\
&HoldsAt(Balance(a_2, x_2), t) \wedge \\
&x_3 > 0 \wedge x_1 \geq x_3) \vee \\
&\exists a\,(e = MonthlyReset(a) \wedge f = ServiceFeeCharged(a)) \vee \\
&\exists a, x\,(e = ChargeServiceFee(a) \wedge \\
&f = Balance(a, x) \wedge \\
&HoldsAt(Balance(a, x), t))
\end{aligned}$$

$$\neg Releases(e, f, t) \tag{4.43}$$

From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have

$$\begin{aligned}
&Happens(e, t) \Leftrightarrow \qquad (4.44)\\
&\exists a, x\,(e = ChargeServiceFee(a) \wedge \\
&HoldsAt(Balance(a, x), t) \wedge \\
&x < MinimumBalance(a) \wedge \\
&\neg HoldsAt(ServiceFeeCharged(a), t)) \vee \\
&\exists a\,(e = MonthlyReset(a) \wedge EndOfMonth(t)) \vee \\
&(e = Transfer(Account1, Account2, 200) \wedge t = 0) \vee \\
&(e = Transfer(Account1, Account2, 400) \wedge t = 1)
\end{aligned}$$

From (4.39) (which follows from (4.44)), (4.34), (4.35), $200 > 0$, $1000 \geq 200$, (4.24) (which follows from (4.41)), and DEC9, we have

$$HoldsAt(Balance(Account1, 800), 1) \tag{4.45}$$

From (4.39) (which follows from (4.44)), (4.34), (4.35), $200 > 0$, $1000 \geq 200$, (4.22) (which follows from (4.41)), and DEC9, we have

$$HoldsAt(Balance(Account2, 1200), 1) \tag{4.46}$$

From (4.26), (4.34), (4.36), $\neg(1000 < 500)$, and (4.44), we have $\neg Happens(ChargeServiceFee(Account1), 0)$. From this, (4.44), and (4.41), we have $\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, ServiceFeeCharged(Account1), 0))$. From this, (4.33), (4.38), and DEC6, we have

$$\neg HoldsAt(ServiceFeeCharged(Account1), 1) \tag{4.47}$$

From (4.40) (which follows from (4.44)), (4.45), (4.46), $400 > 0$, $800 \geq 400$, (4.24) (which follows from (4.41)), and DEC9, we have

$$HoldsAt(Balance(Account1, 400), 2) \tag{4.48}$$

From (4.26), (4.45), (4.36), $\neg(800 < 500)$, and (4.44), we have $\neg Happens(ChargeServiceFee(Account1), 1)$. From this, (4.44), and (4.41), we have $\neg\exists e\,(Happens(e, 1) \wedge Initiates(e, ServiceFeeCharged(Account1), 1))$. From this, (4.47), (4.38), and DEC6, we have

$$\neg HoldsAt(ServiceFeeCharged(Account1), 2)$$

From this, (4.48), (4.36), $400 < 500$, and (4.27) (which follows from (4.44)), we have $Happens(ChargeServiceFee(Account1), 2)$. From this, (4.48), (4.37), (4.31) (which follows from (4.41)), and DEC9, we have $HoldsAt(Balance(Account1, 395), 3)$. ■

4.3 Triggered Fluents

So far, we have discussed how a trigger axiom is used to represent that a certain event occurs when a certain condition becomes true. What if we would like to represent that a fluent becomes true (or false) when a condition becomes true? This cannot be represented directly in the event calculus. Instead, we must introduce an event that is triggered by the condition and that initiates or terminates the fluent.

Thus, we represent that the condition γ initiates a fluent β as follows:

$$\gamma \Rightarrow Happens(\alpha, \tau)$$
$$Initiates(\alpha, \beta, t)$$

We represent that the condition γ terminates a fluent β as follows:

$$\gamma \Rightarrow Happens(\alpha, \tau)$$
$$Terminates(\alpha, \beta, t)$$

Bibliographic Notes

Shanahan (1990) introduced an early form of the trigger axiom into a simplified version of the original event calculus (Kowalski and Sergot, 1986). Shanahan (1995a, pp. 268–272; 1997b, pp. 305–313) uses a predicate $Triggers(s, e)$ to represent that an event e occurs in state s. Trigger axioms in the form used in this book were introduced by Shanahan (1996, p. 685) and are discussed in detail by Shanahan (1997b, pp. 258–265, 325–329). The method for representing triggered fluents is from Morgenstern (2001, p. 353).

