ERIK T. MUELLER

COMMONSENSE REASONING

MK

About the Author

Erik T. Mueller is Research Staff Member at the IBM Thomas J. Watson Research Center. He is the author of *Daydreaming in Humans and Machines* (Ablex, 1990) on a computer model of the stream of thought. He received an S.B. in Computer Science and Engineering from the Massachusetts Institute of Technology, and an M.Sc. and Ph.D. in Computer Science from the University of California, Los Angeles. His current research focuses on commonsense reasoning and its application to story understanding and intelligent user interfaces. See his web site at www.signiform.com/erik/ for more information.

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 2 and Appendix A.

Roadmap

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

Part I 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.

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 $C+$ 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:

$$\begin{gathered}(Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge t_1 < t_2 \wedge \\ \neg \exists e, t\,(Happens(e, t) \wedge t_1 < t \wedge t < t_2 \wedge Terminates(e, f, t))) \Rightarrow \\ HoldsAt(f, t_2)\end{gathered} \tag{1.1}$$

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:

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

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

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

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).

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 v is a variable and γ is a condition, then $\exists v\, \gamma$ 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] α, $\alpha_1, \ldots, \alpha_n$ = event terms; β, $\beta_1, \ldots, \beta_n$ = fluent terms, γ, γ_1, γ_2 = conditions, π, $\pi_1, \ldots, \pi_n$ = *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[Σ; *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[Δ; *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[Σ; *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[Δ; *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 $\Phi =$

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

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

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

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:

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

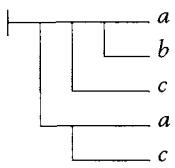
E. Davis (1990, p. 153) shows that Allen's (1983) relations over intervals can be defined in terms of $Start$ and End. For example:

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

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

Table 2.3 Monotonicity of logic

(Frege, 1879/1967, p. 43)	
(Whitehead and Russell, 1925, p. 113)	$\vdash :. p \supset r . \supset : p . q . \supset . r$
(Gentzen, 1935/1969, p. 83)	$\dfrac{\Gamma \rightarrow \Theta}{\mathcal{D}, \Gamma \rightarrow \Theta}$
(Kleene, 1952, p. 89)	*If* $\Gamma \vdash$ E, *then* C, $\Gamma \vdash$ E
(M. Davis, 1959, p. 1)	If $A \vdash \alpha$ and $A \subset B$ then $B \vdash \alpha$

proves other theorems" (p. 217). This property of logic was well known, as shown in Table 2.3. Pratt recalls the following:

> Marvin Minsky (whose office on the 8th floor of 545 Tech Square was across the AI playroom from mine) ran into me as I was walking out of my office into the playroom and asked me whether logicians had a name for the notion that one rule of inference could turn off another, or something to that effect. I said I'd never heard of such a thing, but that logical systems normally were monotonic in the sense that if you added an axiom or rule you could only get more theorems, so that the sort of system he was contemplating would be nonmonotonic. Neither of us had anything to add to this and that was the last I heard about any of that for a couple of years until I started seeing the phrase "nonmonotonic logic" in the literature (personal communication, June 9, 2005).

Minker (1993) provides an overview of nonmonotonic reasoning. Book-length treatments of nonmonotonic reasoning are provided by Gabbay, Hogger, and Robinson (1994); Antoniou (1997); and Brewka, Dix, and Konolige (1997). A collection of readings on nonmonotonic reasoning is provided by Ginsberg (1987). Nonmonotonic logics include default logic (Reiter, 1980b), the nonmonotonic logics of McDermott and Doyle (1980), and autoepistemic logic (R. C. Moore, 1985). Earlier nonmonotonic reasoning methods include the THNOT of PLANNER (Hewitt, 1972), the Unless operator of Sandewall (1972), and negation as failure (K. L. Clark, 1978).

Circumscription

McCarthy (1977, pp. 1040–1041; 1980; 1984a; 1986) introduced the nonmonotonic reasoning method of circumscription. The definitive treatment of

circumscription is provided by Lifschitz (1994). Reiter (1982) studies the relationship of circumscription and predicate completion. Methods for computing circumscription are described by Lifschitz (1985, 1994) and Genesereth and Nilsson (1987, pp. 132–152). Computational aspects of circumscription are explored by Kolaitis and Papadimitriou (1990). Algorithms for computing circumscription are provided by Gabbay and Ohlbach (1992) and Doherty, Łukaszewicz, and Szałas (1997). Not all circumscriptions reduce to formulas of first-order logic. Lifschitz (1985, p. 123) gives the example of using circumscription to assert that the binary relation denoted by a predicate is the transitive closure of the binary relation denoted by another predicate, which is not first-order expressible (Aho and Ullman, 1979, pp. 119–120; Nerode and Shore, 1997, p. 126). See also the discussions of McCarthy (1980, pp. 33–34) and Genesereth and Nilsson (1987, pp. 144–145). Predicate completion was introduced by K. L. Clark (1978, pp. 303–305). Genesereth and Nilsson (1987, pp. 122–127) and Russell and Norvig (2003, p. 355) explain how to compute predicate completion.

Formalizing Domains

E. Davis (1999) provides a useful guide to formalizing domains in first-order logic. Elaboration tolerance is discussed by McCarthy (1998a) and Parmar (2003).

Writing Proofs

Velleman (1994), Garnier and Taylor (1996), and Solow (2002) describe techniques for discovering and writing proofs. Reiter (2001, pp. 389–400) provides rules for finding proofs.

Reasoning Types

Temporal projection is discussed by Wilensky (1983), Hanks and McDermott (1987), and Dean and Boddy (1988). Lifschitz (1989) discusses a number of different types of commonsense reasoning, such as "temporal projection" (p. 207) and "temporal explanation with unknown initial conditions" (p. 208), and provides a benchmark problem for each type of reasoning. Abduction is discussed by Charniak and McDermott (1985, chap. 8), Shanahan (1989; 1997b, chap. 17; 2000a), Denecker, Missiaen, and Bruynooghe (1992), and Hobbs, Stickel, Appelt, and Martin (1993). Reiter (1996, p. 12) points out some drawbacks of abductive reasoning. Postdiction is discussed by E. Giunchiglia, Lee, Lifschitz, McCain, and Turner (2004, p. 65).

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.

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*(α, β, τ) represents that, if an event α occurs at timepoint τ, then fluent β will be true after τ. The predicate *Terminates*(α, β, τ) 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

$$\gamma \Rightarrow \mathit{Initiates}(\alpha, \beta, \tau)$$

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

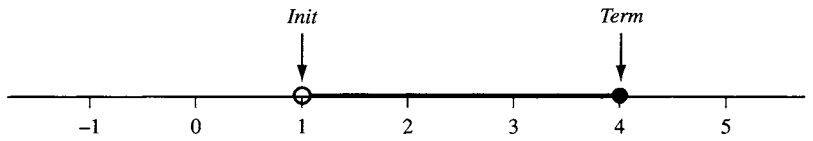


Figure 3.1 Truth value of a fluent in EC.

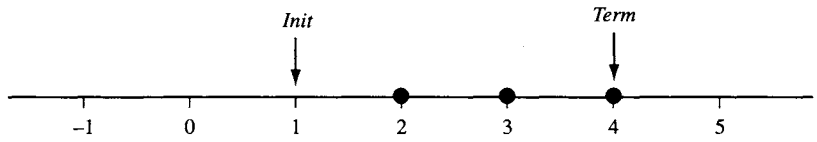


Figure 3.2 Truth value of a fluent in DEC.

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

$$\gamma \Rightarrow \mathit{Terminates}(\alpha, \beta, \tau)$$

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.

If we pick up an idle phone, we expect to get a dial tone. But if we pick up a phone that is ringing, we expect to be connected to the caller.

We can represent our knowledge about telephones using positive and negative effect axioms. If an agent picks up an idle phone, the phone will have a dial tone and will no longer be idle:

$$HoldsAt(Idle(p), t) \Rightarrow Initiates(PickUp(a, p), DialTone(p), t) \tag{3.1}$$

$$HoldsAt(Idle(p), t) \Rightarrow Terminates(PickUp(a, p), Idle(p), t) \tag{3.2}$$

After picking up a phone, an agent may decide not to place a call. If an agent sets down a phone with a dial tone, the phone will be idle and will no longer have a dial tone:

$$HoldsAt(DialTone(p), t) \Rightarrow Initiates(SetDown(a, p), Idle(p), t) \tag{3.3}$$

$$HoldsAt(DialTone(p), t) \Rightarrow Terminates(SetDown(a, p), DialTone(p), t) \tag{3.4}$$

When phone p_1 has a dial tone and an agent dials phone p_2 from p_1, what happens depends on the state of p_2. If p_2 is idle, then p_1 will be ringing p_2, p_1 will no longer have a dial tone, and p_2 will no longer be idle:

$$HoldsAt(DialTone(p_1), t) \wedge HoldsAt(Idle(p_2), t) \Rightarrow Initiates(Dial(a, p_1, p_2), Ringing(p_1, p_2), t) \tag{3.5}$$

$$HoldsAt(DialTone(p_1), t) \wedge HoldsAt(Idle(p_2), t) \Rightarrow Terminates(Dial(a, p_1, p_2), DialTone(p_1), t) \tag{3.6}$$

$$HoldsAt(DialTone(p_1), t) \wedge HoldsAt(Idle(p_2), t) \Rightarrow Terminates(Dial(a, p_1, p_2), Idle(p_2), t) \tag{3.7}$$

If p_2 is not idle, then p_1 will have a busy signal:

$$HoldsAt(DialTone(p_1), t) \wedge \neg HoldsAt(Idle(p_2), t) \Rightarrow Initiates(Dial(a, p_1, p_2), BusySignal(p_1), t) \tag{3.8}$$

$$HoldsAt(DialTone(p_1), t) \wedge \neg HoldsAt(Idle(p_2), t) \Rightarrow Terminates(Dial(a, p_1, p_2), DialTone(p_1), t) \tag{3.9}$$

If an agent sets down a phone with a busy signal, then the phone will be idle and will no longer have a busy signal:

$$HoldsAt(BusySignal(p), t) \Rightarrow \\ Initiates(SetDown(a, p), Idle(p), t) \tag{3.10}$$

$$HoldsAt(BusySignal(p), t) \Rightarrow \\ Terminates(SetDown(a, p), BusySignal(p), t) \tag{3.11}$$

A call may go unanswered. If phone p_1 is ringing phone p_2 and an agent sets down p_1, then p_1 will be idle, p_2 will be idle, and p_1 will no longer be ringing p_2:

$$HoldsAt(Ringing(p_1, p_2), t) \Rightarrow \\ Initiates(SetDown(a, p_1), Idle(p_1), t) \tag{3.12}$$

$$HoldsAt(Ringing(p_1, p_2), t) \Rightarrow \\ Initiates(SetDown(a, p_1), Idle(p_2), t) \tag{3.13}$$

$$HoldsAt(Ringing(p_1, p_2), t) \Rightarrow \\ Terminates(SetDown(a, p_1), Ringing(p_1, p_2), t) \tag{3.14}$$

A call may be answered. If phone p_1 is ringing phone p_2 and an agent picks up p_2, then p_1 will be connected to p_2 and p_1 will no longer be ringing p_2:

$$HoldsAt(Ringing(p_1, p_2), t) \Rightarrow \\ Initiates(PickUp(a, p_2), Connected(p_1, p_2), t) \tag{3.15}$$

$$HoldsAt(Ringing(p_1, p_2), t) \Rightarrow \\ Terminates(PickUp(a, p_2), Ringing(p_1, p_2), t) \tag{3.16}$$

A call may be completed. If phone p_1 is connected to phone p_2 and an agent sets down p_1, then p_1 will be idle, p_2 will be disconnected, and p_1 will no longer be connected to p_2:

$$HoldsAt(Connected(p_1, p_2), t) \Rightarrow \\ Initiates(SetDown(a, p_1), Idle(p_1), t) \tag{3.17}$$

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

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

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

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

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

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

$$\exists a, p_1, p_2\,(e = SetDown(a, p_1) \wedge f = Ringing(p_1, p_2) \wedge$$
$$HoldsAt(Ringing(p_1, p_2), t)) \vee$$
$$\exists a, p_1, p_2\,(e = PickUp(a, p_2) \wedge$$
$$f = Ringing(p_1, p_2) \wedge$$
$$HoldsAt(Ringing(p_1, p_2), t)) \vee$$
$$\exists a, p_1, p_2\,(e = SetDown(a, p_1) \wedge$$
$$f = Connected(p_1, p_2) \wedge$$
$$HoldsAt(Connected(p_1, p_2), t)) \vee$$
$$\exists a, p_1, p_2\,(e = SetDown(a, p_2) \wedge$$
$$f = Connected(p_1, p_2) \wedge$$
$$HoldsAt(Connected(p_1, p_2), t)) \vee$$
$$\exists a, p\,(e = SetDown(a, p) \wedge$$
$$f = Disconnected(p) \wedge$$
$$HoldsAt(Disconnected(p), t))$$

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

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

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

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:

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

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

Functional Modification

An event modifies a value by applying some function. If the value is v and the function F is applied, then the value will be $F(v)$:

$$HoldsAt(Value(o, v), t) \Rightarrow$$
$$Initiates(ApplyF(o), Value(o, F(v)), t)$$

$$HoldsAt(Value(o, v), t) \wedge v \neq F(v) \Rightarrow$$
$$Terminates(ApplyF(o), Value(o, v), t)$$

A common example is incrementing a value:

$$HoldsAt(Value(o, v), t) \Rightarrow Initiates(Increment(o), Value(o, v + 1), t)$$
$$HoldsAt(Value(o, v), t) \Rightarrow Terminates(Increment(o), Value(o, v), t)$$

Many-to-Many Mapping

In general, the mapping between events and fluents is many-to-many. One event may initiate (or terminate) many fluents:

$$Initiates(E(o), F1(o), t)$$
$$Initiates(E(o), F2(o), t)$$

Many events may initiate (or terminate) one fluent:

$$Initiates(E1(o), F(o), t)$$
$$Initiates(E2(o), F(o), t)$$

3.3 Preconditions

Consider the action of turning on a device. We might represent this as follows:

$$Initiates(TurnOn(a, d), On(d), t)$$

That is, if an agent turns on a device, then the device will be on. But there are many things that may prevent the device from going on. The device might be unplugged, it might be broken, its on-off switch might be broken, and so on. A condition that prevents an event from having its intended effects, or that prevents an event from occurring, is called a *qualification*. An event is said to be qualified whenever one or more of its qualifications are true. The problem of representing and reasoning about qualifications is known as the *qualification problem*.

A partial solution to the qualification problem is to use preconditions. (Other solutions are to use state constraints, which are described in Section 3.4, and default reasoning, which is discussed in Chapter 12.) In this section we describe two types of preconditions for events: fluent preconditions and action preconditions. Fluent preconditions allow us to represent qualifications that prevent events from having their intended effects; action preconditions allow us to represent qualifications that prevent events from occurring.

3.3.1 Fluent Preconditions

A *fluent precondition* is a requirement that must be satisfied for the occurrence of an event to have an effect. An event whose fluent precondition is not satisfied may occur, but the event will not have the intended effect.

DEFINITION 3.3

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

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

and

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

are ***fluent precondition axioms***. We say that γ is a fluent precondition of α. A fluent precondition axiom is the same thing as an effect axiom.

For example, if an agent turns on a device, then, provided the device is not broken, the device will be on:

$$\neg HoldsAt(Broken(d), t) \Rightarrow Initiates(TurnOn(a, d), On(d), t)$$

If a device is broken, then the event of turning on the device can occur, but it will not have the intended effect.

3.3.2 Action Preconditions

An *action precondition* is a requirement that must be satisfied for the occurrence of an event. An event whose action precondition is not satisfied cannot occur.

DEFINITION 3.4

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

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

is an ***action precondition axiom***. We say that γ is an action precondition of α.

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:

$$Happens(WalkThroughDoor(a, d), t) \Rightarrow HoldsAt(Near(a, d), t) \quad (3.41)$$

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

$$\neg HoldsAt(Near(Nathan, Door), 1) \quad (3.42)$$
$$Happens(WalkThroughDoor(Nathan, Door), 1) \quad (3.43)$$

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'.

The subgoal to apply o to a is achieved as follows:

1. If a is not of the required form c for the application of o, then
   (a) Find a difference d between a and c.
   (b) Invoke subgoal to reduce d between a and c, producing a'.
   (c) Recursively invoke subgoal to apply o to a', producing a''.
2. Otherwise, apply o to a, producing a'.

GPS evolved into the SOAR problem-solving and learning architecture (Rosenbloom, Laird, and Newell, 1993).

QA3

An early program for reasoning about action and change was QA3 (Green, 1969). The program uses a version of the situation calculus to represent knowledge and uses resolution theorem proving (Robinson, 1965) to solve planning problems such as McCarthy's (1963) monkey and bananas problem.

STRIPS

Another early program for reasoning about action and change was STRIPS (Fikes and Nilsson, 1971; Fikes, Hart, and Nilsson, 1972a, 1972b). STRIPS solves planning problems. It takes a set of operator descriptions, a goal formula, and an initial world model given by a set of formulas and produces a sequence of operators that transforms the initial world model into a model in which the goal is true. An operator description consists of a precondition, a delete list, and an add list. The precondition specifies the conditions under which the operator is applicable, the delete list specifies the formulas that are deleted by the operator, and the add list specifies the formulas that are added by the operator. STRIPS deals with the commonsense law of inertia by making the assumption that, when an operator is applied, any formulas not deleted or added by the operator stay the same. This is called the STRIPS assumption by Waldinger (1977, p. 120). Lifschitz (1987b) formalizes the semantics of STRIPS. A STRIPS operator description

Operator: o
Precondition: c
Delete list: $d_1, \ldots, d_n$
Add list: $a_1, \ldots, a_n$

is similar to the set of event calculus axioms

$$c \Rightarrow \mathit{Terminates}(o, d_1, t)$$
$$\vdots$$
$$c \Rightarrow \mathit{Terminates}(o, d_n, t)$$

$$c \Rightarrow Initiates(o, a_1, t)$$
$$\vdots$$
$$c \Rightarrow Initiates(o, a_n, t)$$

Inconsistency

In EC and DEC, simultaneously initiating and terminating a fluent produces inconsistency. This is also the case in most of the versions of the classical logic event calculus (Shanahan, 1996, 1997b, 1999a, 2004). But in the basic axiomatization of the event calculus of R. Miller and Shanahan (1999, pp. 1–3; 2002, pp. 453–455) (not the particular version of R. Miller and Shanahan used in this book), simultaneously initiating and terminating a fluent results in two classes of models: one in which the fluent is true and one in which the fluent is false. This can be used to represent nondeterminism. R. Miller and Shanahan (2002, pp. 459–460) give the following example:

$$Initiates(TossCoin, HeadsUp, t)$$
$$Terminates(TossCoin, HeadsUp, t)$$
$$Happens(TossCoin, 2)$$

(Nondeterminism can still be represented with EC and DEC using release axioms as discussed in Section 5.2, determining fluents as discussed in Section 9.1, or disjunctive event axioms as discussed in Section 9.2.)

Qualification Problem

The qualification problem was first described by McCarthy and Hayes (1969, p. 489). They considered the problem of how to represent exceptions to statements about the effects of looking up and dialing a person's telephone number. The phrase "qualification problem" appears to have been first used in print by Hayes (1973, p. 56). He uses the example of a robot that concludes that it can drive to the airport and then notices a flat tire. McCarthy (1977, p. 1040) presents the qualification problem and discusses the countless ways a boat could be prevented from crossing a river in the missionaries and cannibals problem. The qualification problem is also discussed by McCarthy (1980, pp. 27–28; 1987, p. 1033) and Hayes (1971, pp. 515–516). Our definitions of qualifications, qualified events, and the qualification problem are from Kvarnström and Doherty (2000a). Ginsberg and Smith (1987b, 1988b) propose the use of state constraints for representing qualifications, and Lin and Reiter (1994, section 2) call these "qualification constraints."

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:

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

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

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

$$Initiates(TurnOnAlarm(a, c), AlarmOn(c), t) \quad (4.3)$$

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

$$Terminates(TurnOffAlarm(a, c), AlarmOn(c), t) \quad (4.4)$$

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

$$Initiates(StartBeeping(c), Beeping(c), t) \quad (4.5)$$

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

$$Terminates(TurnOffAlarm(a, c), Beeping(c), t) \quad (4.6)$$

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

$$HoldsAt(AlarmTime(c, t_1), t) \wedge HoldsAt(AlarmTime(c, t_2), t) \Rightarrow t_1 = t_2 \quad (4.7)$$

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:

$$HoldsAt(AlarmTime(c, t), t) \wedge HoldsAt(AlarmOn(c), t) \Rightarrow \quad (4.8)$$
$$Happens(StartBeeping(c), t)$$

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:

$$\neg HoldsAt(AlarmOn(Clock), 0) \quad (4.9)$$
$$\neg HoldsAt(Beeping(Clock), 0) \quad (4.10)$$
$$HoldsAt(AlarmTime(Clock, 10), 0) \quad (4.11)$$
$$\neg ReleasedAt(f, t) \quad (4.12)$$

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

$$Happens(SetAlarmTime(Nathan, Clock, 2), 0) \quad (4.13)$$
$$Happens(TurnOnAlarm(Nathan, Clock), 1) \quad (4.14)$$

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

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

$$HoldsAt(AlarmTime(c, t), t) \wedge$$
$$HoldsAt(AlarmOn(c), t)) \vee$$
$$(e = SetAlarmTime(Nathan, Clock, 2) \wedge t = 0) \vee$$
$$(e = TurnOnAlarm(Nathan, Clock) \wedge t = 1)$$

From (4.13) (which follows from (4.18)), (4.11), $10 \neq 2$, (4.1) (which follows from (4.15)), and DEC9, we have

$$HoldsAt(AlarmTime(Clock, 2), 1) \tag{4.19}$$

From (4.18) and (4.16), we have $\neg\exists e\,(Happens(e, 1) \wedge Terminates(e, AlarmTime(Clock, 2), 1))$. From this, (4.19), (4.12), and DEC5, we have

$$HoldsAt(AlarmTime(Clock, 2), 2) \tag{4.20}$$

From (4.14) (which follows from (4.18)), (4.3) (which follows from (4.15)), and DEC9, we have

$$HoldsAt(AlarmOn(Clock), 2) \tag{4.21}$$

From (4.20), (4.21), and (4.8) (which follows from (4.18)), we have $Happens(StartBeeping(Clock), 2)$. From this, (4.5) (which follows from (4.15)), and DEC9, we have $HoldsAt(Beeping(Clock), 3)$. ■

4.2 Preventing Repeated Triggering

As long as the condition of a trigger axiom is true, an event is triggered. We must take steps to ensure that events are not repeatedly triggered. In the example in Section 4.1.1, the trigger axiom (4.8) does not repeatedly trigger because the condition is true only at a single instant, when the alarm timepoint equals the current timepoint. In most other cases, we will require a method to prevent repeated triggering.

This may be accomplished using trigger axioms and effect axioms of the form

$$\neg HoldsAt(\beta, \tau) \wedge \gamma \Rightarrow Happens(\alpha, \tau)$$

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

or

$$HoldsAt(\beta, \tau) \wedge \gamma \Rightarrow Happens(\alpha, \tau)$$

$$Terminates(\alpha, \beta, \tau).$$

4.2.1 Example: Bank Account Service Fee

An example in which we prevent the repeated triggering of events in this way is the charging of service fees. We formalize a bank account that charges a monthly service fee if the balance falls below a minimum amount at any time during the month.

First, we describe the basic operation of the account using effect axioms. If the balance of account a_1 is greater than or equal to some amount and that amount is transferred from a_1 to account a_2, then the balance of a_2 increases by that amount and the balance of a_1 decreases by that amount:

$$\begin{gathered} HoldsAt(Balance(a_1, x_1), t) \wedge \\ HoldsAt(Balance(a_2, x_2), t) \wedge \\ x_3 > 0 \wedge x_1 \geq x_3 \Rightarrow \\ Initiates(Transfer(a_1, a_2, x_3), Balance(a_2, x_2 + x_3), t) \end{gathered} \tag{4.22}$$

$$\begin{gathered} HoldsAt(Balance(a_1, x_1), t) \wedge \\ HoldsAt(Balance(a_2, x_2), t) \wedge \\ x_3 > 0 \wedge x_1 \geq x_3 \Rightarrow \\ Terminates(Transfer(a_1, a_2, x_3), Balance(a_2, x_2), t) \end{gathered} \tag{4.23}$$

$$\begin{gathered} HoldsAt(Balance(a_1, x_1), t) \wedge \\ HoldsAt(Balance(a_2, x_2), t) \wedge \\ x_3 > 0 \wedge x_1 \geq x_3 \Rightarrow \\ Initiates(Transfer(a_1, a_2, x_3), Balance(a_1, x_1 - x_3), t) \end{gathered} \tag{4.24}$$

$$\begin{gathered} HoldsAt(Balance(a_1, x_1), t) \wedge \\ HoldsAt(Balance(a_2, x_2), t) \wedge \\ x_3 > 0 \wedge x_1 \geq x_3 \Rightarrow \\ Terminates(Transfer(a_1, a_2, x_3), Balance(a_1, x_1), t) \end{gathered} \tag{4.25}$$

We have a state constraint that says that an account has a unique balance at any given time:

$$\begin{gathered} HoldsAt(Balance(a, x_1), t) \wedge HoldsAt(Balance(a, x_2), t) \Rightarrow \\ x_1 = x_2 \end{gathered} \tag{4.26}$$

We also have a trigger axiom that says that if the balance in an account falls below the minimum balance and a service fee has not yet been charged to the

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

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

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:

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

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

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

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

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

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

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

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

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

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.

CHAPTER 5

The Commonsense Law of Inertia

A quality of the commonsense world is that objects tend to stay in the same state unless they are affected by events. A book sitting on a table remains on the table unless it is picked up, a light stays on until it is turned off, and a falling object continues to fall until it hits something. This is known as the commonsense law of inertia. This chapter discusses the representation of the commonsense law of inertia in the event calculus. We retrace the development of the DEC axioms, and we discuss the enforcement of the commonsense law of inertia and the release of fluents from inertia.

5.1 Representation of the Commonsense Law of Inertia

How should we represent the commonsense law of inertia? Let us consider a version of the event calculus with integer time, using the example of turning on a light. We first represent the effects of events; we represent that, if a light is turned on, then the light will be on at the next timepoint:

$$Happens(TurnOn(l), t) \Rightarrow HoldsAt(On(l), t + 1) \tag{5.1}$$

Suppose two lights are off at timepoint 0, and the second light is turned on at timepoint 0:

$$\neg HoldsAt(On(Light1), 0) \tag{5.2}$$
$$\neg HoldsAt(On(Light2), 0) \tag{5.3}$$
$$Happens(TurnOn(Light2), 0) \tag{5.4}$$
$$Light2 \neq Light1 \tag{5.5}$$

We can show that the second light will be on at timepoint 1.

PROPOSITION 5.1 If $\Sigma = (5.1)$, $\Delta = (5.4)$, $\Omega = (5.5)$, and $\Gamma = (5.2) \wedge (5.3)$, then $\Sigma \wedge \Delta \wedge \Omega \wedge \Gamma \models \mathit{HoldsAt}(\mathit{On}(\mathit{Light2}), 1)$.

Proof From (5.1) and (5.4), we have $\mathit{HoldsAt}(\mathit{On}(\mathit{Light2}), 1)$. ■

5.1.1 Frame Problem

What can we say about the first light? We would like to reason that, because we were not told that the first light was turned on, it will still be off: $\neg\mathit{HoldsAt}(\mathit{On}(\mathit{Light1}), 1)$. Sadly, this does not follow from the domain description $\Sigma \wedge \Delta \wedge \Omega \wedge \Gamma$. The trouble is that we have represented which fluents change truth value when an event occurs but we have not represented which fluents do not change truth value. The problem of representing and reasoning about which fluents do not change when an event occurs is known as the *frame problem*. Several types of axioms have been proposed for specifying when fluents do not change truth value.

5.1.2 Classical Frame Axioms

A *classical frame axiom* represents that a given fluent does not change when a given event occurs. For example, we represent that, if light l is off and a different light is turned on, then l will still be off:

$$\begin{gathered}\neg\mathit{HoldsAt}(\mathit{On}(l_1), t) \wedge \\ \mathit{Happens}(\mathit{TurnOn}(l_2), t) \wedge \\ l_1 \neq l_2 \Rightarrow \\ \neg\mathit{HoldsAt}(\mathit{On}(l_1), t+1)\end{gathered} \tag{5.6}$$

We can then show that after the second light is turned on, the first light is off.

PROPOSITION 5.2 If $\Sigma = (5.1)$, $\Delta = (5.4)$, $\Omega = (5.5)$, $\Gamma = (5.2) \wedge (5.3)$, and CFA = (5.6), then $\Sigma \wedge \Delta \wedge \Omega \wedge \Gamma \wedge \mathit{CFA} \models \neg\mathit{HoldsAt}(\mathit{On}(\mathit{Light1}), 1)$.

Proof From (5.2), (5.4), (5.5), and (5.6), we have $\neg\mathit{HoldsAt}(\mathit{On}(\mathit{Light1}), 1)$. ■

Unfortunately, classical frame axioms are unable to cope with several events occurring at one timepoint. For instance, if we are also given that the first light is turned on at timepoint 0:

$$\mathit{Happens}(\mathit{TurnOn}(\mathit{Light1}), 0) \tag{5.7}$$

then inconsistency results.

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[Δ; *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

not fire (the chamber may still be spinning). We may model this by specifying that, after the gun's chamber is spun, the gun's readiness to shoot is released from the commonsense law of inertia. Then, at any given timepoint after the gun's readiness to shoot is released, the gun may or may not be loaded.

This models a chamber that continues to spin, so that at any given timepoint after the chamber is spun, the gun may or may not be ready to shoot, independently of whether it is ready to shoot at other timepoints. Thus, if the domain of the timepoint sort is the integers and the gun's readiness to shoot is released for n timepoints, then there are 2^n classes of models. (We might instead wish to model a chamber that spins only for an instant. For a way of doing this, see Section 9.1.1.)

We now introduce the *Releases* and *ReleasedAt* predicates. $Releases(e, f, t)$ represents that, if event e occurs at timepoint t, then fluent f will be released from the commonsense law of inertia at $t + 1$. $ReleasedAt(f, t)$ represents that fluent f is released from the commonsense law of inertia at timepoint t. Thus, we have

$$Happens(e, t) \wedge Releases(e, f, t) \Rightarrow ReleasedAt(f, t + 1) \tag{5.24}$$

This is DEC11. We can use *Releases* and *ReleasedAt* to represent our uncertainty about whether a given fluent is true or false at a given timepoint.

We incorporate *ReleasedAt* into (5.14) and (5.11), which gives us the following revised generalized explanation closure axioms:

$$\begin{gathered} HoldsAt(f, t) \wedge \neg ReleasedAt(f, t) \wedge \\ \neg \exists e\, (Happens(e, t) \wedge Terminates(e, f, t)) \Rightarrow \\ HoldsAt(f, t + 1) \end{gathered} \tag{5.25}$$

$$\begin{gathered} \neg HoldsAt(f, t) \wedge \neg ReleasedAt(f, t) \wedge \\ \neg \exists e\, (Happens(e, t) \wedge Initiates(e, f, t)) \Rightarrow \\ \neg HoldsAt(f, t + 1) \end{gathered} \tag{5.26}$$

(5.25) is DEC5 and (5.26) is DEC6.

Thus at any given timepoint a fluent either is, or is not, subject to the commonsense law of inertia. When a fluent is subject to the commonsense law of inertia, its truth value will be the same as its previous value, unless the fluent is true and it is terminated by an occurring event or the fluent is false and it is initiated by an occurring event. When a fluent is released from the commonsense law of inertia, its truth value may or may not be the same as it was previously. That is, its truth value may fluctuate.

5.2.3 Restoring Inertia

We have so far specified how a fluent becomes released from the commonsense law of inertia. We must also specify how a fluent is again made subject to this law.

Although we could add an *Unreleases* predicate, it is convenient to allow *Inititates* and *Terminates* to serve this role because, when a fluent is made subject to the commonsense law of inertia, its truth value is usually also specified.

We have an axiom stating that, if a fluent is initiated or terminated by an event that occurs at timepoint t, then the fluent will no longer be released from the commonsense law of inertia at $t + 1$:

$$Happens(e,t) \wedge (Initiates(e,f,t) \vee Terminates(e,f,t)) \Rightarrow \neg ReleasedAt(f,t+1) \tag{5.27}$$

This is DEC12.

5.2.4 Explanation Closure Axioms for *ReleasedAt*

So far, we have specified when *ReleasedAt* changes truth value, using (5.24) and (5.27). We have not specified when this predicate does not change truth value. We therefore add the following two generalized explanation closure axioms for *ReleasedAt*:

$$ReleasedAt(f,t) \wedge \neg\exists e\,(Happens(e,t) \wedge (Initiates(e,f,t) \vee Terminates(e,f,t))) \Rightarrow ReleasedAt(f,t+1) \tag{5.28}$$

$$\neg ReleasedAt(f,t) \wedge \neg\exists e\,(Happens(e,t) \wedge Releases(e,f,t)) \Rightarrow \neg ReleasedAt(f,t+1) \tag{5.29}$$

(5.28) is DEC7 and (5.29) is DEC8.

5.2.5 Example: Russian Turkey Scenario

This example is due to Erik Sandewall. We modify the example in Section 5.2.1 as follows. If a gun is loaded and an agent spins the gun, then the gun's readiness to shoot will no longer be subject to the commonsense law of inertia:

$$HoldsAt(Loaded(g),t) \Rightarrow Releases(Spin(a,g),Loaded(g),t) \tag{5.30}$$

Instead of waiting at timepoint 1, Nathan spins the gun:

$$Happens(Spin(Nathan,Gun),1) \tag{5.31}$$

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

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

as well as

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

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

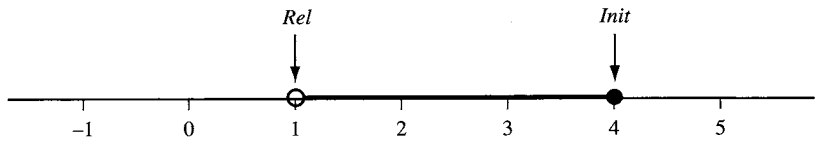


Figure 5.1 Released fluent in EC.

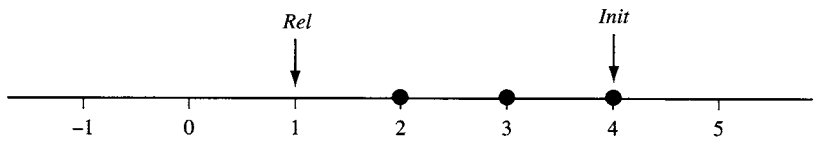


Figure 5.2 Released fluent in DEC.

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

(1997b). The scenario is also discussed by Reiter (2001, pp. 43–46) and Sandewall (1994, pp. 151–152, 197–201). Sun (1995, p. 589) points out that most solutions incorrectly predict that a person who is not shot will be alive forever. This problem could be avoided in a probabilistic treatment (see Section 16.3). The Russian turkey scenario was introduced by Sandewall (1991; 1994, pp. 155–156) and is discussed by Kartha and Lifschitz (1994, p. 344) and Shanahan (1997b, pp. 299-301).

Release from Inertia

Release from the commonsense law of inertia and the *Releases* predicate were introduced into the event calculus by Shanahan (1997b, pp. 299–301), drawing on the occlusion or **X** predicate of Sandewall (1989b, sec. 6; 1994, chap. 11) and the **releases** construct of Kartha and Lifschitz (1994, p. 344). The *ReleasedAt* predicate was incorporated into the event calculus by R. Miller and Shanahan (1999, pp. 17–19; 2002, pp. 471–474). The *ReleasedAt* predicate is similar to the occlusion or **X** predicate of Sandewall. A precursor to *ReleasedAt* is the *Frame* predicate of Lifschitz (1990b, p. 370), which we may relate to *ReleasedAt* via the axiom

$$Frame(f) \Leftrightarrow \forall t \neg ReleasedAt(f, t)$$

Based on McCarthy's (1984a, 1986, 1987) earlier proposals, Lifschitz (1990b, p. 370) proposes the following axiom for the situation calculus:

$$Frame(f) \wedge \neg Ab(f, a, s) \supset Holds(f, Result(a, s)) \equiv Holds(f, s)$$

which is an ancestor of the following axioms for the event calculus (R. Miller and Shanahan, 2002, p. 466):

$$HoldsAt(f, t_2) \leftarrow [HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge Frame(f) \wedge \neg Clipped(t_1, f, t_2)]$$

$$\neg HoldsAt(f, t_2) \leftarrow [\neg HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge Frame(f) \wedge \neg Declipped(t_1, f, t_2)]$$

Exercises

5.1 List three everyday scenarios in which the commonsense law of inertia is not in force.

5.2 Sketch out event calculus representations of the scenarios you listed in Exercise 5.1.

5.3 Prove Proposition 5.6.

5.4 Prove Proposition 5.7.

5.5 (Research Problem) Pursue the proposal of Lifschitz (1990b, p. 372) to have several frames in which the commonsense law of inertia holds.

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:

$$HoldsAt(InRoom(o, r_1), t) \wedge HoldsAt(InRoom(o, r_2), t) \Rightarrow r_1 = r_2 \quad (6.3)$$

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:

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

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:

$$HoldsAt(Holding(a, o), t) \Rightarrow Terminates(LetGoOf(a, o), Holding(a, o), t) \quad (6.5)$$

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:

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

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

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

$$\neg ReleasedAt(f, t) \quad (6.8)$$

$$HoldsAt(InRoom(Nathan, LivingRoom), 0) \quad (6.9)$$

$$HoldsAt(InRoom(Book, LivingRoom), 0) \quad (6.10)$$

Nathan picks up the book and walks into the kitchen:

$$Happens(PickUp(Nathan, Book), 0) \quad (6.11)$$

$$Happens(Walk(Nathan, LivingRoom, Kitchen), 1) \quad (6.12)$$

We also have

$$LivingRoom \neq Kitchen \quad (6.13)$$

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

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

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

$$Initiates(e,f,t) \Leftrightarrow \tag{6.14}$$
$$\exists a, r_1, r_2\, (e = Walk(a, r_1, r_2) \wedge f = InRoom(a, r_2)) \vee$$
$$\exists a, o, r\, (e = PickUp(a, o) \wedge$$
$$f = Holding(a, o) \wedge$$
$$HoldsAt(InRoom(a, r), t) \wedge$$
$$HoldsAt(InRoom(o, r), t)) \vee$$
$$\exists a, o, r_1, r_2\, (e = Walk(a, r_1, r_2) \wedge$$
$$f = InRoom(o, r_2) \wedge$$
$$HoldsAt(Holding(a, o), t))$$

$$Terminates(e,f,t) \Leftrightarrow \tag{6.15}$$
$$\exists a, r_1, r_2\, (e = Walk(a, r_1, r_2) \wedge$$
$$f = InRoom(a, r_1) \wedge$$
$$r_1 \neq r_2) \vee$$
$$\exists a, o\, (e = LetGoOf(a, o) \wedge$$
$$f = Holding(a, o) \wedge$$
$$HoldsAt(Holding(a, o), t)) \vee$$
$$\exists a, o, r_1, r_2\, (e = Walk(a, r_1, r_2) \wedge$$
$$f = InRoom(o, r_1) \wedge$$
$$HoldsAt(Holding(a, o), t) \wedge$$
$$r_1 \neq r_2)$$

$$\neg Releases(e,f,t) \tag{6.16}$$

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

$$Happens(e,t) \Leftrightarrow \tag{6.17}$$
$$(e = PickUp(Nathan, Book) \wedge t = 0) \vee$$
$$(e = Walk(Nathan, LivingRoom, Kitchen) \wedge t = 1)$$

From (6.17) and EC3, we have

$$\neg StoppedIn(0, Holding(Nathan, Book), 1) \tag{6.18}$$

From (6.16) and EC13, we have

$$\neg ReleasedIn(0, Holding(Nathan, Book), 1) \tag{6.19}$$

From (6.11) (which follows from (6.17)), (6.9), (6.10), (6.4) (which follows from (6.14)), $0 < 1$, (6.18), (6.19), and EC14, we have

$$HoldsAt(Holding(Nathan, Book), 1) \tag{6.20}$$

From (6.17) and EC3, we have

$$\neg StoppedIn(1, InRoom(Book, Kitchen), 2) \tag{6.21}$$

From (6.16) and EC13, we have

$$\neg ReleasedIn(1, InRoom(Book, Kitchen), 2) \tag{6.22}$$

From (6.12) (which follows from (6.17)), (6.20), (6.6) (which follows from (6.14)), $1 < 2$, (6.21), (6.22), and EC14, we have $HoldsAt(InRoom(Book, Kitchen), 2)$, as required. ■

6.1.2 Discussion

The advantage of the method of using effect axioms to represent indirect effects is its simplicity. The disadvantage of this method is that more axioms may be required than with other methods. For example, suppose we add *Run* as another way of getting from one room to another:

$$Initiates(Run(a, r_1, r_2), InRoom(a, r_2), t)$$
$$r_1 \neq r_2 \Rightarrow Terminates(Run(a, r_1, r_2), InRoom(a, r_1), t)$$

Previously we represented the indirect effects of walking while holding an object. Now we must represent the indirect effects of running while holding an object as well:

$$HoldsAt(Holding(a, o), t) \Rightarrow$$
$$Initiates(Run(a, r_1, r_2), InRoom(o, r_2), t)$$

$$HoldsAt(Holding(a, o), t) \Rightarrow$$
$$Terminates(Run(a, r_1, r_2), InRoom(o, r_1), t)$$

The indirect effects of changing rooms do not come for free.

6.2 Primitive and Derived Fluents

Another way of dealing with the ramification problem is to divide fluents into two categories: *primitive fluents*, or fluents directly affected by events, and *derived fluents*, or fluents indirectly affected by events. The truth value of a primitive fluent is determined by effect axioms, whereas the truth value of a derived fluent is determined by state constraints. The derived fluent is released from the commonsense law of inertia at all times, so its truth value is permitted to vary and is determined by state constraints.

6.2.1 Example: Device

We start with some positive and negative effect axioms. If an agent switches on a device, then the device will be switched on:

$$Initiates(SwitchOn(a, d), SwitchedOn(d), t) \tag{6.23}$$

If an agent switches off a device, then the device will no longer be switched on:

$$Terminates(SwitchOff(a, d), SwitchedOn(d), t) \tag{6.24}$$

If an agent plugs in a device, then the device will be plugged in:

$$Initiates(PlugIn(a, d), PluggedIn(d), t) \tag{6.25}$$

If an agent unplugs a device, then the device will no longer be plugged in:

$$Terminates(Unplug(a, d), PluggedIn(d), t) \tag{6.26}$$

We use a state constraint to represent the fact that a device is on whenever the device is switched on and the device is plugged in:

$$HoldsAt(On(d), t) \Leftrightarrow \tag{6.27}$$
$$HoldsAt(SwitchedOn(d), t) \wedge HoldsAt(PluggedIn(d), t)$$

The order in which the device is switched on and plugged in does not matter. The device will be on once it is both switched on and plugged in.

Unlike the use of effect axioms for representing indirect effects, when we add other events that initiate or terminate *SwitchedOn*, we do not have to add any further axioms. The state constraint (6.27) represents an indirect effect of *SwitchedOn*, no matter how *SwitchedOn* is initiated or terminated. The same is true for *Plugged In*.

Consider the following observations and narrative. Initially, the device is neither switched on nor plugged in:

$$ReleasedAt(On(d), t) \tag{6.28}$$
$$\neg ReleasedAt(SwitchedOn(d), t) \tag{6.29}$$
$$\neg ReleasedAt(PluggedIn(d), t) \tag{6.30}$$
$$\neg HoldsAt(SwitchedOn(Device1), 0) \tag{6.31}$$
$$\neg HoldsAt(PluggedIn(Device1), 0) \tag{6.32}$$

Then Nathan switches on and plugs in the device:

$$Happens(SwitchOn(Nathan, Device1), 0) \tag{6.33}$$
$$Happens(PlugIn(Nathan, Device1), 1) \tag{6.34}$$

We can show that the device will then be on.

PROPOSITION 6.2 Let $\Sigma = (6.23) \wedge (6.24) \wedge (6.25) \wedge (6.26)$, $\Delta = (6.33) \wedge (6.34)$, $\Omega = U[SwitchOn, PlugIn] \wedge U[SwitchedOn, PluggedIn, On]$, $\Psi = (6.27)$, and $\Gamma = (6.28) \wedge (6.29) \wedge (6.30) \wedge (6.31) \wedge (6.32)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Psi \wedge \Gamma \wedge EC \models HoldsAt(On(Device1), 2)$$

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

$$Initiates(e, f, t) \Leftrightarrow \tag{6.35}$$
$$\exists a, d\,(e = SwitchOn(a, d) \wedge f = SwitchedOn(d)) \vee$$
$$\exists a, d\,(e = PlugIn(a, d) \wedge f = PluggedIn(d))$$

$$Terminates(e, f, t) \Leftrightarrow \tag{6.36}$$
$$\exists a, d\,(e = SwitchOff(a, d) \wedge f = SwitchedOn(d)) \vee$$
$$\exists a, d\,(e = Unplug(a, d) \wedge f = PluggedIn(d))$$

$$\neg Releases(e, f, t) \tag{6.37}$$

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

$$Happens(e, t) \Leftrightarrow \tag{6.38}$$
$$(e = SwitchOn(Nathan, Device1) \wedge t = 0) \vee$$
$$(e = PlugIn(Nathan, Device1) \wedge t = 1)$$

From (6.38), (6.36), and EC3, we have

$$\neg StoppedIn(0, SwitchedOn(Device1), 2) \tag{6.39}$$

From (6.37) and EC13, we have

$$\neg ReleasedIn(0, SwitchedOn(Device1), 2) \tag{6.40}$$

From (6.33) (which follows from (6.38)), (6.23) (which follows from (6.35)), $0 < 2$, (6.39), (6.40), and EC14, we have

$$HoldsAt(SwitchedOn(Device1), 2) \tag{6.41}$$

From (6.38) and EC3, we have

$$\neg StoppedIn(1, PluggedIn(Device1), 2) \tag{6.42}$$

From (6.37) and EC13, we have

$$\neg ReleasedIn(1, PluggedIn(Device1), 2) \tag{6.43}$$

From (6.34) (which follows from (6.38)), (6.25) (which follows from (6.35)), $1 < 2$, (6.42), (6.43), and EC14, we have

$$HoldsAt(PluggedIn(Device1), 2) \tag{6.44}$$

From (6.41), (6.44), and (6.27), we have $HoldsAt(On(Device1), 2)$. ■

6.3 Release Axioms and State Constraints

Another method of dealing with the ramification problem involves temporarily releasing a fluent from the commonsense law of inertia. First, we release the fluent, and we make the fluent subject to a state constraint while it is released. Later, we make the fluent subject to the commonsense law of inertia again.