Proposals for incorporating triggered events into the situation calculus have been made by Pinto (1994), R. Miller (1996), and Reiter (1996). See the discussion in the Bibliographic Notes of Chapter 15. Pinto (1998a) uses triggered events in the situation calculus to represent the starting and stopping of current in an electrical circuit.

Tran and Baral (2004b) incorporate triggered events into an action language inspired by $\mathcal{A}$ (Gelfond and Lifschitz, 1993), implement the language in AnsProlog (Baral, 2003), and apply triggered events to the modeling of molecular interactions in cells. The triggering rule (Tran and Baral, 2004b, p. 555)

$g_1, \ldots, g_m$ **n_triggers** b

represents that action b normally occurs when conditions $g_1, \ldots, g_m$ are true. The inhibition rule (p. 555)

$h_1, \ldots, h_l$ **inhibits** c

represents that action c does not occur when conditions $h_1, \ldots, h_l$ are true.

Triggered events are represented in action language $\mathcal{C}+$ (Giunchiglia et al., 2004) using action dynamic laws of the form (p. 70)

caused F **if** G

where F is an action and G is a condition. $\mathcal{C}+$ is discussed in Section 15.3.1 .

Exercises

4.1 Add a snooze alarm to the alarm clock axiomatization in Section 4.1.1. If when the alarm is beeping an agent presses the snooze button, then the alarm stops beeping and starts beeping again after nine timepoints.

4.2 Use the extended alarm clock axiomatization in Exercise 1 to prove that, if a particular alarm clock is set, the clock starts beeping at the appropriate time, and an agent hits the snooze button, then the alarm stops beeping after that time and is beeping 10 timepoints later.

4.3 Rework the axiomatization in Section 4.2.1 so that a monthly fee is charged at the end of each month rather than immediately.

4.4 Formalize the operation of a mousetrap. Prove that a mouse entering the trap is caught.

4.5 Formalize a price notification service for traders of financial instruments. When the price of a financial instrument falls below or rises above a certain level, the service informs the trader. Prove that, if the trader requests notification of when stock XYZ falls below 100, then when the stock falls below that level the service informs the trader.

4.6 Formalize that you introduce yourself when meeting someone for the first time.

PROPOSITION 5.3 If $\Sigma = (5.1)$, $\Delta = (5.4) \wedge (5.7)$, $\Omega = (5.5)$, $\Gamma = (5.2) \wedge (5.3)$, and CFA = (5.6), then $\Sigma \wedge \Delta \wedge \Omega \wedge \Gamma \wedge CFA$ is inconsistent.

Proof From (5.2), (5.4), (5.5), and (5.6), we have

$$\neg HoldsAt(On(Light1), 1) \tag{5.8}$$

From (5.1) and (5.7), we have $HoldsAt(On(Light1), 1)$, which contradicts (5.8). ■

Notice also that, if we remove (5.4), then we are unable to conclude $\neg HoldsAt(On(Light1), 1)$. Classical frame axioms were developed for use within the situation calculus (see Section 15.1). They are only useful in the event calculus if exactly one event occurs at each timepoint.

5.1.3 Explanation Closure Axioms

Another type of axiom, an *explanation closure axiom*, represents that a given fluent does not change unless certain events occur. For example, we represent that, if a light is off and the light is not turned on, then the light will still be off:

$$\begin{gathered}\neg HoldsAt(On(l), t) \wedge \\ \neg Happens(TurnOn(l), t) \Rightarrow \\ \neg HoldsAt(On(l), t + 1)\end{gathered} \tag{5.9}$$

If we had

$$\neg Happens(TurnOn(Light1), 0)$$

then we could show $\neg HoldsAt(On(Light1), 1)$ from (5.2) and (5.9). But this does not yet follow from the domain description. An additional mechanism is required to limit events to those that are known to have occurred.

5.1.4 Minimizing Event Occurrences

We use the mechanism of circumscription described in Section 2.6 to minimize event occurrences, by minimizing the extension of the *Happens* predicate. This gives us the desired result. If the first and second lights are initially off and the second light is turned on, then the first light will still be off:

PROPOSITION 5.4 Let $\Sigma = (5.1)$, $\Delta = (5.4)$, $\Omega = (5.5)$, $\Gamma = (5.2) \wedge (5.3)$, and ECA = (5.9). Then we have $\Sigma \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Gamma \wedge ECA \models \neg HoldsAt(On(Light1), 1)$.

Proof From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have

$$Happens(e,t) \Leftrightarrow (e = TurnOn(Light2) \wedge t = 0)$$

From this and (5.5), we have $\neg Happens(TurnOn(Light1), 0)$. From this, (5.2), and (5.9), we have $\neg HoldsAt(On(Light1), 1)$. ■

5.1.5 Introduction of *Initiates* Predicate

So far, this method requires us to enumerate all explanation closure axioms for the domain such as (5.9). We can avoid this by introducing the predicate $Initiates(e, f, t)$, which represents that, if event e occurs at timepoint t, then fluent f will be true at $t + 1$:

$$Happens(e,t) \wedge Initiates(e,f,t) \Rightarrow HoldsAt(f, t+1) \tag{5.10}$$

We can then generalize the explanation closure axiom (5.9) into:

$$\neg HoldsAt(f,t) \wedge \neg\exists e\,(Happens(e,t) \wedge Initiates(e,f,t)) \Rightarrow \neg HoldsAt(f, t+1) \tag{5.11}$$

Now suppose that instead of (5.1), we have

$$Initiates(TurnOn(l), On(l), t)$$

In order to show $\neg HoldsAt(On(Light1), 1)$ from $\neg HoldsAt(On(Light1), 0)$, we must show

$$\neg\exists e\,(Happens(e,0) \wedge Initiates(e, On(Light1), 0))$$

From the circumscription of *Happens* in (5.4), we have

$$Happens(e,0) \Leftrightarrow e = TurnOn(Light2)$$

Thus we require:

$$\neg Initiates(TurnOn(Light2), On(Light1), 0)$$

But this does not yet follow from the domain description. We require a method to limit the effects of events to those that are known.

5.1.6 Minimizing Event Effects

Again we use circumscription, this time to minimize the effects of events, by minimizing the extension of *Initiates*. We can again show, this time using *Initiates*, that after turning on the second light, the first light will still be off:

PROPOSITION 5.5 Let $\Sigma = (5.12)$, $\Delta = (5.4)$, $\Omega = (5.5)$, $\Gamma = (5.2) \wedge (5.3)$, and D1 = $(5.10) \wedge (5.11)$. Then we have $CIRC[\Sigma; Initiates] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Gamma \wedge D1 \models \neg HoldsAt(On(Light1), 1)$.

Proof From $CIRC[\Sigma; Initiates]$ and Theorem 2.1, we have

$$Initiates(e, f, t) \Leftrightarrow \exists l\,(e = TurnOn(l) \wedge f = On(l)) \tag{5.12}$$

From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have

$$Happens(e, t) \Leftrightarrow (e = TurnOn(Light2) \wedge t = 0) \tag{5.13}$$

From (5.12) and (5.5), we have

$$\neg Initiates(TurnOn(Light2), On(Light1), 0)$$

From this and (5.13), we have

$$\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, On(Light1), 0))$$

From this, (5.2), and (5.11), we have $\neg HoldsAt(On(Light1), 1)$. ■

These are the beginnings of the discrete event calculus. The axiom (5.11) is similar to DEC6, and (5.10) is the same as DEC9.

5.1.7 Introduction of *Terminates* Predicate

Along similar lines, we introduce the predicate $Terminates(e, f, t)$, which represents that, if event e occurs at timepoint t, then fluent f will be false at $t + 1$. We have an axiom similar to DEC5 and one that is the same as DEC10:

$$\begin{gathered} HoldsAt(f, t) \wedge \\ \neg\exists e\,(Happens(e, t) \wedge Terminates(e, f, t)) \Rightarrow \\ HoldsAt(f, t + 1) \end{gathered} \tag{5.14}$$

$$\begin{gathered} Happens(e, t) \wedge Terminates(e, f, t) \Rightarrow \\ \neg HoldsAt(f, t + 1) \end{gathered} \tag{5.15}$$

5.1.8 Discussion

The effects of events are enforced by (5.10) and (5.15), whereas the commonsense law of inertia is enforced by (5.11) and (5.14). In the conjunction of axioms EC, the effects of events are enforced by EC14 and EC15, whereas the commonsense law of inertia is enforced by EC9, EC10, EC14, and EC15. For instance, EC14 specifies that a fluent f that is initiated by an event that occurs at timepoint t_1 is true at timepoint $t_2 > t_1$ provided that $\neg StoppedIn(t_1, f, t_2)$, which by EC3 is equivalent to

$$\neg\exists e, t\,(Happens(e, t) \wedge t_1 < t < t_2 \wedge Terminates(e, f, t))$$

Notice that EC14 and EC15 enforce both the effects of events and the commonsense law of inertia. EC14 and EC15 enforce the commonsense law of inertia after a fluent has been initiated or terminated by an occurring event, whereas EC9 and EC10 enforce the commonsense law of inertia in all cases. If, for example, the truth value of a fluent is known at timepoint t, then EC9 and EC10 can be used to determine the truth value of the fluent after timepoint t.