6.3.1 Example: Carrying a Book Revisited

An agent picks up a book. While the agent is holding the book, its location is not subject to the commonsense law of inertia. Instead, its location varies with the location of the agent holding it.

We modify the example in Section 6.1.1 to use release axioms and state constraints instead of effect axioms. If an agent picks up an object, then the room of the object will be released from the commonsense law of inertia:

$$Releases(PickUp(a, o), InRoom(o, r), t) \tag{6.45}$$

Whenever an agent is holding an object, if the agent is in one room, then the object is also in that room:

$$HoldsAt(Holding(a, o), t) \wedge HoldsAt(InRoom(a, r), t) \Rightarrow HoldsAt(InRoom(o, r), t) \tag{6.46}$$

If an agent is in a room and the agent lets go of an object, then the object will be in the room, and the fact that the object is in that room will no longer be released from the commonsense law of inertia:

$$HoldsAt(InRoom(a, r), t) \Rightarrow Initiates(LetGoOf(a, o), InRoom(o, r), t) \tag{6.47}$$

Unlike with the use of effect axioms for representing indirect effects, if we add running as another way of moving from room to room, we do not have to add any further axioms. The state constraint (6.46) represents an indirect effect of an agent's location, no matter how the agent's location is changed.

Suppose we have the following observations and narrative. At first, Nathan and the book are in the living room:

$$\neg ReleasedAt(f, t) \tag{6.48}$$

$$HoldsAt(InRoom(Nathan, LivingRoom), 0) \tag{6.49}$$

$$HoldsAt(InRoom(Book, LivingRoom), 0) \tag{6.50}$$

Then Nathan picks up the book and walks into the kitchen:

$$Happens(PickUp(Nathan, Book), 0) \tag{6.51}$$

$$Happens(Walk(Nathan, LivingRoom, Kitchen), 1) \tag{6.52}$$

We also have the fact that the living room and the kitchen are not the same room:

$$LivingRoom \neq Kitchen \tag{6.53}$$

We can show that the book will end up in the kitchen.

PROPOSITION 6.3

If $\Sigma = (6.1) \wedge (6.2) \wedge (6.4) \wedge (6.5) \wedge (6.45) \wedge (6.47)$, $\Delta = (6.51) \wedge (6.52)$, $\Omega = U[Walk, PickUp, LetGoOf] \wedge U[InRoom, Holding] \wedge (6.53)$, $\Psi = (6.3) \wedge (6.46)$, and $\Gamma = (6.48) \wedge (6.49) \wedge (6.50)$, then

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

Proof From CIRC[Σ; *Initiates*, *Terminates*, *Releases*], Theorem 2.2, and Theorem 2.1, we have

$$\begin{aligned}
&Initiates(e,f,t) \Leftrightarrow \\
&\exists a, r_1, r_2\,(e = Walk(a, r_1, r_2) \wedge f = InRoom(a, r_2)) \vee \\
&\exists a, o, r\,(e = PickUp(a, o) \wedge \\
&f = Holding(a, o) \wedge \\
&HoldsAt(InRoom(a, r), t) \wedge \\
&HoldsAt(InRoom(o, r), t)) \vee \\
&\exists a, o, r\,(e = LetGoOf(a, o) \wedge \\
&f = InRoom(o, r) \wedge \\
&HoldsAt(InRoom(a, r), t))
\end{aligned} \tag{6.54}$$

$$\begin{aligned}
&Terminates(e,f,t) \Leftrightarrow \\
&\exists a, r_1, r_2\,(e = Walk(a, r_1, r_2) \wedge \\
&f = InRoom(a, r_1) \wedge \\
&r_1 \neq r_2) \vee \\
&\exists a, o\,(e = LetGoOf(a, o) \wedge \\
&f = Holding(a, o) \wedge \\
&HoldsAt(Holding(a, o), t))
\end{aligned} \tag{6.55}$$

$$\begin{aligned}
&Releases(e,f,t) \Leftrightarrow \\
&\exists a, o, r\,(e = PickUp(a, o) \wedge f = InRoom(o, r))
\end{aligned} \tag{6.56}$$

From CIRC[Δ; *Happens*] and Theorem 2.1, we have

$$\begin{aligned}
&Happens(e,t) \Leftrightarrow \\
&(e = PickUp(Nathan, Book) \wedge t = 0) \vee \\
&(e = Walk(Nathan, LivingRoom, Kitchen) \wedge t = 1)
\end{aligned} \tag{6.57}$$

From (6.57), (6.55), and EC3, we have

$$\neg StoppedIn(0, Holding(Nathan, Book), 2) \tag{6.58}$$

From (6.56) and EC13, we have

$$\neg ReleasedIn(0, Holding(Nathan, Book), 2) \tag{6.59}$$

From (6.51) (which follows from (6.57)), (6.49), (6.50), (6.4) (which follows from (6.54)), $0 < 2$, (6.58), (6.59), and EC14, we have

$$HoldsAt(Holding(Nathan, Book), 2) \tag{6.60}$$

From (6.57) and EC3, we have

$$\neg StoppedIn(1, InRoom(Nathan, Kitchen), 2) \tag{6.61}$$

From (6.57) and EC13, we have

$$\neg Released(1, InRoom(Nathan, Kitchen), 2) \tag{6.62}$$

From (6.52) (which follows from (6.57)), (6.1) (which follows from (6.54)), $1 < 2$, (6.61), (6.62), and EC14, we have $HoldsAt(InRoom(Nathan, Kitchen), 2)$. From this, (6.60), and (6.46), we have $HoldsAt(InRoom(Book, Kitchen), 2)$. ■

6.4 Effect Constraints

Another way of representing indirect effects is to use universally quantified formulas involving *Initiates* and *Terminates*.

DEFINITION 6.1 If γ is a condition, π_1 and π_2 are *Initiates* or *Terminates*, α is an event variable, β_1 and β_2 are fluent terms, and τ is a timepoint term, then

$$(\gamma \wedge \pi_1(\alpha, \beta_1, \tau)) \Rightarrow \pi_2(\alpha, \beta_2, \tau)$$

is an ***effect constraint***.

6.4.1 Example: Carrying a Book Revisited

Consider again the example in Section 6.1.1. We may represent the indirect effects of walking from one room to another using effect constraints:

$$\begin{aligned} &HoldsAt(Holding(a, o), t) \wedge \\ &Initiates(e, InRoom(a, r), t) \Rightarrow \\ &Initiates(e, InRoom(o, r), t) \end{aligned} \tag{6.63}$$

$$\begin{aligned} &HoldsAt(Holding(a, o), t) \wedge \\ &Terminates(e, InRoom(a, r), t) \Rightarrow \\ &Terminates(e, InRoom(o, r), t) \end{aligned} \tag{6.64}$$

Unfortunately, we cannot use the methods given in Section 2.6.1 to compute $CIRC[(6.1) \wedge (6.2) \wedge (6.4) \wedge (6.5) \wedge (6.63) \wedge (6.64); Initiates, Terminates, Releases]$. Instead we simply use (6.63) and (6.64) to derive the effect axioms in Section 6.1.1 that represent the indirect effects. From (6.63) and

$$Initiates(Walk(a, r_1, r_2), InRoom(a, r_2), t)$$

we have

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

which is (6.6). From (6.64) and

$$r_1 \neq r_2 \Rightarrow Terminates(Walk(a, r_1, r_2), InRoom(a, r_1), t)$$

we have

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

which is (6.7).

6.5 Causal Constraints

Michael Thielscher devised the electronic circuit shown in Figure 6.1 in order to point out problems with solutions to the ramification problem based on dividing fluents into primitive and derived fluents, as discussed in Section 6.2. The circuit consists of a battery (B), three switches (S1, S2, and S3), a relay (R), and a light (L). A switch can be open or closed, the relay can be activated or not activated, and

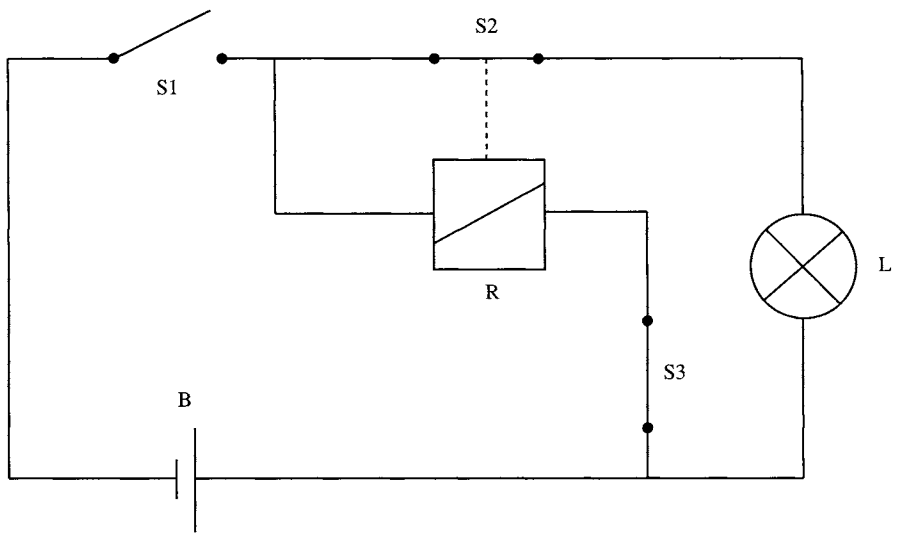


Figure 6.1 Thielscher's circuit (Thielscher, 1996, p. 19).

the light can be lit or not lit. Relay R is activated whenever S1 and S3 are closed. Light L is lit whenever S1 and S2 are closed. Relay R is connected to switch S2, so that when R is activated, S2 is opened.

One behavior of this circuit is as follows. Suppose that initially S1 is open, S2 and S3 are closed, R is not activated, and L is not lit. If we close S1, then R will be activated and S2 will be opened. Because S2 is open, the result will be that light L is not lit.

Now suppose we try to model this behavior using primitive and derived fluents. We use primitive fluents for S1 and S3:

$$Initiates(Close(S1), Closed(S1), t) \tag{6.65}$$
$$Terminates(Open(S1), Closed(S1), t) \tag{6.66}$$
$$Initiates(Close(S3), Closed(S3), t) \tag{6.67}$$
$$Terminates(Open(S3), Closed(S3), t) \tag{6.68}$$

We use derived fluents for R and L:

$$HoldsAt(Activated(R), t) \Leftrightarrow HoldsAt(Closed(S1), t) \wedge HoldsAt(Closed(S3), t) \tag{6.69}$$

$$HoldsAt(Lit(L), t) \Leftrightarrow HoldsAt(Closed(S1), t) \wedge HoldsAt(Closed(S2), t) \tag{6.70}$$

It is not clear what to use for S2, which appears to be both primitive and derived. On the one hand, S2 can be manually opened and closed:

$$Initiates(Close(S2), Closed(S2), t) \tag{6.71}$$
$$Terminates(Open(S2), Closed(S2), t) \tag{6.72}$$

On the other hand, S2 is open whenever R is activated:

$$\neg HoldsAt(Closed(S2), t) \Leftrightarrow HoldsAt(Activated(R), t) \tag{6.73}$$

Unfortunately if we use all three axioms (6.71), (6.72), and (6.73), then we get inconsistency. To see this, consider the following observations and narrative:

$$\neg HoldsAt(Closed(S1), 0) \tag{6.74}$$
$$HoldsAt(Closed(S2), 0) \tag{6.75}$$
$$HoldsAt(Closed(S3), 0) \tag{6.76}$$
$$\neg HoldsAt(Activated(R), 0) \tag{6.77}$$
$$\neg HoldsAt(Lit(L), 0) \tag{6.78}$$
$$\neg ReleasedAt(f, t) \tag{6.79}$$
$$Happens(Close(S1), 0) \tag{6.80}$$

PROPOSITION 6.4 Let $\Sigma = (6.65) \wedge (6.66) \wedge (6.67) \wedge (6.68) \wedge (6.71) \wedge (6.72)$, $\Delta = (6.80)$, $\Omega = U[Close, Open, Activate, Light] \wedge U[Closed, Activated, Lit]$, and $\Psi = (6.69) \wedge (6.70) \wedge (6.73)$, and $\Gamma = (6.74) \wedge (6.75) \wedge (6.76) \wedge (6.77) \wedge (6.78) \wedge (6.79)$. Then $CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Psi \wedge \Gamma \wedge$ DEC is inconsistent.

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

$$\begin{aligned} &Initiates(e,f,t) \Leftrightarrow \\ &(e = Close(S1) \wedge f = Closed(S1)) \vee \\ &(e = Close(S2) \wedge f = Closed(S2)) \vee \\ &(e = Close(S3) \wedge f = Closed(S3)) \end{aligned} \tag{6.81}$$

$$\begin{aligned} &Terminates(e,f,t) \Leftrightarrow \\ &(e = Open(S1) \wedge f = Closed(S1)) \vee \\ &(e = Open(S2) \wedge f = Closed(S2)) \vee \\ &(e = Open(S3) \wedge f = Closed(S3)) \end{aligned} \tag{6.82}$$

$$\neg Releases(e,f,t). \tag{6.83}$$

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

$$\begin{aligned} &Happens(e,t) \Leftrightarrow \\ &(e = Close(S1) \wedge t = 0) \end{aligned} \tag{6.84}$$

From (6.80) (which follows from (6.84)), (6.65) (which follows from (6.81)), and DEC9, we have $HoldsAt(Closed(S1), 1)$. From this, (6.76), and (6.69), we have $HoldsAt(Activated(R), 1)$. From this and (6.73), we have

$$\neg HoldsAt(Closed(S2), 1) \tag{6.85}$$

Yet, from (6.84) and (6.82), we have $\neg\exists e\,(Happens(e, 0) \wedge Terminates(e, Closed(S2), 0))$. From this, (6.75), (6.79), and DEC5, we have $HoldsAt(Closed(S2), 1)$, which contradicts (6.85). ■

Thus, it does not work for S2 to be both a primitive and derived fluent.

In order to deal with this circuit, or any commonsense reasoning problem in which indirect effects interact with one another instantaneously, Murray Shanahan enhances the event calculus as follows. First, he adds the following four predicates.

$Started(f, t)$: Fluent f is true at timepoint t, or f is initiated by an event that occurs at t. We say that f is *started* at t.

Stopped(f,t): Fluent f is false at timepoint t, or f is terminated by an event that occurs at t. We say that f is *stopped* at t.

Initiated(f,t): Fluent f is started at timepoint t, and f is not terminated by any event that occurs at t.

Terminated(f,t): Fluent f is stopped at timepoint t, and f is not initiated by any event that occurs at t.

Second, he adds the following axioms.

AXIOM CC1
$$Started(f,t) \Leftrightarrow (HoldsAt(f,t) \vee \exists e\,(Happens(e,t) \wedge Initiates(e,f,t)))$$

AXIOM CC2
$$Stopped(f,t) \Leftrightarrow (\neg HoldsAt(f,t) \vee \exists e\,(Happens(e,t) \wedge Terminates(e,f,t)))$$

AXIOM CC3
$$Initiated(f,t) \Leftrightarrow (Started(f,t) \wedge \neg\exists e\,(Happens(e,t) \wedge Terminates(e,f,t)))$$

AXIOM CC4
$$Terminated(f,t) \Leftrightarrow (Stopped(f,t) \wedge \neg\exists e\,(Happens(e,t) \wedge Initiates(e,f,t)))$$

We use CC to mean the conjunction of axioms CC1 through CC4. Third, Shanahan introduces a new type of axiom called a causal constraint.

DEFINITION 6.2
If β and $\beta_1,\ldots,\beta_n$ are fluent terms, σ is *Stopped* or *Started*, $\pi_1,\ldots,\pi_n$ are *Initiated* or *Terminated*, and τ is a timepoint term, then

$$\sigma(\beta,\tau) \wedge \pi_1(\beta_1,\tau) \wedge \cdots \wedge \pi_n(\beta_n,\tau) \Rightarrow Happens(\beta,\tau)$$

is a ***causal constraint***.

6.5.1 Example: Thielscher's Circuit

We now show how causal constraints can be used to deal properly with Thielscher's circuit. Our axiomatization of the circuit is the same as Shanahan's, except that we use events such as *Close*(*S1*) instead of *Close1*, use fluents such as *Closed*(*S1*) instead of *Switch1*, and have axioms for opening and closing all switches.

We use the observations and narrative just discussed, and we have the following effect axioms:

$$Initiates(Close(s), Closed(s), t) \tag{6.86}$$
$$Terminates(Open(s), Closed(s), t) \tag{6.87}$$
$$Initiates(Activate(r), Activated(r), t) \tag{6.88}$$
$$Initiates(Light(l), Lit(l), t) \tag{6.89}$$

We have the following causal constraints:

$$\begin{aligned} &Stopped(Lit(L), t) \wedge \\ &Initiated(Closed(S1), t) \wedge \\ &Initiated(Closed(S2), t) \Rightarrow \\ &Happens(Light(L), t) \end{aligned} \tag{6.90}$$

$$\begin{aligned} &Started(Closed(S2), t) \wedge \\ &Initiated(Activated(R), t) \Rightarrow \\ &Happens(Open(S2), t) \end{aligned} \tag{6.91}$$

$$\begin{aligned} &Stopped(Activated(R), t) \wedge \\ &Initiated(Closed(S1), t) \wedge \\ &Initiated(Closed(S3), t) \Rightarrow \\ &Happens(Activate(R), t) \end{aligned} \tag{6.92}$$

Given this axiomatization we can show that after switch S1 is closed, light L will not be lit.

PROPOSITION 6.5 Let $\Sigma = (6.86) \wedge (6.87) \wedge (6.88) \wedge (6.89)$, $\Delta = (6.80) \wedge (6.90) \wedge (6.91) \wedge (6.92)$, $\Omega = U[Close, Open, Activate, Light] \wedge U[Closed, Activated, Lit]$, and $\Gamma = (6.74) \wedge (6.75) \wedge (6.76) \wedge (6.77) \wedge (6.78) \wedge (6.79)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Gamma \wedge \mathrm{DEC} \wedge \mathrm{CC} \models \neg HoldsAt(Lit(L), 1)$$

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

$$\begin{aligned} &Initiates(e, f, t) \Leftrightarrow \\ &\exists s\,(e = Close(s) \wedge f = Closed(s)) \vee \\ &\exists r\,(e = Activate(r) \wedge f = Activated(r)) \vee \\ &\exists l\,(e = Light(l) \wedge f = Lit(l)) \end{aligned} \tag{6.93}$$

$$\begin{aligned} &Terminates(e, f, t) \Leftrightarrow \\ &\exists s\,(e = Open(s) \wedge f = Closed(s)) \end{aligned} \tag{6.94}$$

$$\neg Releases(e, f, t) \tag{6.95}$$

From CIRC[Δ; *Happens*] and Theorem 2.1, we have

$$\begin{aligned}
&Happens(e,t) \Leftrightarrow \\
&(e = Close(S1) \wedge t = 0) \vee \\
&\\
&(e = Light(L) \wedge \\
&Stopped(Lit(L),t) \wedge \\
&Initiated(Closed(S1),t) \wedge \\
&Initiated(Closed(S2),t)) \vee \\
&(e = Open(S2) \wedge \\
&Started(Closed(S2),t) \wedge \\
&Initiated(Activated(R),t)) \vee \\
&(e = Activate(R) \wedge \\
&Stopped(Activated(R),t) \wedge \\
&Initiated(Closed(S1),t) \wedge \\
&Initiated(Closed(S3),t))
\end{aligned} \tag{6.96}$$

From (6.77) and CC2, we have

$$Stopped(Activated(R), 0) \tag{6.97}$$

From (6.80) (which follows from (6.96)), and (6.86) (which follows from (6.93)), we have $\exists e\,(Happens(e,0) \wedge Initiates(e, Closed(S1), 0))$. From this and CC1, we have

$$Started(Closed(S1), 0) \tag{6.98}$$

From (6.94) and (6.96), we have $\neg\exists e\,(Happens(e,0) \wedge Terminates(e, Closed(S1), 0))$. From this, (6.98), and CC3, we have

$$Initiated(Closed(S1), 0) \tag{6.99}$$

From (6.76) and CC1, we have

$$Started(Closed(S3), 0) \tag{6.100}$$

From (6.94) and (6.96), we have $\neg\exists e\,(Happens(e,0) \wedge Terminates(e, Closed(S3), 0))$. From this, (6.100), and CC3, we have $Initiated(Closed(S3), 0)$. From this, (6.97), (6.99), and (6.92) (which follows from (6.96)), we have $Happens(Activate(R), 0)$. From this and (6.88) (which follows from (6.93)), we have $\exists e\,(Happens(e,0) \wedge Initiates(e, Activated(R), 0))$. From this and CC1, we have

$$Started(Activated(R), 0) \tag{6.101}$$

From (6.94), we have $\neg\exists e\,(Happens(e,0) \wedge Terminates(e, Activated(R), 0))$. From this, (6.101), and CC3, we have

$$Initiated(Activated(R), 0) \tag{6.102}$$

From (6.75) and CC1, we have $Started(Closed(S2), 0)$. From this, (6.102), and (6.91) (which follows from (6.96)), we have $Happens(Open(S2), 0)$. From this and (6.87) (which follows from (6.94)), we have $\exists e\,(Happens(e,0) \wedge Terminates(e, Closed(S2), 0))$. From this and CC3, we have $\neg Initiated(Closed(S2), 0)$. From this and (6.96), we have $\neg Happens(Light(L), 0)$. From this and (6.93), we have $\neg\exists e\,(Happens(e,0) \wedge Initiates(e, Lit(L), 0))$. From this, (6.78), (6.79), and DEC6, we have $\neg HoldsAt(Lit(L), 1)$. ■

6.6 Trigger Axioms

There is a further option for representing indirect effects. If the indirect effects are delayed and not instantaneous, then we can represent them using the trigger axioms described in Section 4.1. We can in fact use trigger axioms to represent the behavior of Thielscher's circuit (Figure 6.1). In this case, with the conjunction of axioms DEC the light will light up for two timepoints before going out again; with the conjunction of axioms EC the light will light up for an instant and then go out.

6.6.1 Example: Thielscher's Circuit with Delays

Instead of causal constraints, we use the following trigger axioms:

$$\begin{gathered}\neg HoldsAt(Lit(L), t) \wedge \\ HoldsAt(Closed(S1), t) \wedge \\ HoldsAt(Closed(S2), t) \Rightarrow \\ Happens(Light(L), t)\end{gathered} \tag{6.103}$$

$$\begin{gathered}HoldsAt(Lit(L), t) \wedge \\ (\neg HoldsAt(Closed(S1), t) \vee \neg HoldsAt(Closed(S2), t)) \Rightarrow \\ Happens(Unlight(L), t)\end{gathered} \tag{6.104}$$

$$\begin{gathered}HoldsAt(Closed(S2), t) \wedge \\ HoldsAt(Activated(R), t) \Rightarrow \\ Happens(Open(S2), t)\end{gathered} \tag{6.105}$$

$$\begin{aligned}&\neg HoldsAt(Activated(R), t) \wedge \\ &HoldsAt(Closed(S1), t) \wedge \\ &HoldsAt(Closed(S3), t) \Rightarrow \\ &Happens(Activate(R), t)\end{aligned} \tag{6.106}$$

We add the following effect axiom to those already given:

$$Terminates(Unlight(l), Lit(l), t) \tag{6.107}$$

Given the observations and narrative, we can show that the light is lit at timepoint 2 and is not lit at timepoint 4.

PROPOSITION 6.6

Let $\Sigma = (6.86) \wedge (6.87) \wedge (6.88) \wedge (6.89) \wedge (6.107)$, $\Delta = (6.80) \wedge (6.103) \wedge (6.104) \wedge (6.105) \wedge (6.106)$, $\Omega = U[Close, Open, Activate, Light] \wedge U[Closed, Activated, Lit]$, and $\Gamma = (6.74) \wedge (6.75) \wedge (6.76) \wedge (6.77) \wedge (6.78) \wedge (6.79)$. Then we have

$$\begin{aligned}&CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ &\quad \Omega \wedge \Gamma \wedge \text{DEC} \models HoldsAt(Lit(L), 2) \wedge \neg HoldsAt(Lit(L), 4)\end{aligned}$$

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

$$\begin{aligned}&Initiates(e, f, t) \Leftrightarrow \\ &\exists s\,(e = Close(s) \wedge f = Closed(s)) \vee \\ &\exists r\,(e = Activate(r) \wedge f = Activated(r)) \vee \\ &\exists l\,(e = Light(l) \wedge f = Lit(l))\end{aligned} \tag{6.108}$$

$$\begin{aligned}&Terminates(e, f, t) \Leftrightarrow \\ &\exists s\,(e = Open(s) \wedge f = Closed(s)) \vee \\ &\exists l\,(e = Unlight(l) \wedge Lit(l))\end{aligned} \tag{6.109}$$

$$\neg Releases(e, f, t) \tag{6.110}$$

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

$$\begin{aligned}&Happens(e, t) \Leftrightarrow \\ &(e = Close(S1) \wedge t = 0) \vee\end{aligned} \tag{6.111}$$

$$
\begin{aligned}
&(e = Light(L) \wedge \\
&\neg HoldsAt(Lit(L), t) \wedge \\
&HoldsAt(Closed(S1), t) \wedge \\
&HoldsAt(Closed(S2), t)) \vee \\
&(e = Unlight(L) \wedge \\
&HoldsAt(Lit(L), t) \wedge \\
&(\neg HoldsAt(Closed(S1), t) \vee \neg HoldsAt(Closed(S2), t))) \vee \\
&(e = Open(S2) \wedge \\
&HoldsAt(Closed(S2), t) \wedge \\
&HoldsAt(Activated(R), t)) \vee \\
&(e = Activate(R) \wedge \\
&\neg HoldsAt(Activated(R), t) \wedge \\
&HoldsAt(Closed(S1), t) \wedge \\
&HoldsAt(Closed(S3), t))
\end{aligned}
$$

First we show that the light goes on. From (6.80) (which follows from (6.111)), (6.86) (which follows from (6.108)), and DEC9 we have

$$HoldsAt(Closed(S1), 1) \tag{6.112}$$

From (6.77) and (6.111), we have $\neg Happens(Open(S2), 0)$. From this and (6.109), we have $\neg\exists e\,(Happens(e, 0) \wedge Terminates(e, Closed(S2), 0))$. From this, (6.75), (6.79), and DEC5, we have

$$HoldsAt(Closed(S2), 1) \tag{6.113}$$

From (6.74), (6.111), and (6.108), we have $\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, Lit(L), 0))$. From this, (6.78), (6.79), and DEC6, we have $\neg HoldsAt(Lit(L), 1)$. From this, (6.112), (6.113), and (6.103) (which follows from (6.111)), we have $Happens(Light(L), 1)$. From this, (6.89) (which follows from (6.108)), and DEC9, we have

$$HoldsAt(Lit(L), 2) \tag{6.114}$$

Second we show that the light goes off again. From (6.74) and (6.111), we have $\neg Happens(Activate(R), 0)$. From this and (6.108), we have $\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, Activated(R), 0))$. From this, (6.77), (6.79), and DEC6, we have

$$\neg HoldsAt(Activated(R), 1) \tag{6.115}$$

From (6.111) and (6.109), we have $\neg\exists e\,(Happens(e,0) \wedge Terminates(e, Closed(S3),0))$. From this, (6.76), (6.79), and DEC5, we have $HoldsAt(Closed(S3),1)$. From this, (6.115), (6.112), and (6.106) (which follows from (6.111)), we have $Happens(Activate(R),1)$. From this, (6.88) (which follows from (6.108)), and DEC9, we have

$$HoldsAt(Activated(R),2) \tag{6.116}$$

From (6.115) and (6.111), we have $\neg Happens(Open(S2),1)$. From this and (6.109), we have $\neg\exists e\,(Happens(e,1) \wedge Terminates(e, Closed(S2),1))$. From this, (6.113), (6.79), and DEC5, we have

$$HoldsAt(Closed(S2),2) \tag{6.117}$$

From this, (6.116), and (6.105) (which follows from (6.111)), we have $Happens(Open(S2),2)$. From this, (6.87) (which follows from (6.109)), and DEC10, we have

$$\neg HoldsAt(Closed(S2),3) \tag{6.118}$$

From (6.111) and (6.109), we have $\neg\exists e\,(Happens(e,1) \wedge Terminates(e, Closed(S1),1))$. From this, (6.112), (6.79), and DEC5, we have $HoldsAt(Closed(S1),2)$. From this, (6.117), and (6.111), we have $\neg Happens(Unlight(L),2)$. From this and (6.109), we have $\neg\exists e\,(Happens(e,2) \wedge Terminates(e, Lit(L),2))$. From this, (6.114), (6.79), and DEC5, we have $HoldsAt(Lit(L),3)$. From this, (6.118), and (6.104) (which follows from (6.111)), we have $Happens(Unlight(L),3)$. From this, (6.107) (which follows from (6.109)), and DEC10, we have $\neg HoldsAt(Lit(L),4)$. ■

6.6.2 Example: Shanahan's Circuit with Delays

Shanahan showed that if the circuit is modified to that shown in Figure 6.2, then an axiomatization of the circuit using causal constraints as in section 6.5 produces inconsistency. We can properly represent the behavior of this circuit using trigger axioms. The circuit implements a buzzer that is turned on and off with S1; we ignore the light in this discussion. A moment after S1 is closed, relay R is activated. A moment after R is activated, S2 is open. A moment after S2 is open, R is not activated. A moment after relay R is not activated, S2 is closed. A moment after S2 is closed, R is activated again. The cycle repeats. In a real circuit, the result of a relay quickly and repeatedly activating and deactivating is a buzzing sound.

We have the following trigger axioms:

$$\begin{aligned} &\neg HoldsAt(Lit(L),t) \wedge \\ &HoldsAt(Closed(S1),t) \wedge \end{aligned} \tag{6.119}$$

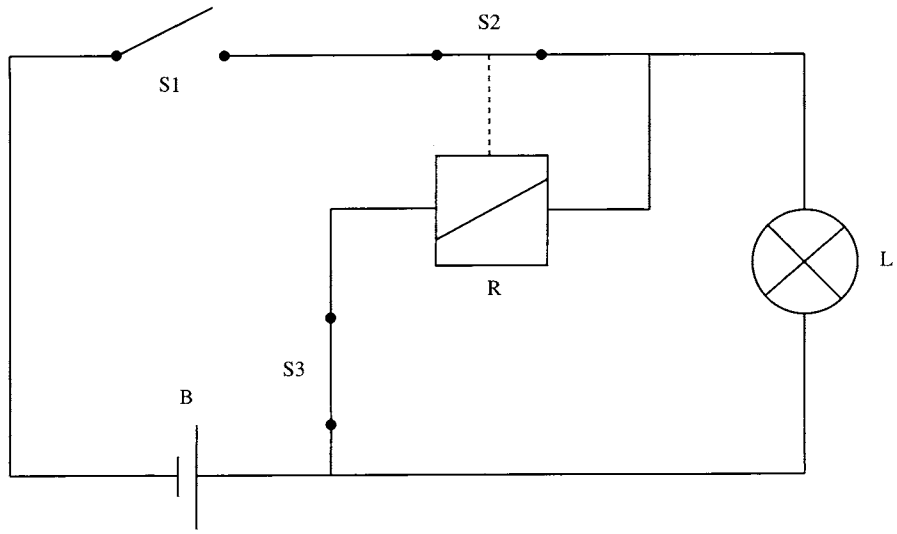


Figure 6.2 Shanahan's circuit (Shanahan, 1996b, p. 145).

$$HoldsAt(Closed(S2), t) \Rightarrow Happens(Light(L), t)$$

$$HoldsAt(Lit(L), t) \wedge (\neg HoldsAt(Closed(S1), t) \vee \neg HoldsAt(Closed(S2), t)) \Rightarrow Happens(Unlight(L), t) \tag{6.120}$$

$$HoldsAt(Closed(S2), t) \wedge HoldsAt(Activated(R), t) \Rightarrow Happens(Open(S2), t) \tag{6.121}$$

$$\neg HoldsAt(Activated(R), t) \wedge HoldsAt(Closed(S1), t) \wedge HoldsAt(Closed(S2), t) \wedge HoldsAt(Closed(S3), t) \Rightarrow Happens(Activate(R), t) \tag{6.122}$$

$$\begin{aligned} &\mathit{HoldsAt}(\mathit{Activated}(R), t) \wedge \\ &(\neg\mathit{HoldsAt}(\mathit{Closed}(S1), t) \vee \\ &\neg\mathit{HoldsAt}(\mathit{Closed}(S2), t) \vee \\ &\neg\mathit{HoldsAt}(\mathit{Closed}(S3), t)) \Rightarrow \\ &\mathit{Happens}(\mathit{Deactivate}(R), t) \end{aligned} \tag{6.123}$$

We add the following effect axiom to those already given:

$$\mathit{Terminates}(\mathit{Deactivate}(r), \mathit{Activated}(r), t) \tag{6.124}$$

We use the observations and narrative as detailed before.

With the conjunction of axioms EC, the circuit buzzes with infinite frequency, whereas with the conjunction of axioms DEC, it buzzes with finite frequency. Using DEC, we can show that R activates at timepoint 1, S2 opens at timepoint 2, and R deactivates at timepoint 3.

PROPOSITION 6.7

Let $\Sigma = (6.86) \wedge (6.87) \wedge (6.88) \wedge (6.89) \wedge (6.107) \wedge (6.124)$, $\Delta = (6.80) \wedge (6.119) \wedge (6.120) \wedge (6.121) \wedge (6.122) \wedge (6.123)$, $\Omega = U[\mathit{Close}, \mathit{Open}, \mathit{Activate}, \mathit{Deactivate}, \mathit{Light}, \mathit{Unlight}] \wedge U[\mathit{Closed}, \mathit{Activated}, \mathit{Lit}]$, and $\Gamma = (6.74) \wedge (6.75) \wedge (6.76) \wedge (6.77) \wedge (6.78) \wedge (6.79)$. Then we have

$$\begin{aligned} \mathit{CIRC}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathit{CIRC}[\Delta; \mathit{Happens}] \wedge \\ \Omega \wedge \Gamma \wedge \mathrm{DEC} \models \mathit{Happens}(\mathit{Activate}(R), 1) \wedge \\ \mathit{HoldsAt}(\mathit{Activated}(R), 2) \wedge \\ \mathit{Happens}(\mathit{Open}(S2), 2) \wedge \\ \neg\mathit{HoldsAt}(\mathit{Closed}(S2), 3) \wedge \\ \mathit{Happens}(\mathit{Deactivate}(R), 3) \wedge \\ \neg\mathit{HoldsAt}(\mathit{Activated}(R), 3) \end{aligned}$$

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

$$\begin{aligned} &\mathit{Initiates}(e, f, t) \Leftrightarrow \\ &\exists s\,(e = \mathit{Close}(s) \wedge f = \mathit{Closed}(s)) \vee \\ &\exists r\,(e = \mathit{Activate}(r) \wedge f = \mathit{Activated}(r)) \vee \\ &\exists l\,(e = \mathit{Light}(l) \wedge f = \mathit{Lit}(l)) \end{aligned} \tag{6.125}$$

$$\begin{aligned}
&\mathit{Terminates}(e,f,t) \Leftrightarrow \\
&\exists s\,(e = \mathit{Open}(s) \wedge f = \mathit{Closed}(s)) \vee \\
&\exists l\,(e = \mathit{Unlight}(l) \wedge \mathit{Lit}(l)) \vee \\
&\exists r\,(e = \mathit{Deactivate}(r) \wedge f = \mathit{Activated}(r))
\end{aligned} \tag{6.126}$$

$$\neg \mathit{Releases}(e,f,t) \tag{6.127}$$

From *CIRC*[Δ; *Happens*] and Theorem 2.1, we have

$$\begin{aligned}
&\mathit{Happens}(e,t) \Leftrightarrow \\
&(e = \mathit{Close}(S1) \wedge t = 0) \vee \\
&(e = \mathit{Light}(L) \wedge \\
&\neg \mathit{HoldsAt}(\mathit{Lit}(L),t) \wedge \\
&\mathit{HoldsAt}(\mathit{Closed}(S1),t) \wedge \\
&\mathit{HoldsAt}(\mathit{Closed}(S2),t)) \vee \\
&(e = \mathit{Unlight}(L) \wedge \\
&\mathit{HoldsAt}(\mathit{Lit}(L),t) \wedge \\
&(\neg \mathit{HoldsAt}(\mathit{Closed}(S1),t) \vee \\
&\neg \mathit{HoldsAt}(\mathit{Closed}(S2),t))) \vee \\
&(e = \mathit{Open}(S2) \wedge \\
&\mathit{HoldsAt}(\mathit{Closed}(S2),t) \wedge \\
&\mathit{HoldsAt}(\mathit{Activated}(R),t)) \vee \\
&(e = \mathit{Activate}(R) \wedge \\
&\neg \mathit{HoldsAt}(\mathit{Activated}(R),t) \wedge \\
&\mathit{HoldsAt}(\mathit{Closed}(S1),t) \wedge \\
&\mathit{HoldsAt}(\mathit{Closed}(S2),t) \wedge \\
&\mathit{HoldsAt}(\mathit{Closed}(S3),t)) \vee \\
&(e = \mathit{Deactivate}(R) \wedge \\
&\mathit{HoldsAt}(\mathit{Activated}(R),t) \wedge \\
&(\neg \mathit{HoldsAt}(\mathit{Closed}(S1),t) \vee \\
&\neg \mathit{HoldsAt}(\mathit{Closed}(S2),t) \vee \\
&\neg \mathit{HoldsAt}(\mathit{Closed}(S3),t)))
\end{aligned} \tag{6.128}$$

From (6.80) (which follows from (6.128)), (6.86) (which follows from (6.125)), and DEC9, we have

$$\mathit{HoldsAt}(\mathit{Closed}(S1), 1) \tag{6.129}$$

From (6.74) and (6.128), we have $\neg Happens(Activate(R), 0)$. From this and (6.125), we have $\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, Activated(R), 0))$. From this, (6.77), (6.79), and DEC6, we have

$$\neg HoldsAt(Activated(R), 1) \tag{6.130}$$

From (6.126), (6.128), and (6.77), we have $\neg\exists e\,(Happens(e, 0) \wedge Terminates(e, Closed(S2), 0))$. From this, (6.75), (6.79), and DEC5, we have

$$HoldsAt(Closed(S2), 1) \tag{6.131}$$

From (6.128) and (6.126), we have $\neg\exists e\,(Happens(e, 0) \wedge Terminates(e, Closed(S3), 0))$. From this, (6.76), (6.79), and DEC5, we have

$$HoldsAt(Closed(S3), 1) \tag{6.132}$$

From this, (6.130), (6.129), (6.131), and (6.122) (which follows from (6.128)), we have

$$Happens(Activate(R), 1)$$

From this, (6.88) (which follows from (6.125)), and DEC9, we have

$$HoldsAt(Activated(R), 2) \tag{6.133}$$

From (6.130) and (6.128), we have $\neg Happens(Open(S2), 1)$. From this and (6.126), we have $\neg\exists\, e\,(Happens(e, 1) \wedge Terminates(e, Closed(S2), 1))$. From this, (6.131), (6.79), and DEC5, we have

$$HoldsAt(Closed(S2), 2) \tag{6.134}$$

From this, (6.133), and (6.121) (which follows from (6.128)), we have

$$Happens(Open(S2), 2)$$

From this, (6.87) (which follows from (6.126)), and DEC10, we have

$$\neg HoldsAt(Closed(S2), 3) \tag{6.135}$$

From (6.126) and (6.128), we have $\neg\exists e\,(Happens(e, 1) \wedge Terminates(e, Closed(S1), 1))$. From this, (6.129), (6.79), and DEC5, we have

$$HoldsAt(Closed(S1), 2) \tag{6.136}$$

From (6.126) and (6.128), we have $\neg\exists e\,(Happens(e, 1) \wedge Terminates(e, Closed(S3), 1))$. From this, (6.132), (6.79), and DEC5, we have $HoldsAt(Closed(S3), 2)$. From this, (6.126), (6.128), (6.136), and (6.134), we have $\neg\exists e\,(Happens(e, 2) \wedge Terminates(Activated(R), 2))$. From this, (6.133),

(6.79), and DEC5, we have $HoldsAt(Activated(R), 3)$. From this, (6.135), and (6.123) (which follows from (6.128)), we have

$$Happens(Deactivate(R), 3)$$

From this, (6.124) (which follows from (6.126)), and DEC10, we have $\neg HoldsAt(Activated(R), 3)$. ■

Delayed indirect effects such as the results of dropping an object may be represented using trajectory and antitrajectory axioms, which are described in Chapter 7.

Bibliographic Notes

Ramification Problem

Minsky (1961) noted something similar to the frame and ramification problems:

One problem that has been a great nuisance to us arises in connection with non-mathematical problems in which actions affect the state of some subject domain. ... One must then deduce all the consequences of an action in so far as it affects propositions that one is planning to use. (p. 217)

Green (1969, p. 98) mentions a problem with representing the effects of pushing an object that accidentally hits and moves other objects. Waldinger (1977, pp. 124–129) discusses the difficulty of representing "actions with indirect side effects" (p. 125) using the add lists and delete lists of STRIPS (Fikes and Nilsson, 1971). He gives the examples of pushing a row of boxes to the right by pushing the leftmost box and moving each component of a complex object by moving the object.

Finger (1987) considers ramifications in the context of automating the design of such things as electronic circuits, computer programs, and robot plans. The task is to find a design (similar to a plan) D that achieves a goal specification G given a world model W. He defines a *ramification* as a formula N such that $W \models (D \wedge G) \supset N$ (p. 9) and provides the example that consuming fuel is a ramification of driving an automobile (p. 69). He presents methods for finding ramifications (pp. 92–126) and using them to (1) eliminate inconsistent goals if a ramification is known to be impossible, (2) reduce the search space by providing additional constraints, and (3) provide additional heuristics (pp. 70–71). He defines the *ramification problem* as the problem of having to anticipate and deal with an unbounded set of consequences of a design (p. 16).

Taking their inspiration from Finger, Ginsberg and Smith (1987a, 1988a) define the *ramification problem* as the problem representing all the consequences

of an action. They propose an approach to the ramification problem based on state constraints and possible worlds. A *possible world* is defined as a set of logical formulas. The consequences of an action performed in a world S are determined by finding the nearest possible worlds to S in which the direct effects of the action hold. They consider what a household robot would have to reason about and formalize a room in a house with a television set, plant, two ventilation ducts, and other objects. A state constraint represents that the room is stuffy if and only if both ventilation ducts are blocked. When a plant is blocking one duct and a television set is moved on top of the other duct, two nearest possible worlds result: one in which both ducts are blocked and the room is stuffy, and another in which the plant is no longer blocking a duct, the television set is blocking a duct, and the room is not stuffy.

Haas (1987) uses the example of a robot carrying a box and encodes the location of the box (*InRoom*) using a state constraint. Schubert (1990) uses the robot example and encodes "implicit effects" (p. 32) within both state constraints (his axiom A8, p. 33) and explanation closure axioms (his axiom A20, p. 35). The robot examples derive from the robot world model of Fikes and Nilsson (1971) and Fikes et al. (1972a) and from the blocks world (Winston, 1970; Winograd, 1972; Sacerdoti, 1977).

In the formulation of the situation calculus of Reiter (1991, 2001) and Pirri and Reiter (1999), a successor state axiom for a fluent specifies the truth value of the fluent in the successor situation $do(\alpha, \sigma)$ that results from performing action α in situation σ. Thus, a successor axiom for a fluent completely specifies how actions affect the fluent. Reiter (1991; 2001, 23–35) presents a method for automatically compiling effect axioms into successor state axioms in order to solve the frame problem. Lin and Reiter (1994, sec. 5) and Reiter (2001, pp. 401–402) propose a method for dealing with state constraints in the situation calculus. The method involves making the effects implied by state constraints explicit. Additional effect axioms are derived from state constraints, and then effect axioms are compiled into successor state axioms as usual. They use an example of painting blocks. The action $paint(b, c)$ represents painting block b color c. An effect axiom states that the color of block b in the situation $do(paint(b, c), s)$ is c. A state constraint says that a block has a unique color in a situation. Their method arrives at the indirect effect that a block is no longer red after it is painted yellow. Note that we would not treat this as an indirect effect but would rather simply have the two (direct) effect axioms as follows:

$$Initiates(Paint(b, c), Color(b, c), t)$$

$$HoldsAt(Color(b, c_1), t) \wedge c_1 \neq c_2 \Rightarrow$$
$$Terminates(Paint(b, c_2), Color(b, c_1), t)$$

Similarly, Baral (2003, pp. 233–238) considers the indirect effects resulting from the constraint that an object can be at one location at a time.

Effect Axioms

Kakas and Miller (1997a, p. 6) point out that, if effect axioms alone are used to represent indirect effects, then incompletely specified initial situations can lead to unintended models. Suppose that the observations and narrative in the example in Section 6.1 are empty. Then, there are models of the domain description in which the following formulas are true:

$$HoldsAt(Holding(Nathan, Book), 0)$$
$$HoldsAt(InRoom(Nathan, Kitchen), 0)$$
$$HoldsAt(InRoom(Book, LivingRoom), 0)$$

Primitive and Derived Fluents

The distinction between primitive and derived fluents goes back to STRIPS (Fikes and Nilsson, 1971), which had the notions of a "primitive clause" and a "derived clause" (p. 198). Fahlman (1974, pp. 18–20) distinguishes "primary data" and "secondary data." Kowalski (1979) classifies relations into "primitive" and "derived" (p. 137) in his version of the situation calculus. Lifschitz (1987a) distinguishes "primitive fluents" (p. 45) and fluents "defined in terms of primitive fluents" (p. 44). He writes a state constraint that defines *clear* in terms of the primitive fluent *at* (p. 50):

$$holds(clear\ l, s) \equiv \forall b\ \neg holds(at(b, l), s) \tag{6.137}$$

Lifschitz (1990b) makes a similar distinction between "frame fluents" and "arbitrary fluents" (p. 370). Shanahan (1999b) discusses the division of fluents into "primitive" and "derived" (p. 141) in the event calculus. State constraints for representing ramifications were introduced into the classical logic event calculus by Shanahan (1995a, pp. 262–264; 1997b, pp. 323–325). The reasons for using state constraints to encode indirect effects rather than effect axioms include "[avoiding] duplication of the same information on the consequent side of several action laws" (Sandewall, 1996, p. 99) and "[ensuring] a modular representation and ... dramatically [shortening] an axiomatisation" (Shanahan, 1999b, p. 141). Shanahan (1997b, pp. 123–124) calculates that the number of effect axioms needed to replace a state constraint

$$HoldsAt(\beta_1, t) \wedge \cdots \wedge HoldsAt(\beta_n, t) \Rightarrow HoldsAt(\beta, t)$$

is on the order of $\sum_{i=1}^{n} m_i$, where m_i is the number of effect axioms for β_i. Baral (2003) argues for the use of constraints rather than effect axioms to represent ramifications and uses the example of having to add separate effect axioms for the indirect effects of every type of transportation: "*drive_to*(*Y*), *fly_to*(*Y*), *take_a_train_to*(*Y*), *take_a_bus_to*(*Y*)" (p. 233).