Some redundancy exists between, say, EC9 and EC14, because after a fluent is initiated by an occurring event, both EC9 and EC14 specify that the fluent is true until it is terminated by an occurring event. This redundancy is not present in the conjunction of axioms DEC.

5.2 Representing Release from the Commonsense Law of Inertia

We may not always wish the commonsense law of inertia to be in force. In this section, we describe how fluents can be released from the commonsense law of inertia and then, at a later time, can again be made subject to this law.

5.2.1 Example: Yale Shooting Scenario

We start by considering the example of shooting a turkey. If a gun is loaded at timepoint 1 and used to shoot a turkey at timepoint 3, then the gun will fire and the turkey will no longer be alive. This simple example assumes that the shooter does not miss. In this case, the fact that the gun is loaded is subject to the commonsense law of inertia.

This example is due to Steve Hanks and Drew McDermott. If an agent loads a gun, then the gun will be loaded:

$$Initiates(Load(a, g), Loaded(g), t) \tag{5.16}$$

If a gun is loaded and agent a_1 shoots the gun at agent a_2, then a_2 will no longer be alive:

$$HoldsAt(Loaded(g), t) \Rightarrow \\ Terminates(Shoot(a_1, a_2, g), Alive(a_2), t) \tag{5.17}$$

If a gun is loaded and an agent shoots the gun, then the gun will no longer be loaded:

$$HoldsAt(Loaded(g), t) \Rightarrow \\ Terminates(Shoot(a_1, a_2, g), Loaded(g), t) \tag{5.18}$$

Consider the following observations and narrative. Initially, the turkey is alive and the gun is not loaded:

$$HoldsAt(Alive(Turkey), 0) \tag{5.19}$$
$$\neg HoldsAt(Loaded(Gun), 0) \tag{5.20}$$

Nathan loads the gun at timepoint 0, waits at timepoint 1, and shoots the turkey at timepoint 2:

$$Happens(Load(Nathan, Gun), 0) \tag{5.21}$$
$$Happens(Wait(Nathan), 1) \tag{5.22}$$
$$Happens(Shoot(Nathan, Turkey, Gun), 2) \tag{5.23}$$

We can then show that the turkey will no longer be alive at timepoint 3.

PROPOSITION 5.6

Let $\Sigma = (5.16) \wedge (5.17) \wedge (5.18)$, $\Delta = (5.21) \wedge (5.22) \wedge (5.23)$, $\Omega = U[Load, Wait, Shoot] \wedge U[Loaded, Alive]$, and $\Gamma = (5.19) \wedge (5.20)$, and $D2 = (5.10) \wedge (5.11) \wedge (5.14) \wedge (5.15)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ \Omega \wedge \Gamma \wedge D2 \models \neg HoldsAt(Alive(Turkey), 3)$$

Proof See Exercise 3. ■

5.2.2 Releasing from Inertia

By contrast, consider a gun that is loaded at timepoint 1 and whose chamber is spun at timepoint 2. If the trigger is pulled at timepoint 3, then the gun may or may

Notice that given axioms (5.15), (5.27), and (5.29), axiom (5.18) ensures that, after a loaded gun is fired, the gun is no longer loaded and *Loaded*(*Gun*) is no longer released from the commonsense law of inertia.

The fluent term *Alive*(*Turkey*) is subject to the commonsense law of inertia at all times:

$$\neg ReleasedAt(Alive(Turkey), t) \tag{5.32}$$

Only initially is *Loaded*(*Gun*) subject to the commonsense law of inertia:

$$\neg ReleasedAt(Loaded(Gun), 0) \tag{5.33}$$

We can then show that there are two possible outcomes: (1) If the gun is loaded at timepoint 2, then the turkey will no longer be alive at timepoint 3, and (2) if the gun is not loaded at timepoint 2, then the turkey will still be alive at timepoint 3.

PROPOSITION 5.7

Let $\Sigma = (5.16) \wedge (5.17) \wedge (5.18) \wedge (5.30)$, $\Delta = (5.21) \wedge (5.31) \wedge (5.23)$, $\Omega = U[Load, Spin, Shoot] \wedge U[Loaded, Alive]$, and $\Gamma = (5.19) \wedge (5.20) \wedge (5.32) \wedge (5.33)$, and D3 = $(5.10) \wedge (5.15) \wedge (5.24) \wedge (5.25) \wedge (5.26) \wedge (5.27) \wedge (5.28) \wedge (5.29)$. Then we have

$$\begin{aligned} CIRC[\Sigma; Initiates, Terminates, Releases] \wedge \\ CIRC[\Delta; Happens] \wedge \Omega \wedge \Gamma \wedge \mathrm{D3} \wedge HoldsAt(Loaded(Gun), 2) \\ \models \neg HoldsAt(Alive(Turkey), 3) \end{aligned} \tag{5.34}$$

as well as

$$\begin{aligned} CIRC[\Sigma; Initiates, Terminates, Releases] \wedge \\ CIRC[\Delta; Happens] \wedge \Omega \wedge \Gamma \wedge \mathrm{D3} \wedge \neg HoldsAt(Loaded(Gun), 2) \\ \models HoldsAt(Alive(Turkey), 3) \end{aligned} \tag{5.35}$$

Proof See Exercise 4. ■

In the Russian turkey scenario, release from the commonsense law of inertia is used to model nondeterminism. Nondeterminism is discussed further in Chapter 9. Release from the commonsense law of inertia is also useful for representing indirect effects (discussed in Chapter 6) and continuous change (discussed in Chapter 7).

5.3 Release Axioms

A fluent is released from the commonsense law of inertia as follows.

DEFINITION 5.1

If γ is a condition, α is an event term, β is a fluent term, and τ is a timepoint term, then

$$\gamma \Rightarrow Releases(\alpha, \beta, \tau)$$

is a ***release axiom***. This represents that, if γ is true and α occurs at τ, then β will be released from the commonsense law of inertia after τ.

A fluent is again made subject to the commonsense law of inertia as follows. We represent that, if γ is true and α occurs at τ, then β will no longer be released from the commonsense law of inertia after τ using a positive or negative effect axiom:

$$\gamma \Rightarrow Initiates(\alpha, \beta, \tau)$$
$$\gamma \Rightarrow Terminates(\alpha, \beta, \tau)$$

In the *Inititates* case, the fluent will become true and not released; in the *Terminates* case, the fluent will become false and not released.

In EC, a fluent is released for times greater than the time of the releasing event and is not released for times greater than the time of the initiating or terminating event. In DEC, a fluent is released starting one timepoint after the time of the releasing event and is not released starting one timepoint after the time of the initiating or terminating event. For example, suppose we have $Releases(Rel, Fluent, t)$, $Happens(Rel, 1)$, $Initiates(Init, Fluent, t)$, and $Happens(Init, 4)$. Figure 5.1 shows when *Fluent* is released using the conjunction of axioms EC; Figure 5.2 shows when *Fluent* is released using DEC.

Bibliographic Notes

Frame Problem

The frame problem was first described by McCarthy and Hayes (1969, p. 487). According to Shanahan (1997b), "McCarthy relates that he was reading a book on geometry at the time he coined the term 'frame problem', and that he thought of

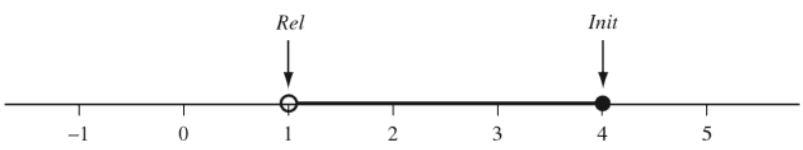

Figure 5.1 Released fluent in EC.