Walking Turkey Scenario

Baker (1991, pp. 19–21) first described the walking turkey scenario due to Matthew L. Ginsberg: When a turkey that is walking is shot, it is no longer walking. Lin (1995) treats the ramification problem in the situation calculus by introducing a predicate $Caused(p, v, s)$, which represents that fluent p is caused to have truth value v in situation v. (A similar proposal is made by McCain and Turner, 1995.) The walking turkey scenario is treated as follows. The direct effect of shooting is specified by the effect axiom (Lin, 1995, p. 1989).

$$Caused(dead, true, do(shoot, s))$$

The indirect effect of shooting on walking is specified by the "causal rule" (p. 1990):

$$dead(s) \supset Caused(walking, false, s)$$

The following successor state axiom for *dead* follows from the predicate completion (K. L. Clark, 1978, pp. 303–305) of *Caused* in the conjunction of effect axioms and causal rules (simplified from Lin, 1995, p. 1990):

$$walking(do(a, s)) \equiv$$
$$walking(s) \wedge \neg dead(s) \wedge a \neq shoot$$

Thus, the following indirect effect of *shoot* is derived (simplified from p. 1990):

$$\neg walking(do(shoot, s))$$

Gustafsson and Doherty (1996) treat indirect effects in the features and fluents framework, which is discussed in Section 15.2. They handle the walking turkey scenario using the "causal constraint" (we would handle this using an effect constraint) (p. 93)

$$\forall t[t]\ \neg alive \gg \neg walking$$

which gets translated into (p. 93)

$$\forall t[(\neg Holds(t, alive) \rightarrow \neg Holds(t, walking)) \wedge$$
$$(Holds(t-1, alive) \wedge \neg Holds(t, alive) \rightarrow$$
$$Occlude(t, walking))]$$

The event calculus analog of Gustafsson and Doherty's treatment of the walking turkey scenario is:

$$\begin{gathered} \mathit{Terminates}(\mathit{Shoot}(a,b),\mathit{Alive}(b),t) \\ \mathit{Releases}(\mathit{Shoot}(a,b),\mathit{Walking}(b),t) \\ \neg\mathit{HoldsAt}(\mathit{Alive}(a),t) \Rightarrow \neg\mathit{HoldsAt}(\mathit{Walking}(a),t). \end{gathered}$$

Effect Constraints

Shanahan (1997b, pp. 286–288, 324–325; 1999b, pp. 141–142) devised the effect constraint as a simple way of handling the walking turkey scenario in the event calculus. We need only add the effect constraint:

$$\mathit{Terminates}(e,\mathit{Alive}(a),t) \Rightarrow \mathit{Terminates}(e,\mathit{Walking}(a),t)$$

Fluent Calculus

Thielscher (1996, 1997) proposes the following method to deal with indirect effects in the fluent calculus. We are given a set of direct effect descriptions, causal relationships, and state constraints. A causal relationship is an expression such as (Thielscher, 1997, p. 323)

$$sw_1 \ \mathit{causes}\ \mathit{light}\ \mathit{if}\ sw_2$$

The successor state resulting from performing an action α in a state of the world is computed as follows. The direct effects of α are first computed, giving the first intermediate state. Zero or more additional intermediate states are obtained by repeatedly applying causal relationships to intermediate states. Any intermediate state satisfying the state constraints is a successor state. Thus, causal relationships are permitted to interact with one another before a successor state is obtained. A successor state is not guaranteed to exist, and there may be several successor states. Thielscher also proposes a method for automatically compiling state constraints into causal relationships.

Causal Constraints

The circuit in Figure 6.1 is taken from Thielscher (1996, p. 19). This circuit is a slight modification of the circuit of Thielscher (1995), which in turn was inspired by Lifschitz's (1990b, pp. 371–372) circuit consisting of two switches and a light. Thielscher's circuit, which is handled by the methods he proposed, led to the introduction of causal constraints into the event calculus by Shanahan (1999b). The event calculus axioms for causal constraints and the axiomatization of Thielscher's circuit are from Shanahan (1999b). Our proof of Proposition 6.5 is based on that of Shanahan (1999b, pp. 143–144). The modified version of Thielscher's circuit

(Figure 6.2) is taken from Shanahan (1999b, p. 145), who proves that a formalization of the circuit using causal constraints results in inconsistency. Shanahan (1999b, p. 146) suggests that this circuit could be more properly modeled by taking delays into account.

Exercises

6.1 Suppose that in the example in Section 6.5.1, (6.90) is replaced with the state constraint

$$HoldsAt(Lit(L), t) \Leftrightarrow HoldsAt(Closed(S1), t) \wedge HoldsAt(Closed(S2), t)$$

Does the axiomatization still yield the conclusion that the light is off?

6.2 Suppose that in the example in Section 6.5.1, *Initiated* is replaced with *Started* in (6.90), (6.91), and (6.92). Does the axiomatization yield the conclusion that the light is off?

6.3 Formalize the circuit of Figure 6.1 using effect axioms, effect constraints, and state constraints instead of causal constraints. Discuss the advantages and disadvantages of this axiomatization as compared to that using causal constraints.

6.4 Formalize the following example from Denecker, Dupré, and Belleghem (1998, pp. 4–5): An electronic counter is connected to an input line. The value of the counter is initially zero. The value of the input line is initially false. The value of the input line may fluctuate arbitrarily between false and true. The counter counts how many times the input line changes from false to true. That is, whenever the value of the input line changes from false to true, the value of the counter is incremented by one.

6.5 (Research Problem) Automate the translation of circuit diagrams into event calculus axiomatizations.

CHAPTER 7

Continuous Change

Discrete change is change that is limited to a countable, usually finite, set of timepoints. We represent discrete change in the event calculus using effect axioms. In a number of commonsense domains ranging from the physical to the mental, we find continuous change. A function $f(t)$ is *continuous at a point* t_0 if and only if for every $\epsilon > 0$, there is a $\delta > 0$ such that $|t - t_0| < \delta$ implies $|f(t) - f(t_0)| < \epsilon$. A function is *continuous on an interval* if and only if it is continuous at all points of the interval. Examples of continuous change include the change in the height of a falling object, the location of a projectile, the water level of a filling bathtub, the volume of a balloon in the process of inflation, the frequency of a siren, the hunger level of an animal, and the anger level of a person. In the discrete event calculus, time is limited to the integers; there we speak of gradual change. In this chapter, we discuss the representation of continuous change and gradual change in the event calculus. We retrace the development of *Trajectory*. We discuss the use of trajectory and antitrajectory axioms.

7.1 Trajectory Axioms

How should we represent continuous change? Let us consider the changing height of a falling object without air resistance.

7.1.1 Example: Falling Object

An object O1 is at rest at height H prior to timepoint T_1, starts to fall at timepoint T_1, and hits the ground at timepoint $T_2 = T_1 + \sqrt{2H/G}$, where G is the acceleration

due to gravity (9.8 m/s^2):

$$t \leq T_1 \Rightarrow HoldsAt(Height(O1, H), t) \tag{7.1}$$

$$t > T_1 \wedge t \leq T_2 \Rightarrow HoldsAt(Height(O1, H - \frac{1}{2}G(t - T_1)^2), t) \tag{7.2}$$

$$t > T_2 \Rightarrow HoldsAt(Height(O1, 0), t) \tag{7.3}$$

We have a state constraint that says that an object has a unique height at any given time:

$$HoldsAt(Height(o, h_1), t) \wedge HoldsAt(Height(o, h_2), t) \Rightarrow h_1 = h_2 \tag{7.4}$$

The height of O1 is released from the commonsense law of inertia:

$$ReleasedAt(Height(O1, h), t) \tag{7.5}$$

From this axiomatization, we can determine the height of O1 at any timepoint. For instance, if $(T_1 + 2) \leq T_2$, then the height of O1 at timepoint $T_1 + 2$ is $H - 2G$.

PROPOSITION 7.1 If $(T_1 + 2) \leq T_2$, Ψ = (7.4), and Γ = (7.1) $\wedge$ (7.2) $\wedge$ (7.3) $\wedge$ (7.5), then $\Psi \wedge \Gamma \models HoldsAt(Height(O1, H - 2G), T_1 + 2)$.

Proof From $(T_1 + 2) > T_1$, $(T_1 + 2) \leq T_2$, and (7.2), we have $HoldsAt(Height(O1, H - \frac{1}{2}G((T_1 + 2) - T_1)^2), T_1 + 2)$ or $HoldsAt(Height(O1, H - 2G), T_1 + 2)$. ■

Although this axiomatization describes a falling object, it fails to capture our commonsense knowledge about falling objects. Namely, when an object is dropped, it starts to fall; when an object hits the ground, it stops falling. Therefore, we integrate events into our representation of continuous change. One event will set in motion an episode of continuous change, whereas another event will stop the episode. We use fluents to represent episodes of continuous change.

7.1.2 Example: Falling Object with Events

We model a falling object using events. We have an episode of no change (the object at rest above the ground), followed by an episode of continuous change (falling), followed by another episode of no change (the object hits the ground).

We use the following axiomatization. If an agent drops an object, then the object will be falling, and the height of the object will be released from the commonsense law of inertia:

$$Initiates(Drop(a, o), Falling(o), t) \tag{7.6}$$

$$Releases(Drop(a, o), Height(o, h), t) \tag{7.7}$$

In EC and DEC, events are instantaneous. Notice that we are using the fluent *Falling* here as a way of representing an event with duration or a process. (We present a modification of EC for events with duration in Appendix C.)

We describe the motion of the object from the moment it is dropped until it stops falling as follows:

$$\begin{aligned} &HoldsAt(Height(o,h),t_1) \wedge \\ &Happens(Drop(a,o),t_1) \wedge 0 < t_2 \wedge \\ &\neg StoppedIn(t_1, Falling(o), t_1 + t_2) \Rightarrow \\ &HoldsAt(Height(o, h - \frac{1}{2}Gt_2^2), t_1 + t_2) \end{aligned} \tag{7.8}$$

If an object hits the ground, then the object will no longer be falling:

$$Terminates(HitGround(o), Falling(o), t) \tag{7.9}$$

A falling object hits the ground when its height becomes zero:

$$\begin{aligned} &HoldsAt(Falling(o),t) \wedge \\ &HoldsAt(Height(o,0),t) \Rightarrow \\ &Happens(HitGround(o),t) \end{aligned} \tag{7.10}$$

If an object hits the ground and the height of the object is h, then the height of the object will no longer be released from the commonsense law of inertia and the height of the object will be h:

$$\begin{aligned} &HoldsAt(Height(o,h),t) \Rightarrow \\ &Initiates(HitGround(o), Height(o,h), t) \end{aligned} \tag{7.11}$$

An object has a unique height:

$$\begin{aligned} &HoldsAt(Height(o,h_1),t) \wedge HoldsAt(Height(o,h_2),t) \Rightarrow \\ &h_1 = h_2 \end{aligned} \tag{7.12}$$

At timepoint 0, Nathan drops an apple whose height is G/2:

$$\neg HoldsAt(Falling(Apple), 0) \tag{7.13}$$

$$HoldsAt(Height(Apple, G/2), 0) \tag{7.14}$$

$$Happens(Drop(Nathan, Apple), 0) \tag{7.15}$$

We can show that the apple will hit the ground at timepoint 1 and that its height at timepoint 2 will be zero.

PROPOSITION 7.2 Let $\Sigma = (7.6) \wedge (7.7) \wedge (7.9) \wedge (7.11)$, $\Delta = (7.10) \wedge (7.15)$, $\Omega = U[Drop, HitGround] \wedge U[Falling, Height]$, $\Psi = (7.12)$, $\Pi = (7.8)$, and $\Gamma = (7.13) \wedge (7.14)$. Then we have

$$\begin{aligned} &CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ &\qquad \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge EC \\ &\models HoldsAt(Height(Apple, 0), 1) \wedge \\ &\quad Happens(HitGround(Apple), 1) \wedge \\ &\qquad HoldsAt(Height(Apple, 0), 2) \end{aligned}$$

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

$$\begin{aligned} &Initiates(e, f, t) \Leftrightarrow \\ &\exists a, o\,(e = Drop(a, o) \wedge f = Falling(o)) \vee \\ &\exists o, h\,(e = HitGround(o) \wedge \\ &f = Height(o, h) \wedge \\ &HoldsAt(Height(o, h), t)) \end{aligned} \tag{7.16}$$

$$\begin{aligned} &Terminates(e, f, t) \Leftrightarrow \\ &\exists o\,(e = HitGround(o) \wedge f = Falling(o)) \end{aligned} \tag{7.17}$$

$$\begin{aligned} &Releases(e, f, t) \Leftrightarrow \\ &\exists a, o, h\,(e = Drop(a, o) \wedge f = Height(o, h)) \end{aligned} \tag{7.18}$$

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

$$\begin{aligned} &Happens(e, t) \Leftrightarrow \\ &\exists o\,(e = HitGround(o) \wedge \\ &HoldsAt(Falling(o), t) \wedge \\ &HoldsAt(Height(o, 0), t)) \vee \\ &(e = Drop(Nathan, Apple) \wedge t = 0) \end{aligned} \tag{7.19}$$

We can show

$$\neg\exists t\,(0 < t < 1 \wedge Happens(HitGround(Apple), t)) \tag{7.20}$$

To see this, suppose, to the contrary, that

$$\exists t\,(0 < t < 1 \wedge Happens(HitGround(Apple), t))$$

Let $Happens(HitGround(Apple), \tau)$ be the first such event. That is, we have

$$0 < \tau < 1 \tag{7.21}$$

$$Happens(HitGround(Apple), \tau) \tag{7.22}$$

$$\neg\exists\tau'\,(0 < \tau' < \tau \wedge Happens(HitGround(Apple), \tau')) \tag{7.23}$$

From (7.22) and (7.19), we have

$$HoldsAt(Height(Apple, 0), \tau) \tag{7.24}$$

From (7.17), (7.23), and EC3, we have $\neg StoppedIn(0, Falling(Apple), \tau)$. From this, (7.14), (7.15) (which follows from (7.19)), $0 < \tau$ (which follows from (7.21)), and (7.8), we have

$$HoldsAt(Height(Apple, \frac{1}{2}G(1 - \tau^2)), \tau)$$

From this, (7.21), and (7.12), we have $\neg HoldsAt(Height(Apple, 0), \tau)$, which contradicts (7.24).

From (7.17), (7.20), and EC3, we have

$$\neg StoppedIn(0, Falling(Apple), 1) \tag{7.25}$$

From this, (7.14), (7.15) (which follows from (7.19)), $0 < 1$, and (7.8), we have

$$HoldsAt(Height(Apple, 0), 1) \tag{7.26}$$

From (7.18), we have

$$\neg ReleasedIn(0, Falling(Apple), 1)$$

From this, (7.15) (which follows from (7.19)), (7.6) (which follows from (7.16)), $0 < 1$, (7.25), and EC14, we have $HoldsAt(Falling(Apple), 1)$. From this, (7.26), and (7.10) (which follows from (7.19)), we have

$$Happens(HitGround(Apple), 1) \tag{7.27}$$

From (7.17) and EC3, we have

$$\neg StoppedIn(1, Height(Apple, 0), 2) \tag{7.28}$$

From (7.18), (7.19), and EC13, we have $\neg ReleasedIn(1, Height(Apple, 0), 2)$. From this, (7.27), (7.26), (7.11) (which follows from (7.16)), $1 < 2$, (7.28), and EC14, we have $HoldsAt(Height(Apple,0),2)$. ■

7.1.3 Introduction of *Trajectory* Predicate

We introduce the predicate $Trajectory(\beta_1, \tau_1, \beta_2, \tau_2)$, which allows us to express (7.8) more compactly.

DEFINITION 7.1

If γ is a condition, β_1 and β_2 are fluent terms, and τ_1 and τ_2 are timepoint terms, then

$$\gamma \Rightarrow Trajectory(\beta_1, \tau_1, \beta_2, \tau_2)$$

is a ***trajectory axiom***. This represents that, if γ is true, fluent β_1 is initiated by an event that occurs at timepoint τ_1, and $\tau_2 > 0$, then fluent β_2 will be true at timepoint $\tau_1 + \tau_2$.

Thus, instead of (7.8) we may simply write:

$$HoldsAt(Height(o, h), t_1) \Rightarrow \tag{7.29}$$
$$Trajectory(Falling(o), t_1, Height(o, h - \frac{1}{2}Gt_2^2), t_2)$$

Using the *Trajectory* predicate, we show that the apple hits the ground at timepoint 1 and that it has zero height at timepoint 2.

PROPOSITION 7.3

Let $\Sigma = (7.6) \wedge (7.7) \wedge (7.9) \wedge (7.11)$, $\Delta = (7.10) \wedge (7.15)$, $\Omega = U[Drop, HitGround] \wedge U[Falling, Height]$, $\Psi = (7.12)$, $\Pi = (7.29)$, and $\Gamma = (7.13) \wedge (7.14)$. Then we have

$$\begin{aligned} CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge EC \\ \models HoldsAt(Height(Apple, 0), 1) \wedge \\ Happens(HitGround(Apple), 1) \wedge \\ HoldsAt(Height(Apple, 0), 2) \end{aligned}$$

Proof The proof is identical to that of Proposition 7.2, except that (7.8) is replaced with the conjunction of (7.29), (7.6) (which follows from (7.16)), and EC5. ■

7.2 Antitrajectory Axioms

The *AntiTrajectory* predicate is analogous to the *Trajectory* predicate, except that it is brought into play when a fluent is terminated rather than initiated by an occurring event.

DEFINITION 7.2

If γ is a condition, β_1 and β_2 are fluent terms, and τ_1 and τ_2 are timepoint terms, then

$$\gamma \Rightarrow AntiTrajectory(\beta_1, \tau_1, \beta_2, \tau_2)$$

is an ***antitrajectory axiom***. This represents that if γ is true, fluent β_1 is terminated by an event that occurs at timepoint τ_1, and $\tau_2 > 0$, then fluent β_2 will be true at timepoint $\tau_1 + \tau_2$.

7.2.1 Example: Hot Air Balloon

This example and the trajectory and antitrajectory axioms are due to Rob Miller and Murray Shanahan. We have two effect axioms. If an agent turns on the heater of a hot air balloon, then the heater will be on:

$$Initiates(TurnOnHeater(a, b), HeaterOn(b), t) \tag{7.30}$$

If an agent turns off the heater of a hot air balloon, then the heater will no longer be on:

$$Terminates(TurnOffHeater(a, b), HeaterOn(b), t) \tag{7.31}$$

We have a state constraint:

$$HoldsAt(Height(b, h_1), t) \wedge HoldsAt(Height(b, h_2), t) \Rightarrow h_1 = h_2 \tag{7.32}$$

We have the following trajectory and antitrajectory axioms:

$$HoldsAt(Height(b, h), t_1) \Rightarrow Trajectory(HeaterOn(b), t_1, Height(b, h + V \cdot t_2), t_2) \tag{7.33}$$

$$HoldsAt(Height(b, h), t_1) \Rightarrow AntiTrajectory(HeaterOn(b), t_1, Height(b, h - V \cdot t_2), t_2) \tag{7.34}$$

where V is the velocity of the balloon.

We have the following observations and narrative. The balloon starts out at height 0:

$$HoldsAt(Height(Balloon, 0), 0) \tag{7.35}$$

The height of a balloon is always released from the commonsense law of inertia:

$$ReleasedAt(Height(b, h), t) \tag{7.36}$$

At timepoint 0 the heater is turned on, and at timepoint 2 the heater is turned off:

$$Happens(TurnOnHeater(Nathan, Balloon), 0) \tag{7.37}$$
$$Happens(TurnOffHeater(Nathan, Balloon), 2) \tag{7.38}$$

We can show that the height of the balloon is $2V$ at timepoint 2 and is V at timepoint 3.

PROPOSITION 7.4 Let $\Sigma = (7.30) \wedge (7.31)$, $\Delta = (7.37) \wedge (7.38)$, $\Omega = U[TurnOnHeater, TurnOffHeater] \wedge U[HeaterOn, Height]$, $\Psi = (7.32)$, $\Pi = (7.33) \wedge (7.34)$, and $\Gamma = (7.35) \wedge (7.36)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge EC \models HoldsAt(Height(Balloon, 2V), 2) \wedge HoldsAt(Height(Balloon, V), 3)$$

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

$$Initiates(e, f, t) \Leftrightarrow \exists a, b\,(e = TurnOnHeater(a, b) \wedge HeaterOn(b)) \tag{7.39}$$

$$Terminates(e, f, t) \Leftrightarrow \exists a, b\,(e = TurnOffHeater(a, b) \wedge HeaterOn(b)) \tag{7.40}$$

$$\neg Releases(e, f, t) \tag{7.41}$$

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

$$Happens(e, t) \Leftrightarrow (e = TurnOnHeater(Nathan, Balloon) \wedge t = 0) \wedge (e = TurnOffHeater(Nathan, Balloon) \wedge t = 2) \tag{7.42}$$

From (7.42) and EC3, we have $\neg StoppedIn(0, HeaterOn(Balloon), 2)$. From this, (7.37) (which follows from (7.42)), (7.30) (which follows from (7.39)), $0 < 2$, (7.35), (7.33), and EC5, we have

$$HoldsAt(Height(Balloon, 2V), 2) \tag{7.43}$$

From (7.42) and EC4, we have $\neg StartedIn(2, HeaterOff(Balloon), 3)$. From this, (7.38) (which follows from (7.42)), (7.31) (which follows from (7.40)), $0 < 1$, (7.43), (7.34), and EC6, we have $HoldsAt(Height(Balloon, V), 3)$. ■

7.3 Using *AntiTrajectory* Instead of *Releases*

We may sometimes use *AntiTrajectory* to replace *Releases*.

7.3.1 Example: Falling Object with *AntiTrajectory*

Consider again the example of the falling object. First, we state that the height of an object is always released from the commonsense law of inertia:

$$ReleasedAt(Height(o, h), t) \tag{7.44}$$

Then, we replace (7.7) and (7.11) with

$$HoldsAt(Height(o, h), t_1) \Rightarrow \\ AntiTrajectory(Falling(o), t_1, Height(o, h), t_2) \tag{7.45}$$

That is, when an object stops falling, its height stays constant.

We can show that the apple hits the ground at timepoint 1, and has a height of zero at timepoint 2.