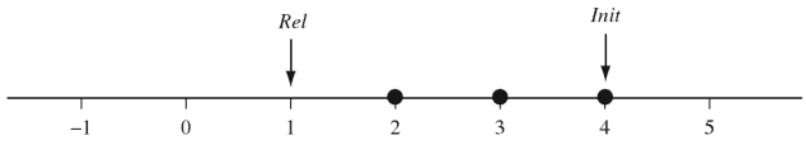

Figure 5.2 Released fluent in DEC.

the frame problem as analogous to that of choosing a co-ordinate frame" (p. 25). Hayes (1971) defines a frame as "a classification of statements into groups which are independent in the sense that an action may alter members of one group without affecting any of the other groups" (p. 497). See also the discussions of McCarthy (1977, p. 1040) and Lifschitz (1990b, p. 366). An introduction to the frame problem is provided by Shanahan (2002), and a modern book-length treatment is provided by Shanahan (1997b). Earlier book-length discussions are provided by Brown (1987), Pylyshyn (1987), and Ford and Pylyshyn (1996).

Frame Axioms

Frame axioms were introduced by McCarthy and Hayes (1969, pp. 484–485). Hayes (1971) was apparently the first to call them " 'frame' axioms" (p. 514). We use the term *classical frame axiom* from Kautz and Selman (1996, p. 1197) to distinguish the frame axioms of McCarthy and Hayes from explanation closure axioms. Classical frame axioms were originally written in the situation calculus. Using Reiter's notation, a sample classical frame axiom is:

$$on(d_2, s) \wedge d_1 \neq d_2 \supset on(d_2, do(turn\_off(d_1), s))$$

Using Shanahan's notation, the same axiom is written as:

$$Holds(On(d_2), s) \wedge d_1 \neq d_2 \rightarrow Holds(On(d_2), Result(TurnOff(d_1), s))$$

To handle e events and f fluents, on the order of $2 \cdot e \cdot f$ classical frame axioms are required (Reiter, 2001, p. 22).

Kowalski (1974; 1979, pp. 133–146) introduces a way of combining classical frame axioms that relies on representing that fluent $F(x_1, \ldots, x_n)$ is true in situation σ as $Holds(F(x_1, \ldots, x_n), \sigma)$ rather than $F(x_1, \ldots, x_n, \sigma)$. See the discussions of Nilsson (1980, pp. 311–315) and Shanahan (1997b, pp. 231–241). Using Shanahan's notation, suppose we have the effect axiom:

$$\neg Holds(On(d), Result(TurnOff(d), s))$$

We can then use a single frame axiom:

$$Holds(f, s) \wedge f \neq On(d) \rightarrow Holds(f, Result(TurnOff(d), s))$$

instead of several frame axioms:

$$Holds(On(d_2), s) \wedge d_1 \neq d_2 \rightarrow Holds(On(d_2), Result(TurnOff(d_1), s))$$
$$Holds(Broken(d_1), s) \rightarrow Holds(Broken(d_1), Result(TurnOff(d_1), s))$$
$$\vdots$$

Explanation closure axioms were first proposed by Haas (1987), who called them "domain-specific frame axioms" (p. 343). They were further developed and named "explanation-closure" axioms by Schubert (1990, p. 25). E. Davis (1990) proposed similar axioms for "framing primitive events by fluents" (p. 206). Using Reiter's situation calculus notation, a sample explanation closure axiom is:

$$on(d, s) \wedge \neg on(d, do(a, s)) \supset a = turn\_off(d)$$

which corresponds to the single effect axiom:

$$\neg on(d, do(turn\_off(d), s))$$

Pednault (1989) proposed two constraints to facilitate the generation of classical frame axioms: (1) separate effect axioms must be written for each fluent, and (2) the effects of actions must be completely specified by the effect axioms.

Synthesizing the proposals of Haas, Schubert, E. Davis, and Pednault, Reiter (1991; 2001, pp. 28–32) provided a method for automatically constructing explanation closure axioms given a set of effect axioms. To handle f fluents, on the order of $2 \cdot f$ explanation closure axioms are required (Reiter, 2001, p. 27). Explanation closure axioms were first used in the event calculus by Shanahan and Witkowski (2004). Axioms DEC5, DEC6, DEC7, and DEC8, which resemble explanation closure axioms extended to allow fluents to be released from the commonsense law of inertia, were introduced by Mueller (2004a). Our review of classical frame axioms and explanation closure axioms is loosely based on that of Ernst, Millstein, and Weld (1997, pp. 1170–1171). The form of explanation closure axiom we give,

$$HoldsAt(F, t) \wedge \neg Happens(E_1, t) \wedge \cdots \wedge \neg Happens(E_n, t) \Rightarrow$$
$$HoldsAt(F, t + 1)$$

is logically equivalent to the form usually given:

$$HoldsAt(F, t) \wedge \neg HoldsAt(F, t + 1) \Rightarrow$$
$$Happens(E_1, t) \vee \cdots \vee Happens(E_n, t)$$

Shanahan (1996, pp. 684–685; 1997b, pp. 315–330) introduced the forced separation version of the event calculus in which *Inititates*, *Terminates*, and *Releases* are circumscribed separately from *Happens* and the observation (*HoldsAt*) formulas and event calculus axioms are outside the scope of any circumscription.