PROPOSITION 7.5 Let $\Sigma = (7.6) \wedge (7.9)$, $\Delta = (7.10) \wedge (7.15)$, $\Omega = U[Drop, HitGround] \wedge U[Falling, Height]$, $\Psi = (7.12)$, $\Pi = (7.29) \wedge (7.45)$, and $\Gamma = (7.13) \wedge (7.14) \wedge (7.44)$. Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge \mathrm{EC} \\ \models HoldsAt(Height(Apple, 0), 1) \wedge \\ Happens(HitGround(Apple), 1) \wedge \\ HoldsAt(Height(Apple, 0), 2)$$

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

$$Initiates(e, f, t) \Leftrightarrow \\ \exists a, o\,(e = Drop(a, o) \wedge f = Falling(o)) \tag{7.46}$$

$$\begin{aligned} &\mathit{Terminates}(e,f,t) \Leftrightarrow \\ &\exists o\,(e = \mathit{HitGround}(o) \wedge f = \mathit{Falling}(o)) \end{aligned} \tag{7.47}$$

$$\neg \mathit{Releases}(e,f,t) \tag{7.48}$$

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

$$\begin{aligned} &\mathit{Happens}(e,t) \Leftrightarrow \\ &\exists o\,(e = \mathit{HitGround}(o) \wedge \\ &\mathit{HoldsAt}(\mathit{Falling}(o),t) \wedge \\ &\mathit{HoldsAt}(\mathit{Height}(o,0),t)) \vee \\ &(e = \mathit{Drop}(\mathit{Nathan},\mathit{Apple}) \wedge t = 0) \end{aligned} \tag{7.49}$$

We can show

$$\neg\exists t\,(0 < t < 1 \wedge \mathit{Happens}(\mathit{HitGround}(\mathit{Apple}),t)) \tag{7.50}$$

Suppose for contradiction that

$$\exists t\,(0 < t < 1 \wedge \mathit{Happens}(\mathit{HitGround}(\mathit{Apple}),t))$$

Let $\mathit{Happens}(\mathit{HitGround}(\mathit{Apple}),\tau)$ be the first such event. Thus we have

$$0 < \tau < 1 \tag{7.51}$$

$$\mathit{Happens}(\mathit{HitGround}(\mathit{Apple}),\tau). \tag{7.52}$$

$$\neg\exists \tau'\,(0 < \tau' < \tau \wedge \mathit{Happens}(\mathit{HitGround}(\mathit{Apple}),\tau')) \tag{7.53}$$

From (7.52) and (7.49), we have

$$\mathit{HoldsAt}(\mathit{Height}(\mathit{Apple},0),\tau) \tag{7.54}$$

From (7.47), (7.53), and EC3, we have $\neg\mathit{StoppedIn}(0,\mathit{Falling}(\mathit{Apple}),\tau)$. From this, (7.15) (which follows from (7.49)), (7.6) (which follows from (7.46)), $0 < \tau$ (which follows from 7.51), (7.14), (7.29), and EC5, we have

$$\mathit{HoldsAt}(\mathit{Height}(\mathit{Apple},\frac{1}{2}G(1-\tau^2)),\tau)$$

From this, (7.51), and (7.12), we have $\neg\mathit{HoldsAt}(\mathit{Height}(\mathit{Apple},0),\tau)$, which contradicts (7.54).

From (7.47), (7.50), and EC3, we have

$$\neg\mathit{StoppedIn}(0,\mathit{Falling}(\mathit{Apple}),1) \tag{7.55}$$

From this, (7.15) (which follows from (7.49)), (7.6) (which follows from (7.46)), $0 < 1$, (7.14), (7.29), and EC5, we have

$$HoldsAt(Height(Apple, 0), 1) \tag{7.56}$$

From (7.48), we have

$$\neg ReleasedIn(0, Falling(Apple), 1)$$

From this, (7.55), (7.15) (which follows from (7.49)), (7.6) (which follows from (7.46)), $0 < 1$, and EC14, we have $HoldsAt(Falling(Apple), 1)$. From this, (7.56), and (7.10) (which follows from (7.49)), we have

$$Happens(HitGround(Apple), 1) \tag{7.57}$$

From (7.46), (7.49), and EC4, we have $\neg StartedIn(1, Falling(Apple), 2)$. From this, (7.57), (7.9) (which follows from (7.47)), $0 < 1$, (7.56), (7.45), and EC6, we have $HoldsAt(Height(Apple, 0), 2)$. ■

Bibliographic Notes

Hendrix (1973) extends the language of STRIPS (Fikes and Nilsson, 1971) to deal with "continuous, gradual change" (Hendrix, 1973, p. 146). A process such as filling a bucket is subject to "continuance conditions" (p. 161), or conditions necessary for the process to continue. Filling a bucket has the continuance conditions that the tap must be turned on, the bucket must be facing up, and the amount of water in the bucket must be less than its capacity. When any such condition is violated, the filling process is interrupted. Proposals for integrating continuous change into the situation calculus have been made by Gelfond et al. (1991), Pinto (1994), and R. Miller and Shanahan (1994). See the discussion in the Bibliographic Notes of Chapter 15. Proposals have been made for other formalisms for reasoning about action and change (McDermott, 1982; Sandewall, 1989a; Van Belleghem, Denecker, and De Schreye, 1994; Herrmann and Thielscher, 1996). The area of qualitative reasoning, discussed in Section 16.1, deals with continuous change. Shanahan (1995a) introduced the classical logic event calculus, which included the *Trajectory* predicate (p. 268). This predicate was previously introduced by Shanahan (1990) into a simplified version of the original event calculus (Kowalski and Sergot, 1986).

The example of a falling object has been used by McCarthy (1963), McCarthy and Hayes (1969, p. 479), and Shanahan (1990, p. 598). The method for showing $\neg StoppedIn(0, Falling(Apple), 1)$ in the proofs of Propositions 7.2, 7.3, 7.5, and 10.2 is taken from Shanahan (1997b, pp. 328–329; 2004, p. 161). Reiter (2001, pp. 149–150) discusses the use of fluents to represent processes or actions with

duration. The *AntiTrajectory* predicate was introduced by R. Miller and Shanahan (1999, pp. 13–15; 2002, pp. 466–470). The hot air balloon example is from R. Miller and Shanahan (1999, p. 14; 2002, p. 468). Shanahan (1990) shows how multiple flows into a vessel can be handled. Van Belleghem et al. (1994) show how to handle multiple simultaneous (and possibly continuously changing) influences on a quantity. R. Miller and Shanahan (1996) treat continuously changing parameters in the event calculus using differential equations. We have discussed trajectories in the context of continuous change. But as R. Miller and Shanahan (2002, p. 468) note, trajectories are not required to be continuous. They give the example of using a trajectory axiom to model a blinking light.

Exercises

7.1 Formalize an expanding balloon.

7.2 Formalize the following. A person's level of hunger gradually increases. When a person's hunger level rises above a certain threshold, the person starts to eat. When a person finishes eating, the person's hunger level is lowered and starts to increase again.

7.3 Extend the formalization of the falling object in Section 7.1.2 so that (1) an object P is initially resting on a high object Q; (2) when P is pushed, P is no longer on Q and P starts falling; and (3) after P hits the ground, P rests on the ground.

7.4 Extend the formalization of the hot air balloon in Section 7.2.1 to deal with what happens when the balloon hits the ground.

7.5 Formalize a puck bouncing off a wall in the game of air hockey. Assume there is no friction. Assume there is a horizontal wall at $y = 0$ and a puck. If the y coordinate of a moving puck is 0, then the puck will bounce off the wall.

7.6 Extend the formalization of air hockey so that an agent may hit the puck with a mallet.

7.7 Formalize a rocket with constant thrust. When the thrust is on, it moves upward with an acceleration of $T - G$, where T is the acceleration due to the thrust. When the thrust is off, it falls with an acceleration of G.

7.8 Formalize snow falling from the sky. When it is snowing, snowflakes fall. Model the falling of individual snowflakes.

CHAPTER 8

Concurrent Events

The event calculus allows several events to occur at the same time, because we may have event occurrences $Happens(\alpha_1, \tau_1)$ and $Happens(\alpha_2, \tau_2)$ such that $\alpha_1 \neq \alpha_2$ and $\tau_1 = \tau_2$. In this case, we say that the event occurrences are concurrent or simultaneous. In this chapter, we first consider some concurrent event occurrences that we might want to prohibit and methods for prohibiting them. Then, we discuss concurrent event occurrences that have cumulative and canceling effects and how to handle them.

8.1 Restricting Concurrency

The commonsense world places certain restrictions on what events can occur simultaneously. This section presents two methods for representing these restrictions in the event calculus: (1) state constraints and (2) event occurrence constraints.

8.1.1 State Constraints

The first method of restricting concurrency is to use state constraints. This method relies on the fact that, if the effects of two events are inconsistent with a state constraint, then those events cannot occur concurrently. Consider an agent that can walk from one room to another. We have the effect axioms:

$$Initiates(Walk(a, r_1, r_2), InRoom(a, r_2), t)$$
$$r_1 \neq r_2 \Rightarrow Terminates(Walk(a, r_1, r_2), InRoom(a, r_1), t)$$

Now, a commonsense fact is that an agent cannot walk into one room at the same time that the agent walks into some other room. This can be represented using a state constraint that says that an agent can be in one room at a time:

$$\begin{aligned} &HoldsAt(InRoom(a, r_1), t) \wedge \\ &HoldsAt(InRoom(a, r_2), t) \Rightarrow \\ &r_1 = r_2 \end{aligned} \tag{8.1}$$

Given these axioms, we cannot, say, have both Nathan walking into the kitchen at timepoint 0:

$$Happens(Walk(Nathan, Hallway, Kitchen), 0)$$

and Nathan walking into the living room at timepoint 0:

$$Happens(Walk(Nathan, Hallway, LivingRoom), 0)$$

If we did, we would get inconsistency. We could show

$$\begin{aligned} &HoldsAt(InRoom(Nathan, Kitchen), 1) \wedge \\ &HoldsAt(InRoom(Nathan, LivingRoom), 1) \end{aligned}$$

which contradicts (8.1).

Similarly, the fact that two agents cannot simultaneously pick up the same object can be represented using a state constraint that says that an object can be held by one agent at a time:

$$\begin{aligned} &HoldsAt(Holding(a_1, o), t) \wedge HoldsAt(Holding(a_2, o), t) \Rightarrow \\ &a_1 = a_2 \end{aligned}$$

8.1.2 Event Occurrence Constraints

The second method of restricting concurrency is to use *event occurrence constraints*, or formulas involving two or more atoms of the form $Happens(\alpha, \tau)$, where α is an event term and τ is a timepoint term. We now describe several common idioms.

Totally Ordered Plans

When solving abduction or planning problems, we may wish to insist that the plan be a *totally ordered plan*, or simple sequence of event occurrences. We can do this with an event occurrence constraint that says that at most one event can occur at a time:

$$Happens(e_1, t) \wedge Happens(e_2, t) \Rightarrow e_1 = e_2$$

Different Conflicting Actions of Two Agents

It may be desirable to restrict concurrency for different but conflicting actions of two agents. Consider the following effect axioms dealing with moving a jar and putting a penny in a jar:

$$Initiates(Move(a, j, l_1, l_2), At(j, l_2), t)$$

$$l_1 \neq l_2 \Rightarrow Terminates(Move(a, j, l_1, l_2), At(j, l_1), t)$$

$$HoldsAt(At(p, l), t) \wedge HoldsAt(At(j, l), t) \Rightarrow$$
$$Initiates(PutIn(a, p, j, l), In(p, j), t)$$

We may wish to state that an agent cannot place a penny in a jar at the same time that another agent moves the jar, even though this is permitted by these axioms. We can do this by adding the axiom

$$Happens(Move(a_1, j, l_1, l_2), t) \wedge a_1 \neq a_2 \Rightarrow$$
$$\neg Happens(PutIn(a_2, p, j, l_1), t)$$

Similar Conflicting Actions of Two Agents

It is sometimes desirable to restrict concurrency for similar actions of two agents. Suppose we have an effect axiom that says that, if a piece of food is not eaten and an agent eats the piece of food, then the piece of food will be eaten:

$$\neg HoldsAt(Eaten(p), t) \Rightarrow$$
$$Initiates(Eat(a, p), Eaten(p), t)$$

Nothing will prevent two agents from simultaneously eating the same piece of food. To prevent this, we may add the axiom

$$Happens(Eat(a_1, p), t) \wedge a_1 \neq a_2 \Rightarrow \neg Happens(Eat(a_2, p), t)$$

Different Conflicting Actions of One Agent

We may wish to restrict concurrency for different but conflicting actions within a single agent. An agent may not be able to pat the agent's head and rub the agent's stomach at the same time:

$$Happens(PatHead(a), t) \Rightarrow \neg Happens(RubStomach(a), t)$$

Similar Conflicting Actions of One Agent

We may also wish to restrict concurrency for similar actions within a single agent. Even if an agent can be holding several objects at the same time, the agent may not be able to pick up two objects simultaneously:

$$Happens(PickUp(a, o_1), t) \wedge Happens(PickUp(a, o_2), t) \Rightarrow o_1 = o_2$$

8.1.3 Discussion

In order to recognize situations that might require restrictions on concurrency, we look for actions that use, consume, or provide the same resource at the same time.

Notice that EC and DEC disallow concurrent event occurrences with contradictory effects. If a fluent is simultaneously initiated and terminated by an occurring event, we get inconsistency. Consider the effect axioms

$$Initiates(Break(a, d), Broken(d), t)$$
$$Terminates(Fix(a, d), Broken(d), t)$$

and the event occurrence formulas

$$Happens(Break(Nathan, TV), 1)$$
$$Happens(Fix(Nathan, TV), 1)$$

From these we have both $HoldsAt(Broken(TV), 2)$ and $\neg HoldsAt(Broken(TV), 2)$.

8.2 Cumulative and Canceling Effects

When several events occur at the same time, the effects of those events might differ from what the effects would have been had the events occurred at different times. Specifically, two or more concurrent events can produce new, additional effects. In this case, we say that the effects of the events are cumulative. Further, one event can cancel the normal effects of another event.

We represent cumulative effects using modified effect axioms.

DEFINITION 8.1

If γ is a condition, α and $\alpha_1, \ldots, \alpha_n$ are event terms, β is a fluent term, and τ is a timepoint term, then

$$\gamma \wedge Happens(\alpha_1, \tau) \wedge \cdots \wedge Happens(\alpha_n, \tau) \Rightarrow Initiates(\alpha, \beta, \tau)$$

is a ***positive cumulative effect axiom***. This represents that, if γ is true and $\alpha_1, \ldots, \alpha_n$ all occur at τ, then β will be true after τ.

DEFINITION 8.2

If γ is a condition, α and $\alpha_1, \ldots, \alpha_n$ are event terms, β is a fluent term, and τ is a timepoint term, then

$$\gamma \wedge Happens(\alpha_1, \tau) \wedge \cdots \wedge Happens(\alpha_n, \tau) \Rightarrow Terminates(\alpha, \beta, \tau)$$

is a ***negative cumulative effect axiom***. This represents that if γ is true and $\alpha_1, \ldots, \alpha_n$ all occur at τ, then β will be false after τ.

We represent that one event cancels the effect of another event as follows. Suppose that a positive effect axiom represents that, if γ is true, then α_1 initiates a fluent β:

$$\gamma \Rightarrow Initiates(\alpha 1, \beta, \tau)$$

In order to represent that α_2 cancels the effect β of α_1, we rewrite this axiom as:

$$\gamma \wedge \neg Happens(\alpha_2, \tau) \Rightarrow Initiates(\alpha 1, \beta, \tau)$$

We treat negative effect axioms in a similar fashion.

8.2.1 Example: Camera with Flash

Consider a camera with a flash unit. If a camera's shutter is released at the same instant that its flash is triggered, then we have the cumulative effect that a properly exposed picture is taken:

$$Happens(TriggerFlash(c), t) \Rightarrow Initiates(ReleaseShutter(c), ProperlyExposedPicture(t), t) \quad (8.2)$$

If a camera's shutter is released and its flash is not triggered at the same instant, then an improperly exposed picture is taken:

$$\neg Happens(TriggerFlash(c), t) \Rightarrow Initiates(ReleaseShutter(c), ImproperlyExposedPicture(t), t) \quad (8.3)$$

The shutter is released at timepoints 0 and 1, but only at timepoint 1 is the flash triggered:

$$\neg ReleasedAt(f, t) \quad (8.4)$$

$$Happens(ReleaseShutter(Camera), 0) \quad (8.5)$$

$$Happens(TriggerFlash(Camera), 1) \quad (8.6)$$

$$Happens(ReleaseShutter(Camera), 1) \quad (8.7)$$

We can show that an improperly exposed picture is taken at timepoint 0 and that a properly exposed picture is taken at timepoint 1.

PROPOSITION 8.1

Let $\Sigma = (8.2) \wedge (8.3)$, $\Delta = (8.5) \wedge (8.6) \wedge (8.7)$, $\Omega = U[TriggerFlash, ReleaseShutter] \wedge U[ProperlyExposedPicture, ImproperlyExposedPicture]$, and $\Gamma = (8.4)$. Then we have

$$\begin{aligned} &CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ &\qquad \Omega \wedge \Gamma \wedge \mathrm{DEC} \\ &\models HoldsAt(ImproperlyExposedPicture(0), 1) \wedge \\ &\qquad HoldsAt(ProperlyExposedPicture(1), 2) \end{aligned}$$

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

$$\begin{aligned} &Initiates(e,f,t) \Leftrightarrow \\ &\exists c\,(e = ReleaseShutter(c) \wedge \\ &f = ProperlyExposedPicture(t) \wedge \\ &Happens(TriggerFlash(c), t)) \vee \\ &\exists c\,(e = ReleaseShutter(c) \wedge \\ &f = ImproperlyExposedPicture(t) \wedge \\ &\neg Happens(TriggerFlash(c), t)) \end{aligned} \tag{8.8}$$

$$\neg Terminates(e,f,t) \tag{8.9}$$

$$\neg Releases(e,f,t) \tag{8.10}$$

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

$$\begin{aligned} &Happens(e,t) \Leftrightarrow \\ &(e = ReleaseShutter(Camera) \wedge t = 0) \vee \\ &(e = TriggerFlash(Camera) \wedge t = 1) \vee \\ &(e = ReleaseShutter(Camera) \wedge t = 1) \end{aligned} \tag{8.11}$$

From (8.11), we have $\neg Happens(TriggerFlash(Camera), 0)$. From this, (8.5) (which follows from (8.11)), (8.3) (which follows from (8.8)), and DEC9, we have $HoldsAt(ImproperlyExposedPicture(0), 1)$. From (8.6) and (8.7) (which both follow from (8.11)), (8.2) (which follows from (8.8)), and DEC9, we have $HoldsAt(ProperlyExposedPicture(1), 2)$. ■

8.2.2 Example: Moving Robot

Another example is that of a simple robot. At any time, the robot has a unique location in two-dimensional space:

$$\begin{gathered} HoldsAt(Location(r, x_1, y_1), t) \wedge \\ HoldsAt(Location(r, x_2, y_2), t) \Rightarrow \\ x_1 = x_2 \wedge y_1 = y_2 \end{gathered} \tag{8.12}$$

And at any time, a robot has a unique direction given by an angle in radians measured counterclockwise from the x axis:

$$\begin{gathered} HoldsAt(Direction(r, d_1), t) \wedge \\ HoldsAt(Direction(r, d_2), t) \Rightarrow \\ d_1 = d_2 \end{gathered} \tag{8.13}$$

A robot has a left wheel and a right wheel. The wheels can be moved separately or at the same time. Moving the left wheel cancels the effect of moving the right wheel, and moving the right wheel cancels the effect of moving the left wheel. Thus, if a robot moves only its left wheel, then the robot will rotate clockwise by $\pi/180$ radians (1 degree):

$$\begin{gathered} \neg Happens(MoveRightWheel(r), t) \wedge \\ HoldsAt(Direction(r, d), t) \Rightarrow \\ Initiates(MoveLeftWheel(r), Direction(r, d - \pi/180), t) \end{gathered} \tag{8.14}$$

$$\begin{gathered} \neg Happens(MoveRightWheel(r), t) \wedge \\ HoldsAt(Direction(r, d), t) \Rightarrow \\ Terminates(MoveLeftWheel(r), Direction(r, d), t) \end{gathered} \tag{8.15}$$

If a robot moves its only right wheel, then the robot will rotate counterclockwise by $\pi/180$ radians:

$$\begin{gathered} \neg Happens(MoveLeftWheel(r), t) \wedge \\ HoldsAt(Direction(r, d), t) \Rightarrow \\ Initiates(MoveRightWheel(r), Direction(r, d + \pi/180), t) \end{gathered} \tag{8.16}$$

$$\begin{gathered} \neg Happens(MoveLeftWheel(r), t) \wedge \\ HoldsAt(Direction(r, d), t) \Rightarrow \\ Terminates(MoveRightWheel(r), Direction(r, d), t) \end{gathered} \tag{8.17}$$

Moving the left and right wheels simultaneously has a cumulative effect. Namely, if a robot moves its left and right wheels at the same time, then the robot will move forward by one unit:

$$\begin{aligned} &Happens(MoveLeftWheel(r), t) \wedge \\ &HoldsAt(Location(r, x, y), t) \wedge \\ &HoldsAt(Direction(r, d), t) \Rightarrow \\ &Initiates(MoveRightWheel(r), Location(r, x + Cos(d), y + Sin(d)), t) \end{aligned} \tag{8.18}$$

$$\begin{aligned} &Happens(MoveLeftWheel(r), t) \wedge \\ &HoldsAt(Location(r, x, y), t) \Rightarrow \\ &Terminates(MoveRightWheel(r), Location(r, x, y), t) \end{aligned} \tag{8.19}$$

Consider the following observations and narrative. The robot is initially at location $\langle 0, 0 \rangle$, and its direction is 0 (facing right):

$$\neg ReleasedAt(f, t) \tag{8.20}$$

$$HoldsAt(Location(Robot, 0, 0), 0) \tag{8.21}$$

$$HoldsAt(Direction(Robot, 0), 0) \tag{8.22}$$

At timepoint 0, the robot moves its right wheel; at timepoint 1, the robot moves both its left and right wheels:

$$Happens(MoveRightWheel(Robot), 0) \tag{8.23}$$

$$Happens(MoveLeftWheel(Robot), 1) \tag{8.24}$$

$$Happens(MoveRightWheel(Robot), 1) \tag{8.25}$$

We can show that at timepoint 2, the robot will be at location $\langle Cos(\pi/180), Sin(\pi/180) \rangle$.