The technique of forced separation derives from the following previous proposals: (1) the filtered preferential entailment or filtering of Sandewall (1989b; 1994, pp. 213–215, 242–243), in which minimization is applied to effect axioms ("action laws") but not to observation formulas, which in turn derives from the preferential entailment of (Shoham 1988, p. 76); (2) the proposal of Crawford and Etherington (1992) to separate the description of the system from the observations; (3) the extension of Sandewall's techniques by Doherty and Łukaszewicz (1994) and Doherty (1994), in which circumscription is applied to schedule statements involving the *Occlude* predicate but not to observation statements or to the nochange axiom, as discussed in Section 15.2.1; (4) the method of Kartha and Lifschitz (1995), in which circumscription is applied to effect axioms and state constraints but not to observation formulas (Shanahan, 1997b, pp. 315–318); and (5) the method of Lin (1995), in which the *Caused* predicate is minimized using circumscription or predicate completion.

E. Davis (1990) discusses the need for axioms of "nonoccurrence of extraneous events" (p. 208) when using explanation closure axioms. The rationale for the use of circumscription over less powerful methods is given by Shanahan (1998b):

We could use negation-as-failure (or rather, say, predicate completion). Using circumscription does allow for the addition of, for example, disjunctive facts, however. Predicate completion is only defined for a certain class of theories. Event [*sic*] though this class encompasses most of what we're interested in, there doesn't seem any point in ruling out exceptions. (p. 329)

Commonsense Law of Inertia

The phrase *commonsense law of inertia* was originated by John McCarthy (personal communication, May 18, 2005; Lifschitz, 1987c, p. 186). The phrase appears to have been first used in print by Lifschitz (1987a, p. 45; 1987c, p. 186). Hanks and McDermott (1987) mention the "inertia of the world" (p. 395) and attribute the phrase "inertial property of facts" (p. 394) to John McCarthy. Fluents subject to the commonsense law of inertia are sometimes called "frame fluents" (Lifschitz, 1990b, p. 370; R. Miller and Shanahan, 2002, p. 471) or are said to be "in the frame" (Lifschitz, 1990b, p. 369; R. Miller and Shanahan, 2002, p. 474) or to "belong to the frame" (R. Miller and Shanahan, 1996, p. 67).

Yale Shooting Scenario

The Yale shooting scenario was introduced by Hanks and McDermott (1985; 1986; 1987, pp. 387–390), who use the scenario to point out problems with McCarthy's (1984a, 1986) initial attempt at solving the frame problem using circumscription. The scenario and the various treatments of it are discussed at length by Shanahan

CHAPTER 6

Indirect Effects of Events

Suppose that a book is sitting on a table in a living room and an agent is in the living room. Normally, when the agent walks out of the room, the book remains in the living room; but, if the agent picks up the book and walks out of the living room, then the book is no longer in the living room. That is, an indirect effect or ramification of the agent walking out of the living room is that the book the agent is holding changes location. The problem of representing and reasoning about the indirect effects of events is known as the *ramification problem*. This chapter presents several methods for representing indirect effects and dealing with the ramification problem in the event calculus. We discuss the use of effect axioms, primitive and derived fluents, release axioms and state constraints, effect constraints, causal constraints, and trigger axioms.

6.1 Effect Axioms

One way of representing indirect effects is to represent them the same way that direct effects are represented, namely, using positive and negative effect axioms.