PROPOSITION 8.2

Let $\Sigma = (8.14) \wedge (8.15) \wedge (8.16) \wedge (8.17) \wedge (8.18) \wedge (8.19)$, $\Delta = (8.23) \wedge (8.24) \wedge (8.25)$, $\Omega = U[MoveLeftWheel, MoveRightWheel] \wedge U[Direction, Location]$, $\Psi = (8.12) \wedge (8.13)$, and $\Gamma = (8.20) \wedge (8.21) \wedge (8.22)$. Then we have

$$\begin{aligned} &CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ &\Omega \wedge \Psi \wedge \Gamma \wedge \text{DEC} \\ &\models HoldsAt(Location(Robot, Cos(\pi/180), Sin(\pi/180)), 2) \end{aligned}$$

Proof From CIRC[Σ; *Initiates, Terminates, Releases*], Theorem 2.2, and Theorem 2.1, we have

$$
\begin{aligned}
&\mathit{Initiates}(e,f,t) \Leftrightarrow \\
&\exists r,d\,(e = \mathit{MoveLeftWheel}(r) \wedge \\
&f = \mathit{Direction}(r, d - \pi/180) \wedge \\
&\neg\mathit{Happens}(\mathit{MoveRightWheel}(r), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Direction}(r,d), t)) \vee \\
&\exists r,d\,(e = \mathit{MoveRightWheel}(r) \wedge \\
&f = \mathit{Direction}(r, d + \pi/180) \wedge \\
&\neg\mathit{Happens}(\mathit{MoveLeftWheel}(r), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Direction}(r,d), t)) \vee \\
&\exists r,x,y,d\,(e = \mathit{MoveRightWheel}(r) \wedge \\
&f = \mathit{Location}(r, x + \mathit{Cos}(d), y + \mathit{Sin}(d)) \wedge \\
&\mathit{Happens}(\mathit{MoveLeftWheel}(r), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Location}(r,x,y), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Direction}(r,d), t))
\end{aligned}
\tag{8.26}
$$

$$
\begin{aligned}
&\mathit{Terminates}(e,f,t) \Leftrightarrow \\
&\exists r,d\,(e = \mathit{MoveLeftWheel}(r) \wedge \\
&f = \mathit{Direction}(r,d) \wedge \\
&\neg\mathit{Happens}(\mathit{MoveRightWheel}(r), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Direction}(r,d), t)) \vee \\
&\exists r,d\,(e = \mathit{MoveRightWheel}(r) \wedge \\
&f = \mathit{Direction}(r,d) \wedge \\
&\neg\mathit{Happens}(\mathit{MoveLeftWheel}(r), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Direction}(r,d), t)) \vee \\
&\exists r,x,y,d\,(e = \mathit{MoveRightWheel}(r) \wedge \\
&f = \mathit{Location}(r,x,y) \wedge \\
&\mathit{Happens}(\mathit{MoveLeftWheel}(r), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Location}(r,x,y), t))
\end{aligned}
\tag{8.27}
$$

$$
\neg\mathit{Releases}(e,f,t) \tag{8.28}
$$

From CIRC[Δ; *Happens*] and Theorem 2.1, we have

$$
\begin{aligned}
&\mathit{Happens}(e,t) \Leftrightarrow \\
&(e = \mathit{MoveRightWheel}(\mathit{Robot}) \wedge t = 0) \vee
\end{aligned}
\tag{8.29}
$$

$$(e = MoveLeftWheel(Robot) \wedge t = 1) \vee$$
$$(e = MoveRightWheel(Robot) \wedge t = 1)$$

From (8.29), we have $\neg Happens(MoveLeftWheel(Robot), 0)$. From this, (8.23) (which follows from (8.29)), (8.22), (8.16) (which follows from (8.26)), and DEC9, we have

$$HoldsAt(Direction(Robot, \pi/180), 1) \tag{8.30}$$

From (8.29) and (8.27), we have $\neg\exists e\,(Happens(e, 0) \wedge Terminates(e, Location(Robot, 0, 0), 0))$. From this, (8.21), (8.20), and DEC5, we have $HoldsAt(Location(Robot, 0, 0), 1)$. From this, (8.24) and (8.25) (which both follow from (8.29)), (8.30), (8.18) (which follows from (8.26)), and DEC9, we have $HoldsAt(Location(Robot, Cos(\pi/180), Sin(\pi/180)), 2)$. ■

Bibliographic Notes

Our discussion of restricting concurrency draws on the notions of "exclusive and shared resources" of Pinto (1994, p. 59), the "complete exclusion" and "conflict exclusion" of Ernst et al. (1997, p. 1171), and the "interference" and "competing needs" of Blum and Furst (1997, p. 286). The technique of preventing more than one event from occurring at a time is used in the event calculus by Shanahan and Witkowski (2004). The example of putting a penny in a jar is based on the example of loading cargo into a rocket of Veloso (1989).

Concurrent events were originally considered in the situation calculus. Schubert (1990, pp. 56–61) introduces a function $Costart(a_1, a_2)$ that maps two actions a_1 and a_2 that start at the same time into a new action that is composed of a_1 and a_2 and that stops when a_1 and a_2 have both stopped. He gives the example of two people lifting a sofa (p. 56), but does not treat cumulative and canceling effects. Gelfond et al. (1991) do treat cumulative and canceling effects. They describe the soup bowl scenario (p. 174), based on Pednault's (1987) scenario of two people lifting a table from opposite sides: If a soup bowl is lifted with one hand, the soup spills. If the bowl is lifted with both hands, the soup does not spill. Gelfond et al. use the predicate $Cancels(a, f)$ to represent that an action a (such as *Lift*) cancels the effect of a on a fluent f (such as *Spilled*). Lin and Shoham (1992, 1995) introduce "global actions" and "primitive actions" (Lin and Shoham, 1995, p. 304). A global action consists of a set of primitive actions. The predicate $Canceled(g_1, g_2, s)$ represents that a primitive action in g_2 cancels the global action g_1.

Pinto (1994, pp. 57–66) introduces a function + that maps two actions a_1 and a_2 into the action of performing a_1 and a_2 simultaneously. He discusses the "precondition interaction problem," which is that, in a given situation it may be

possible to perform a_1 and it may be possible to perform a_2, yet it may not be possible to perform $a_1 + a_2$. He uses the example of an agent that can paint wall A or paint wall B but cannot paint both walls A and B simultaneously. He proposes a predicate $precInt(a_1, a_2)$, which represents that the preconditions of a_1 interact with the preconditions of a_2, and the axiom (p. 59):

$$Poss(a_1 + a_2, s) \equiv Poss(a_1, s) \wedge Poss(a_2, s) \wedge \neg precInt(a_1, a_2)$$

The predicate $Poss(a, s)$ represents that it is possible to perform action a in situation s. Pinto proposes that $precInt(a_1, a_2)$ could be determined on the basis of which resources are required to perform a_1 and a_2 and whether the use of those resources must be exclusive or can be shared. For example, a brush, wall, paint canister, and light are required to paint a wall. The use of the brush, wall, and paint canister must be exclusive, but the use of the light can be shared.

Pinto (1994) also discusses the "effect interaction problem," which is that concurrent actions may have cumulative and canceling effects, called "synergy" and "cancellation" (p. 60). He proposes to deal with this problem by breaking effects into "primary" and "secondary" effects (p. 61). Primary effects are the effects that must always occur when an action is performed; secondary effects are the effects that are subject to synergy and cancellation. He represents primary effects using effect axioms and secondary effects using state constraints. He treats the soup bowl scenario by using state constraints that determine when the bowl of soup is lifted or spilled. The analog in the event calculus consists of two effect axioms and two state constraints:

$$Initiates(LiftLeft, LiftedLeft, t)$$
$$Initiates(LiftRight, LiftedRight, t)$$
$$HoldsAt(Lifted, t) \Leftrightarrow HoldsAt(LiftedLeft, t) \wedge HoldsAt(LiftedRight, t)$$
$$HoldsAt(Spilled, t) \Leftrightarrow HoldsAt(LiftedLeft, t) \veebar HoldsAt(LiftedRight, t)$$

Reiter (1996) presents a variation on Pinto's (1994) approach. Reiter adds a sort called "concurrent" (p. 3) for sets of actions, redefines *do* as a function from sets of actions and situations to situations, and discusses successor state axioms that take concurrency into account. See also the review of Reiter (2001, chap. 7). Baral and Gelfond (1997) extend action language $\mathcal{A}$ for concurrent actions.

Following the work in the situation calculus, Shanahan (1995a, pp. 265–268; 1997b, pp. 301–304) introduces a method for handling concurrent events with cumulative and canceling effects into the event calculus. The notation $e_1 \& e_2$ (also $e_1 + e_2$) represents a compound event that consists of e_1 and e_2. The predicate $Cancels(e_1, e_2)$ represents that, if e_1 occurs concurrently with e_2, then e_1 cancels the effects of e_2. Shanahan extends the circumscriptive event calculus for concurrent events and shows that it can handle a problem involving a shopping cart. If the cart is pushed, then it moves forward; if it is pulled, then it moves backward. But if it is pushed and pulled at the same time, then it spins around. Shanahan (1999a,

pp. 424–426) incorporates concurrent events into the forced separation version of the event calculus. In this case, the predicate is $Cancels(e_1, e_2, f)$, which represents that e_1 cancels the effect of e_2 on fluent f. Finally, R. Miller and Shanahan (1999, pp. 7–8; 2002, pp. 460–461) show that concurrent events with cumulative and canceling effects can be represented using effect axioms with *Happens* preconditions, which we call cumulative effect axioms. R. Miller and Shanahan (1999, p. 8; 2002, pp. 460–461) show how to handle the soup bowl scenario using their approach. Shanahan (1996) uses the example of a robot but not to demonstrate concurrent events with cumulative and canceling effects. R. Miller and Shanahan (1996) treat the case of a robot that can vary its bearing continuously as it moves.

Exercises

8.1 Write an algorithm that takes an axiomatization (list of axioms) as input and produces suggestions for possible event occurrence constraints as output. The idea is that these constraints are examined by a person, who decides whether they should be added to the axiomatization. The algorithm should work by looking for events that initiate or terminate fluents that are the preconditions of other events. Specifically, whenever it is the case that

- $Initiates(\alpha_1, \beta, \tau)$, and $\neg HoldsAt(\beta, \tau)$ is an action or fluent precondition of α_2, or
- $Terminates(\alpha_1, \beta, \tau)$, and $HoldsAt(\beta, \tau)$ is an action or fluent precondition of α_2,

(where α_1 and α_2 are event terms, β is a fluent term, and τ is a timepoint term); the algorithm should produce as output an axiom that states that α_1 and α_2 cannot occur simultaneously.

8.2 Extend the algorithm in Exercise 1 to suggest event occurrence constraints for similar actions of one or two agents.

8.3 Formalize the mixing of two colors of paint together as follows. If yellow paint is placed on a palette, the result is yellow. If blue paint is placed on a palette, the result is blue. But if yellow and blue paint are placed on a palette, the result is green.

8.4 Formalize the ripping of a piece of paper as follows. If the paper is pulled left, it moves left. If it is pulled right, it moves right. But if it is simultaneously pulled left and right, it rips. Discuss what a more detailed axiomatization of ripping a paper would look like.

8.5 Formalize the clapping of hands.

CHAPTER 9

Nondeterministic Effects of Events

An event has *nondeterministic effects* if the event can have two or more alternative effects. For example, flipping a coin has two alternative effects: the coin may land with heads showing, or it may land with tails showing. We represent nondeterministic effects of events in the event calculus by allowing event occurrences to give rise to several classes of models. In this chapter, we describe two ways of representing nondeterministic effects in the event calculus: determining fluents and disjunctive event axioms. These techniques enable the representation of uncertainty about the effects of events.

9.1 Determining Fluents

One way of representing nondeterministic effects is to use determining fluents. A *determining fluent* is a fluent that (1) is released from the commonsense law of inertia so that its truth value is permitted to vary arbitrarily from timepoint to timepoint and (2) is used in the condition of one or more effect axioms in order to determine the effects of events on other fluents.

Suppose that DF is a fluent that is released from the commonsense law of inertia:

$$ReleasedAt(DF, t)$$

Then, this fluent may be used to determine whether the effect of E is $F1$:

$$HoldsAt(DF, t) \Rightarrow Initiates(E, F1, t)$$

or whether the effect of E is $F2$:

$$\neg HoldsAt(DF, \tau) \Rightarrow Initiates(E, F2, t)$$

9.1.1 Example: Roulette Wheel

We consider the example of spinning a roulette wheel that can land on the numbers 1, 2, or 3. We use a determining fluent *WheelNumberDeterminer*. We make the wheel an argument of this fluent so that at any given timepoint different wheels may land on different numbers. This fluent is released from the commonsense law of inertia:

$$ReleasedAt(WheelNumberDeterminer(w, n), t) \tag{9.1}$$

For any given wheel, this fluent specifies one of the numbers 1, 2, or 3 at any given timepoint:

$$HoldsAt(WheelNumberDeterminer(w, n), t) \Rightarrow \\ n = 1 \vee n = 2 \vee n = 3 \tag{9.2}$$

And for any given wheel, this fluent specifies a unique number at any given timepoint:

$$HoldsAt(WheelNumberDeterminer(w, n_1), t) \wedge \\ HoldsAt(WheelNumberDeterminer(w, n_2), t) \Rightarrow \\ n_1 = n_2 \tag{9.3}$$

We have several effect axioms. If the determining fluent specifies a particular number for a wheel and a dealer spins the wheel, then the wheel will be on that number:

$$HoldsAt(WheelNumberDeterminer(w, n), t) \Rightarrow \\ Initiates(Spin(d, w), WheelNumber(w, n), t) \tag{9.4}$$

If a wheel is on one number, the determining fluent specifies another number for the wheel, the first number is different from the second number, and a dealer spins the wheel, then the wheel will no longer be on the first number:

$$HoldsAt(WheelNumber(w, n_1), t) \wedge \\ HoldsAt(WheelNumberDeterminer(w, n_2), t) \wedge \\ n_1 \neq n_2 \Rightarrow \\ Terminates(Spin(d, w), WheelNumber(w, n_1), t) \tag{9.5}$$

If a dealer resets a wheel, then the wheel is no longer on any number:

$$Terminates(Reset(d, w), WheelNumber(w, n), t) \tag{9.6}$$

We have a state constraint that says that the wheel is on at most one number at a time:

$$\begin{aligned} &HoldsAt(WheelNumber(w, n_1), t) \wedge \\ &HoldsAt(WheelNumber(w, n_2), t) \Rightarrow \\ &n_1 = n_2 \end{aligned} \tag{9.7}$$

Suppose that a wheel that is not on any number is spun:

$$\neg HoldsAt(WheelNumber(Wheel, n), 0) \tag{9.8}$$

$$\neg ReleasedAt(WheelNumber(Wheel, n), t) \tag{9.9}$$

$$Happens(Spin(Dealer, Wheel), 0) \tag{9.10}$$

We can show that the wheel will be on one of the numbers 1, 2, or 3 and that, if it lands on a given number, it will still be on that number one timepoint later.

PROPOSITION 9.1 Let $\Sigma = (9.4) \wedge (9.5) \wedge (9.6)$, $\Delta = (9.10)$, $\Omega = U[Spin, Reset] \wedge U[WheelNumberDeterminer, WheelNumber]$, $\Psi = (9.2) \wedge (9.3) \wedge (9.7)$, and $\Gamma = (9.1) \wedge (9.8) \wedge (9.9)$. Then we have

$$\begin{aligned} &CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ &\Omega \wedge \Psi \wedge \Gamma \wedge \mathrm{EC} \models (HoldsAt(WheelNumber(Wheel, 1), 1) \vee \\ &\quad HoldsAt(WheelNumber(Wheel, 2), 1) \vee \\ &\quad HoldsAt(WheelNumber(Wheel, 3), 1)) \wedge \\ &\quad (HoldsAt(WheelNumber(Wheel, k), 1) \Rightarrow \\ &\quad HoldsAt(WheelNumber(Wheel, k), 2)) \end{aligned}$$

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

$$\begin{aligned} &Initiates(e, f, t) \Leftrightarrow \\ &\exists d, w, n\, (e = Spin(d, w) \wedge \\ &f = WheelNumber(w, n) \wedge \\ &HoldsAt(WheelNumberDeterminer(w, n), t)) \end{aligned} \tag{9.11}$$

$$\begin{aligned} &Terminates(e, f, t) \Leftrightarrow \\ &\exists d, w, n_1, n_2\, (e = Spin(d, w) \wedge \\ &f = WheelNumber(w, n_1) \wedge \\ &HoldsAt(WheelNumber(w, n_1), t) \wedge \\ &HoldsAt(WheelNumberDeterminer(w, n_2), t) \wedge \end{aligned} \tag{9.12}$$

$$n_1 \neq n_2) \vee$$
$$\exists d, w, n\,(e = Reset(d, w) \wedge f = WheelNumber(w, n))$$

$$\neg Releases(e, f, t) \tag{9.13}$$

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

$$Happens(e, t) \Leftrightarrow (e = Spin(Dealer, Wheel) \wedge t = 0) \tag{9.14}$$

First, we show

$$HoldsAt(WheelNumber(Wheel, 1), 1) \vee$$
$$HoldsAt(WheelNumber(Wheel, 2), 1) \vee$$
$$HoldsAt(WheelNumber(Wheel, 3), 1)$$

We consider three cases, which, from (9.2) and (9.3), are exhaustive.

Case 1: $HoldsAt(WheelNumberDeterminer(Wheel, 1), 0)$.
From this, (9.10) (which follows from (9.14)), (9.4) (which follows from (9.11)), $0 < 1$, $\neg StoppedIn(0, WheelNumber(Wheel, 1), 1)$ (which follows from (9.14) and EC3), $\neg ReleasedIn(0, WheelNumber(Wheel, 1), 1)$ (which follows from (9.14) and EC13), and EC14, we have $HoldsAt(WheelNumber(Wheel, 1), 1)$.

Case 2: $HoldsAt(WheelNumberDeterminer(Wheel, 2), 0)$.
By a similar argument, we have $HoldsAt(WheelNumber(Wheel, 2), 1)$.

Case 3: $HoldsAt(WheelNumberDeterminer(Wheel, 3), 0)$.
By a similar argument, we have $HoldsAt(WheelNumber(Wheel, 3), 1)$.

Second, we show

$$HoldsAt(WheelNumber(Wheel, k), 1) \Rightarrow HoldsAt(WheelNumber(Wheel, k), 2)$$

Let κ be an arbitrary integer. We must show

$$HoldsAt(WheelNumber(Wheel, \kappa), 1) \Rightarrow HoldsAt(WheelNumber(Wheel, \kappa), 2)$$

Suppose $HoldsAt(WheelNumber(Wheel, \kappa), 1)$. From this, $1 < 2$, $PersistsBetween(1, WheelNumber(Wheel, \kappa), 2)$ (which follows from (9.9) and EC7), $\neg Clipped(1, WheelNumber(Wheel, \kappa), 2)$ (which follows from (9.14) and EC1), and EC9, we have $HoldsAt(WheelNumber(Wheel, \kappa), 2)$, as required. ■

9.2 Disjunctive Event Axioms

Another way of representing nondeterministic effects is to use disjunctive event axioms.

DEFINITION 9.1

If α and $\alpha_1, \ldots, \alpha_n$ are event terms and τ is a timepoint term, then

$$Happens(\alpha, \tau) \Rightarrow$$
$$Happens(\alpha_1, \tau) \vee \cdots \vee Happens(\alpha_n, \tau)$$

is a ***disjunctive event axiom***.

9.2.1 Example: Running and Driving

We start with a disjunctive event axiom that says that, if an agent goes to a location, then the agent runs or drives to that location:

$$Happens(Go(a, l), t) \Rightarrow \quad (9.15)$$
$$Happens(Run(a, l), t) \vee$$
$$Happens(Drive(a, l), t)$$

We have an effect axiom that says that, if an agent runs to a location, then the agent will be tired:

$$Initiates(Run(a, l), Tired(a), t) \quad (9.16)$$

Suppose we are told that Nathan was initially not tired and went to the bookstore:

$$\neg ReleasedAt(f, t) \quad (9.17)$$
$$\neg HoldsAt(Tired(Nathan), 0) \quad (9.18)$$
$$Happens(Go(Nathan, Bookstore), 0) \quad (9.19)$$

The event $Go(Nathan, Bookstore)$ has a nondeterministic effect: Nathan may or may not be tired afterward. Now suppose we are told that Nathan was tired afterward:

$$HoldsAt(Tired(Nathan), 1) \quad (9.20)$$

We can then show that Nathan must have run to the bookstore.

We cannot use the methods given in Section 2.6.1 to compute

$$CIRC[(9.15) \wedge (9.19); Happens] \tag{9.21}$$

Instead we replace (9.21) with

$$\begin{aligned}
&(Happens(e,t) \Leftrightarrow \\
&(e = Go(Nathan, Bookstore) \wedge t = 0) \wedge \\
&(e = Run(Nathan, Bookstore) \wedge t = 0)) \vee \\
&(Happens(e,t) \Leftrightarrow \\
&(e = Go(Nathan, Bookstore) \wedge t = 0) \wedge \\
&(e = Drive(Nathan, Bookstore) \wedge t = 0))
\end{aligned} \tag{9.22}$$

PROPOSITION 9.2 Let $\Sigma = (9.16)$, $\Delta = (9.22)$, $\Omega = U[Go, Run, Drive]$, and $\Gamma = (9.17) \wedge (9.18) \wedge (9.20)$. Then we have

$$\begin{aligned}
&CIRC[\Sigma; Initiates, Terminates, Releases] \wedge \Delta \wedge \Omega \wedge \Gamma \wedge \text{EC} \\
&\models Happens(Run(Nathan, Bookstore), 0)
\end{aligned}$$

Proof Suppose, to the contrary, that

$$\neg Happens(Run(Nathan, Bookstore), 0) \tag{9.23}$$

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

$$Initiates(e,f,t) \Leftrightarrow \exists a, l\,(e = Run(a,l) \wedge Tired(a)) \tag{9.24}$$

$$\neg Terminates(e,f,t) \tag{9.25}$$

$$\neg Releases(e,f,t) \tag{9.26}$$

From (9.17) and EC7, we have

$$PersistsBetween(0, Tired(Nathan), 1) \tag{9.27}$$

From (9.24), (9.22), (9.23), and EC2, we have $\neg Declipped(0, Tired(Nathan), 1))$. From this, (9.18), $0 < 1$, (9.27), and EC10, we have $\neg HoldsAt(Tired(Nathan), 1)$, which contradicts (9.20). Therefore, we have $Happens(Run(Nathan, Bookstore), 0)$, as required. ■

Bibliographic Notes

The method of determining fluents was introduced by Shanahan (1997b, pp. 294–297, 301; 1999a, pp. 419–420). The roulette wheel example is similar to other examples of nondeterminism such as flipping a coin (Kartha, 1994, pp. 386–387; Shanahan, 1999a, p. 419) and Raymond Reiter's scenario of throwing an object onto a chessboard (Kartha and Lifschitz, 1994, p. 344; Baral, 1995, p. 2018; Shanahan 1995a, pp. 264–265; 1997b, pp. 290–298; 1999a, pp. 419–420; Elkan, 1996, p. 63; Lin 1996, p. 671). Disjunctive event axioms were introduced by Shanahan (1997b, pp. 297–298, 342–345, 359–361). Our formalization of the example of running and driving using disjunctive event axioms is based on Shanahan's (1997b, pp. 359–361) formalization of Kartha's (1994, pp. 382–383) bus ride scenario.

Exercises

9.1 Formalize that a lightbulb burns out at a random time.

9.2 Formalize the operation of buttons and lights used in game shows. There are two contestants. Each contestant has a button and a light. The light goes on for the first contestant to push the button. If both contestants press their button at the same instant, the tie is broken at random and only one light goes on.

9.3 Extend the formalization in Exercise 1 for n contestants.

9.4 Formalize that a balloon may or may not pop when it comes into contact with a pin.

9.5 List three everyday scenarios involving nondeterminism, and sketch out event calculus representations of them.

CHAPTER 10

Space

Many instances of commonsense reasoning involve space. In this chapter, we present two event calculus axiomatizations of space: relational space and metric space. We describe how these axiomatizations can be used to solve some sample problems. A closely related issue that we consider is object identity: Two objects observed at different times and locations in space may or may not be the same object.

10.1 Relational Space

In the commonsense world, objects stand in various spatial relations to other objects. For example, a pencil is in a jar, a person is in a room, a person is holding a book, a glass is on a table, or a person is wearing a shirt. A *relational space* consists of a set of objects and a binary relation on the set. In this section we present a representation of a relational space in the event calculus. (Other representations of relational space are discussed in Sections 6.1.1, 6.3.1, and 6.4.1.)

10.1.1 Basic Representation

We start with a basic representation of relational space, which serves as a prototype for more elaborate representations. The representation consists of:

- an object sort, with variables o, o_1, o_2, ...
- an agent sort, which is a subsort of the object sort, with variables a, a_1, a_2, ...

- a fluent $IN(o_1, o_2)$, which represents an abstract spatial relation between object o_1 and object o_2
- an event $MOVE(a, o_1, o_2, o_3)$, which represents the action of agent a adding the abstract spatial relation between object o_1 and object o_3, and dropping the abstract spatial relation between object o_1 and object o_2

IN represents an irreflexive, antisymmetric, intransitive, and functional relation.