6.1.1 Example: Carrying a Book

An agent picks up a book; the book then moves along with the agent. We start with the following spatial theory. If an agent walks from room r_1 to room r_2, then the agent will be in r_2 and will no longer be in r_1:

$$Initiates(Walk(a, r_1, r_2), InRoom(a, r_2), t) \tag{6.1}$$

$$r_1 \neq r_2 \Rightarrow Terminates(Walk(a, r_1, r_2), InRoom(a, r_1), t) \tag{6.2}$$

An object is in one room at a time:

$$HoldsAt(InRoom(o, r_1), t) \wedge HoldsAt(InRoom(o, r_2), t) \Rightarrow r_1 = r_2 \tag{6.3}$$

If an agent is in the same room as an object and the agent picks up the object, then the agent will be holding the object:

$$HoldsAt(InRoom(a, r), t) \wedge HoldsAt(InRoom(o, r), t) \Rightarrow Initiates(PickUp(a, o), Holding(a, o), t) \tag{6.4}$$

If an agent is holding an object and the agent lets go of the object, then the agent will no longer be holding the object:

$$HoldsAt(Holding(a, o), t) \Rightarrow Terminates(LetGoOf(a, o), Holding(a, o), t) \tag{6.5}$$

We then represent the indirect effects of walking while holding an object using positive and negative effect axioms. If an agent is holding an object and the agent walks from room r_1 to room r_2, then the object will be in r_2 and will no longer be in r_1:

$$HoldsAt(Holding(a, o), t) \Rightarrow Initiates(Walk(a, r_1, r_2), InRoom(o, r_2), t) \tag{6.6}$$

$$HoldsAt(Holding(a, o), t) \wedge r_1 \neq r_2 \Rightarrow Terminates(Walk(a, r_1, r_2), InRoom(o, r_1), t) \tag{6.7}$$

Now consider the following observations and narrative. Nathan and the book start out in the living room:

$$\neg ReleasedAt(f, t) \tag{6.8}$$

$$HoldsAt(InRoom(Nathan, LivingRoom), 0) \tag{6.9}$$

$$HoldsAt(InRoom(Book, LivingRoom), 0) \tag{6.10}$$

Nathan picks up the book and walks into the kitchen:

$$Happens(PickUp(Nathan, Book), 0) \tag{6.11}$$

$$Happens(Walk(Nathan, LivingRoom, Kitchen), 1) \tag{6.12}$$

We also have

$$LivingRoom \neq Kitchen \tag{6.13}$$

We can then show that the book will be in the kitchen.

PROPOSITION 6.1

Let $\Sigma = (6.1) \wedge (6.2) \wedge (6.4) \wedge (6.5) \wedge (6.6) \wedge (6.7)$, $\Delta = (6.11) \wedge (6.12)$, $\Omega = U[Walk, PickUp, LetGoOf] \wedge U[InRoom, Holding] \wedge (6.13)$, $\Psi = (6.3)$, and $\Gamma = (6.8) \wedge (6.9) \wedge (6.10)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Psi \wedge \Gamma \wedge \mathrm{EC} \models HoldsAt(InRoom(Book, Kitchen), 2)$$

Proof From $CIRC[\Sigma; Initiates, Terminates, Releases]$, Theorem 2.2, and Theorem 2.1, we have

$$\begin{aligned}
&Initiates(e, f, t) \Leftrightarrow && (6.14)\\
&\exists a, r_1, r_2\,(e = Walk(a, r_1, r_2) \wedge f = InRoom(a, r_2)) \vee\\
&\exists a, o, r\,(e = PickUp(a, o) \wedge\\
&f = Holding(a, o) \wedge\\
&HoldsAt(InRoom(a, r), t) \wedge\\
&HoldsAt(InRoom(o, r), t)) \vee\\
&\exists a, o, r_1, r_2\,(e = Walk(a, r_1, r_2) \wedge\\
&f = InRoom(o, r_2) \wedge\\
&HoldsAt(Holding(a, o), t))\\
&Terminates(e, f, t) \Leftrightarrow && (6.15)\\
&\exists a, r_1, r_2\,(e = Walk(a, r_1, r_2) \wedge\\
&f = InRoom(a, r_1) \wedge\\
&r_1 \neq r_2) \vee\\
&\exists a, o\,(e = LetGoOf(a, o) \wedge\\
&f = Holding(a, o) \wedge\\
&HoldsAt(Holding(a, o), t)) \vee\\
&\exists a, o, r_1, r_2\,(e = Walk(a, r_1, r_2) \wedge\\
&f = InRoom(o, r_1) \wedge\\
&HoldsAt(Holding(a, o), t) \wedge\\
&r_1 \neq r_2)\\
&\neg Releases(e, f, t) && (6.16)
\end{aligned}$$

From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have

$$\begin{aligned}
&Happens(e, t) \Leftrightarrow && (6.17)\\
&(e = PickUp(Nathan, Book) \wedge t = 0) \vee\\
&(e = Walk(Nathan, LivingRoom, Kitchen) \wedge t = 1)
\end{aligned}$$