AXIOM RS1 $\neg HoldsAt(IN(o, o), t)$

AXIOM RS2 $HoldsAt(IN(o_1, o_2), t) \Rightarrow \neg HoldsAt(IN(o_2, o_1), t)$

AXIOM RS3 $HoldsAt(IN(o_1, o_2), t) \wedge HoldsAt(IN(o_2, o_3), t) \Rightarrow \neg HoldsAt(IN(o_1, o_3), t)$

AXIOM RS4 $HoldsAt(IN(o, o_1), t) \wedge HoldsAt(IN(o, o_2), t) \Rightarrow o_1 = o_2$

MOVE initiates and terminates *IN*.

AXIOM RS5 $Initiates(MOVE(a, o_1, o_2, o_3), IN(o_1, o_3), t)$

AXIOM RS6 $Terminates(MOVE(a, o_1, o_2, o_3), IN(o_1, o_2), t)$

10.1.2 Extended Representation

We now extend the basic representation to deal with agents and physical objects within rooms. We add the following subsorts of the object sort:

- a physical object sort, with variables p, p_1, p_2, ...
- a room sort, with variables r, r_1, r_2, ...

As shown in Table 10.1, we use *IN* to represent different spatial relationships between objects, depending on the sorts of the arguments of *IN*, and *MOVE* to represent different spatial actions, depending on the sorts of the arguments to *MOVE*. For example, $IN(p_1, p_2)$ represents that physical object p_1 is in or inside physical object p_2, whereas $IN(p, a)$ represents that agent a is holding physical object p.

The fluent $IN(o, r)$ represents that object o is directly in room r. We introduce a fluent $INROOM(o, r)$, which represents that object o is indirectly in room r.

Table 10.1 Meaning of *IN* and *MOVE*[a]

Fluent or event	*Meaning*
$IN(p_1, p_2)$	p_1 is in p_2
$IN(a, r)$	a is in r
$IN(p, r)$	p is in r
$IN(p, a)$	a is holding p
$MOVE(a, p_1, r, p_2)$	a (in r) puts p_1 (in r) into p_2
$MOVE(a, p_1, p_2, r)$	a (in r) removes p_1 from p_2 into r
$MOVE(a, a, r_1, r_2)$	a goes from r_1 to r_2
$MOVE(a, p, r, a)$	a (in r) picks up p
$MOVE(a, p, a, r)$	a (in r) sets down p into r

[a] a is an agent; p, p_1, and p_2 are physical objects; r, r_1, and r_2 are rooms.

AXIOM RS7
$$HoldsAt(IN(o, r), t) \Rightarrow HoldsAt(INROOM(o, r), t)$$

AXIOM RS8
$$HoldsAt(IN(o_1, o_2), t) \wedge HoldsAt(INROOM(o_2, r), t) \Rightarrow HoldsAt(INROOM(o_1, r), t)$$

An object can be indirectly in at most one room at a time.

AXIOM RS9
$$HoldsAt(INROOM(o, r_1), t) \wedge HoldsAt(INROOM(o, r_2), t) \Rightarrow r_1 = r_2$$

We then replace the general RS5 and RS6 with the more specific RS10 through RS19, which follow.

In order for an agent to put physical object p_1 into physical object p_2, the agent and p_1 must be directly in the same room and p_2 must be indirectly in that room.

AXIOM RS10
$$HoldsAt(IN(a, r), t) \wedge$$
$$HoldsAt(IN(p_1, r), t) \wedge$$
$$HoldsAt(INROOM(p_2, r), t) \Rightarrow$$
$$Initiates(MOVE(a, p_1, r, p_2), IN(p_1, p_2), t)$$

AXIOM RS11
$$HoldsAt(IN(a, r), t) \wedge$$
$$HoldsAt(IN(p_1, r), t) \wedge$$
$$HoldsAt(INROOM(p_2, r), t) \Rightarrow$$
$$Terminates(MOVE(a, p_1, r, p_2), IN(p_1, r), t)$$

In order for an agent to remove physical object p_1 from physical object p_2 and put p_1 in a room, the agent must be directly in that room.

AXIOM RS12
$$HoldsAt(IN(a, r), t) \Rightarrow Initiates(MOVE(a, p_1, p_2, r), IN(p_1, r), t)$$

AXIOM RS13
$$HoldsAt(IN(a, r), t) \Rightarrow Terminates(MOVE(a, p_1, p_2, r), IN(p_1, p_2), t)$$

In order for an agent to go from room r_1 to room r_2, the agent must be directly in r_1.

AXIOM RS14
$$HoldsAt(IN(a, r_1), t) \Rightarrow Initiates(MOVE(a, a, r_1, r_2), IN(a, r_2), t)$$

AXIOM RS15
$$HoldsAt(IN(a, r_1), t) \Rightarrow Terminates(MOVE(a, a, r_1, r_2), IN(a, r_1), t)$$

In order for an agent to pick up a physical object, the agent and the physical object must be directly in the same room.

AXIOM RS16
$$HoldsAt(IN(a, r), t) \wedge HoldsAt(IN(p, r), t) \Rightarrow Initiates(MOVE(a, p, r, a), IN(p, a), t)$$

AXIOM RS17
$$HoldsAt(IN(a, r), t) \wedge HoldsAt(IN(p, r), t) \Rightarrow Terminates(MOVE(a, p, r, a), IN(p, r), t)$$

In order for an agent to set down a physical object into a room, the agent must be holding the physical object, and the agent must be directly in the room.

AXIOM RS18
$$HoldsAt(IN(p, a), t) \wedge HoldsAt(IN(a, r), t) \Rightarrow Initiates(MOVE(a, p, a, r), IN(p, r), t)$$

AXIOM RS19
$$HoldsAt(IN(p, a), t) \wedge HoldsAt(IN(a, r), t) \Rightarrow Terminates(MOVE(a, p, a, r), IN(p, a), t)$$

10.1.3 Example: Moving a Newspaper and a Box

Lisa, a newspaper, and a box are in the living room. She puts the newspaper in the box, picks up the box, and walks into the kitchen:

$$HoldsAt(IN(Lisa, LivingRoom), 0) \tag{10.1}$$
$$HoldsAt(IN(Newspaper, LivingRoom), 0) \tag{10.2}$$
$$HoldsAt(IN(Box, LivingRoom), 0) \tag{10.3}$$
$$Happens(MOVE(Lisa, Newspaper, LivingRoom, Box), 0) \tag{10.4}$$
$$Happens(MOVE(Lisa, Box, LivingRoom, Lisa), 1) \tag{10.5}$$
$$Happens(MOVE(Lisa, Lisa, LivingRoom, Kitchen), 2) \tag{10.6}$$

Lisa then sets down the box and walks back into the living room:

$$Happens(MOVE(Lisa, Box, Lisa, Kitchen), 3) \tag{10.7}$$
$$Happens(MOVE(Lisa, Lisa, Kitchen, LivingRoom), 4) \tag{10.8}$$

The fluent *INROOM* is always released from the commonsense law of inertia, and *IN* is never released from this law:

$$ReleasedAt(INROOM(o_1, o_2), t) \tag{10.9}$$
$$\neg ReleasedAt(IN(o_1, o_2), t) \tag{10.10}$$

We can show that Lisa will be in the living room, but the newspaper and box will be in the kitchen.

PROPOSITION 10.1 Let Σ be the conjunction of RS10 through RS19. Let Δ be the conjunction of (10.4), (10.5), (10.6), (10.7), and (10.8). Let Ω be $U[IN, INROOM]$. Let Ψ be the conjunction of RS1 through RS4, and RS7 through RS9. Let Γ be the conjunction of (10.1), (10.2), (10.3), (10.9), and (10.10). Then we have

$$\begin{aligned} CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ \Omega \wedge \Psi \wedge \Gamma \wedge \mathrm{EC} \\ \models HoldsAt(IN(Lisa, LivingRoom), 5) \wedge \\ HoldsAt(IN(Box, Kitchen), 5) \wedge \\ HoldsAt(INROOM(Newspaper, Kitchen), 5) \end{aligned}$$

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

$$\begin{aligned} &Initiates(e, f, t) \Leftrightarrow \qquad (10.11) \\ &\exists a, p_1, p_2, r \ (e = MOVE(a, p_1, r, p_2) \ \wedge \end{aligned}$$

$$
\begin{aligned}
& f = IN(p_1, p_2) \wedge \\
& HoldsAt(IN(a, r), t) \wedge \\
& HoldsAt(IN(p_1, r), t) \wedge \\
& HoldsAt(INROOM(p_2, r), t)) \vee \\
& \exists a, p_1, p_2, r\,(e = MOVE(a, p_1, p_2, r) \wedge \\
& f = IN(p_1, r), \wedge \\
& HoldsAt(IN(a, r), t)) \vee \\
& \exists a, r_1, r_2\,(e = MOVE(a, a, r_1, r_2) \wedge \\
& f = IN(a, r_2) \wedge \\
& HoldsAt(IN(a, r_1), t)) \vee \\
& \exists a, p, r\,(e = MOVE(a, p, r, a) \wedge \\
& f = IN(p, a) \wedge \\
& HoldsAt(IN(a, r), t) \wedge \\
& HoldsAt(IN(p, r), t)) \vee \\
& \exists a, p, r\,(e = MOVE(a, p, a, r) \wedge \\
& f = IN(p, r) \wedge \\
& HoldsAt(IN(p, a), t) \wedge \\
& HoldsAt(IN(a, r), t))
\end{aligned}
$$

$$
\begin{aligned}
& Terminates(e, f, t) \Leftrightarrow \qquad (10.12) \\
& \exists a, p_1, p_2, r\,(e = MOVE(a, p_1, r, p_2) \wedge \\
& f = IN(p_1, r) \wedge \\
& HoldsAt(IN(a, r), t) \wedge \\
& HoldsAt(IN(p_1, r), t) \wedge \\
& HoldsAt(INROOM(p_2, r), t)) \vee \\
& \exists a, p_1, p_2, r\,(e = MOVE(a, p_1, p_2, r) \wedge \\
& f = IN(p_1, p_2) \wedge \\
& HoldsAt(IN(a, r), t)) \vee \\
& \exists a, r_1, r_2\,(e = MOVE(a, a, r_1, r_2) \wedge \\
& f = IN(a, r_1) \wedge \\
& HoldsAt(IN(a, r_1), t)) \vee \\
& \exists a, p, r\,(e = MOVE(a, p, r, a) \wedge \\
& f = IN(p, r) \wedge \\
& HoldsAt(IN(a, r), t) \wedge \\
& HoldsAt(IN(p, r), t)) \vee
\end{aligned}
$$

$$\exists a, p, r\,(e = MOVE(a, p, a, r) \wedge$$
$$f = IN(p, a) \wedge$$
$$HoldsAt(IN(p, a), t) \wedge$$
$$HoldsAt(IN(a, r), t))$$

$$\neg Releases(e, f, t) \tag{10.13}$$

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

$$Happens(e, t) \Leftrightarrow \tag{10.14}$$
$$(e = MOVE(Lisa, Newspaper, LivingRoom, Box) \wedge t = 0) \vee$$
$$(e = MOVE(Lisa, Box, LivingRoom, Lisa) \wedge t = 1) \vee$$
$$(e = MOVE(Lisa, Lisa, LivingRoom, Kitchen) \wedge t = 2) \vee$$
$$(e = MOVE(Lisa, Box, Lisa, Kitchen) \wedge t = 3) \vee$$
$$(e = MOVE(Lisa, Lisa, Kitchen, LivingRoom) \wedge t = 4)$$

From (10.1), $0 < 2$, $PersistsBetween(0, IN(Lisa, LivingRoom), 2)$ (which follows from (10.10) and EC7), $\neg Clipped(0, IN(Lisa, LivingRoom), 2)$ (which follows from (10.12), (10.14), and EC1), and EC9, we have

$$HoldsAt(IN(Lisa, LivingRoom), 2) \tag{10.15}$$

From (10.6) (which follows from (10.14)), (10.15), RS14 (which follows from (10.11)), $2 < 4$, $\neg StoppedIn(2, IN(Lisa, Kitchen), 4)$ (which follows from (10.12), (10.14), and EC3), $\neg ReleasedIn(2, IN(Lisa, Kitchen), 4)$ (which follows from (10.13) and EC13), and EC14, we have $HoldsAt(IN(Lisa, Kitchen), 4)$. From this, (10.8) (which follows from (10.14)), RS14 (which follows from (10.11)), $4 < 5$, $\neg StoppedIn(4, IN(Lisa, LivingRoom), 5)$ (which follows from (10.14) and EC3), $\neg ReleasedIn(4, IN(Lisa, LivingRoom), 5)$ (which follows from (10.13) and EC13), and EC14, we have $HoldsAt(IN(Lisa, LivingRoom), 5)$.

From (10.1), $0 < 1$, $PersistsBetween(0, IN(Lisa, LivingRoom), 1)$ (which follows from (10.10) and EC7), $\neg Clipped(0, IN(Lisa, LivingRoom), 1)$ (which follows from (10.12), (10.14), and EC1), and EC9, we have

$$HoldsAt(IN(Lisa, LivingRoom), 1) \tag{10.16}$$

From (10.3), $0 < 1$, $PersistsBetween(0, IN(Box, LivingRoom), 1)$ (which follows from (10.10) and EC7), $\neg Clipped(0, IN(Box, LivingRoom), 1)$ (which follows from (10.12), (10.14), and EC1), and EC9, we have

$$HoldsAt(IN(Box, LivingRoom), 1) \tag{10.17}$$

From (10.6) (which follows from (10.14)), (10.15), RS14 (which follows from (10.11)), $2 < 3$, $\neg StoppedIn(2, IN(Lisa, Kitchen), 3)$ (which follows from (10.14) and EC3), $\neg ReleasedIn(2, IN(Lisa, Kitchen), 3)$ (which follows from (10.13) and EC13), and EC14, we have

$$HoldsAt(IN(Lisa, Kitchen), 3) \tag{10.18}$$

From (10.5) (which follows from (10.14)), (10.16), (10.17), RS16, $1 < 3$, $\neg StoppedIn(1, IN(Box, Lisa), 3)$ (which follows from (10.12), (10.14), and EC3), $\neg ReleasedIn(1, IN(Box, Lisa), 3)$ (which follows from (10.13) and EC13), and EC14, we have $HoldsAt(IN(Box, Lisa), 3)$. From this, (10.7) (which follows from (10.14)), (10.18), RS18 (which follows from (10.11)), $3 < 5$, $\neg StoppedIn(3, IN(Box, Kitchen), 5)$ (which follows from (10.12), (10.14), and EC3), $\neg ReleasedIn(3, IN(Box, Kitchen), 5)$ (which follows from (10.13) and EC13), and EC14, we have

$$HoldsAt(IN(Box, Kitchen), 5) \tag{10.19}$$

From (10.3) and RS7, we have $HoldsAt(INROOM(Box, LivingRoom), 0)$. From this, (10.4) (which follows from (10.14)), (10.1), (10.2), RS10, $0 < 5$, $\neg StoppedIn(0, IN(Newspaper, Box), 5)$ (which follows from (10.12), (10.14), and EC3), $\neg ReleasedIn(0, IN(Newspaper, Box), 5)$ (which follows from (10.13) and EC13), and EC14, we have

$$HoldsAt(IN(Newspaper, Box), 5) \tag{10.20}$$

From (10.19) and RS7, we have $HoldsAt(INROOM(Box, Kitchen), 5)$. From this, (10.20), and RS8, we have $HoldsAt(INROOM(Newspaper, Kitchen), 5)$. ■

10.2 Metric Space

DEFINITION 10.1

A ***metric space*** is a set M of objects called ***points*** and a function d (the distance function or metric) from $M \times M$ to the set of real numbers such that, for every $x, y, z \in M$,

- $d(x, y) \geq 0$
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$
- $d(x, z) \leq d(x, y) + d(y, z)$ (known as the ***triangle inequality***)

The commonsense world involves objects located at specific points in a metric space and objects that trace specific paths over time in that space. For example, when an apple located eight feet above the ground starts to fall, it follows a particular path toward the ground. In this section, we present a representation of metric space in the event calculus. (Another representation of metric space is discussed in Section 8.2.2.)

We start with a basic representation as we did for relational space:

- an object sort, with variables o, o_1, o_2, ...
- a real number sort, with variables x, x_1, x_2, ...; y, y_1, y_2, ...; and z, z_1, z_2, ...
- a representation for points in three-dimensional space consisting of triples of real numbers $\langle x, y, z \rangle$
- a fluent $AT(o, x, y, z)$, which represents that object o is located at point $\langle x, y, z \rangle$
- a function $D(x_1, y_1, z_1, x_2, y_2, z_2)$, which represents the distance between the points $\langle x_1, y_1, z_1 \rangle$ and $\langle x_2, y_2, z_2 \rangle$

We may wish to require an object to be located at one point at a time via the following axiom.

AXIOM MS1

$$HoldsAt(AT(o, x_1, y_1, z_1), t) \wedge HoldsAt(AT(o, x_2, y_2, z_2), t) \Rightarrow$$
$$x_1 = x_2 \wedge y_1 = y_2 \wedge z_1 = z_2$$

We specify the *DIST* function.

AXIOM MS2

$$DIST(x_1, y_1, z_1, x_2, y_2, z_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

We can then extend this basic representation with motion and collisions.

10.2.1 Example: Two Baseballs Colliding

Nathan and Ryan each throw a baseball into the air. A moment later, the baseballs collide.

We treat this example using both relational and metric space. We start with some effect axioms. If an agent is holding an object and the agent throws the object into the air with a velocity vector $\langle v_x, v_y, v_z \rangle$, where the z axis represents up, then the object will be flying with that velocity vector:

$$HoldsAt(IN(o, a), t) \Rightarrow \tag{10.21}$$
$$Initiates(Throw(a, o, v_x, v_y, v_z), Flying(o, v_x, v_y, v_z), t)$$

If an agent is holding an object and the agent throws the object into the air, then the agent will no longer be holding the object:

$$HoldsAt(IN(o,a),t) \Rightarrow \\ Terminates(Throw(a,o,v_x,v_y,v_z),IN(o,a),t) \tag{10.22}$$

If an object is thrown into the air, then its location and velocity will no longer be subject to the commonsense law of inertia:

$$HoldsAt(IN(o,a),t) \Rightarrow \\ Releases(Throw(a,o,v_x,v_y,v_z),AT(o,x,y,z),t) \tag{10.23}$$

$$HoldsAt(IN(o,a),t) \Rightarrow \\ Releases(Throw(a,o,v_{x1},v_{y1},v_{z1}),Velocity(o,v_{x2},v_{y2},v_{z2}),t) \tag{10.24}$$

We specify the location and velocity of the flying object using the following trajectory axioms:

$$HoldsAt(AT(o,x_0,y_0,z_0),t) \wedge \\ x_1 = x_0 + v_x t_2 \wedge \\ y_1 = y_0 + v_y t_2 \wedge \\ z_1 = z_0 + v_z t_2 - \frac{1}{2} G t_2^2 \Rightarrow \\ Trajectory(Flying(o,v_x,v_y,v_z),t_1,AT(o,x_1,y_1,z_1),t_2) \tag{10.25}$$

$$v_{z2} = v_{z1} - G t_2 \Rightarrow \\ Trajectory(Flying(o,v_x,v_y,v_{z1}),t_1,Velocity(o,v_x,v_y,v_{z2}),t_2) \tag{10.26}$$

We have state constraints that say that at any given moment an object has a unique velocity and is flying with a unique velocity:

$$HoldsAt(Velocity(o,v_{x1},v_{y1},v_{z1}),t) \wedge \\ HoldsAt(Velocity(o,v_{x2},v_{y2},v_{z2}),t) \Rightarrow \\ v_{x1} = v_{x2} \wedge v_{y1} = v_{y2} \wedge v_{z1} = v_{z2} \tag{10.27}$$

$$HoldsAt(Flying(o,v_{x1},v_{y1},v_{z1}),t) \wedge \\ HoldsAt(Flying(o,v_{x2},v_{y2},v_{z2}),t) \Rightarrow \\ v_{x1} = v_{x2} \wedge v_{y1} = v_{y2} \wedge v_{z1} = v_{z2} \tag{10.28}$$

We use a trigger axiom to represent that an elastic collision occurs when two flying objects are at the same location:

$$\begin{aligned} & o_1 \neq o_2 \wedge \\ & \mathit{HoldsAt}(\mathit{Flying}(o_1, v_{1x}, v_{1y}, v_{1z}), t) \wedge \\ & \mathit{HoldsAt}(\mathit{Flying}(o_2, v_{2x}, v_{2y}, v_{2z}), t) \wedge \\ & \mathit{HoldsAt}(\mathit{AT}(o_1, x, y, z), t) \wedge \\ & \mathit{HoldsAt}(\mathit{AT}(o_2, x, y, z), t) \Rightarrow \\ & \mathit{Happens}(\mathit{Collide}(o_1, o_2), t) \end{aligned} \tag{10.29}$$

We use effect axioms to represent that, when object o_1 elastically collides with object o_2, object o_1 assumes the velocity of o_2:

$$\begin{aligned} & \mathit{HoldsAt}(\mathit{Velocity}(o_1, v_{1x}, v_{1y}, v_{1z}), t) \wedge \\ & \mathit{HoldsAt}(\mathit{Velocity}(o_2, v_{2x}, v_{2y}, v_{2z}), t) \Rightarrow \\ & \mathit{Initiates}(\mathit{Collide}(o_1, o_2), \mathit{Flying}(o_1, v_{2x}, v_{2y}, v_{2z}), t) \end{aligned} \tag{10.30}$$

$$\begin{aligned} & \mathit{HoldsAt}(\mathit{Velocity}(o_1, v_x, v_y, v_z), t) \Rightarrow \\ & \mathit{Terminates}(\mathit{Collide}(o_1, o_2), \mathit{Flying}(o_1, v_x, v_y, v_z), t) \end{aligned} \tag{10.31}$$

That is, two elastically colliding objects exchange their velocities. We assume the objects are points of equal mass.

Let us use the following observations and narrative. Initially, one baseball is at rest at location $\langle 0, 0, G/2 \rangle$, and another baseball is at rest at location $\langle 1, 0, G/2 \rangle$:

$$\neg \mathit{ReleasedAt}(f, t) \tag{10.32}$$

$$\mathit{HoldsAt}(\mathit{AT}(\mathit{Baseball1}, 0, 0, G/2), 0) \tag{10.33}$$

$$\mathit{HoldsAt}(\mathit{Velocity}(\mathit{Baseball1}, 0, 0, 0), 0) \tag{10.34}$$

$$\mathit{HoldsAt}(\mathit{AT}(\mathit{Baseball2}, 1, 0, G/2), 0) \tag{10.35}$$

$$\mathit{HoldsAt}(\mathit{Velocity}(\mathit{Baseball2}, 0, 0, 0), 0) \tag{10.36}$$

Nathan is holding the first baseball, and Ryan is holding the second baseball:

$$\mathit{HoldsAt}(\mathit{IN}(\mathit{Baseball1}, \mathit{Nathan}), 0) \tag{10.37}$$

$$\mathit{HoldsAt}(\mathit{IN}(\mathit{Baseball2}, \mathit{Ryan}), 0) \tag{10.38}$$

Nathan and Ryan throw the baseballs toward one another. Nathan throws the first baseball with velocity $\langle 1, 0, 0 \rangle$, and Ryan throws the second baseball with velocity $\langle -1, 0, 0 \rangle$:

$$\mathit{Happens}(\mathit{Throw}(\mathit{Nathan}, \mathit{Baseball1}, 1, 0, 0), 0) \tag{10.39}$$

$$\mathit{Happens}(\mathit{Throw}(\mathit{Ryan}, \mathit{Baseball2}, -1, 0, 0), 0) \tag{10.40}$$

We also have the fact that the two baseballs are not the same object:

$$Baseball1 \neq Baseball2 \tag{10.41}$$

We can then show that the baseballs will collide. In order to show this, we must assume that the two baseballs are the only objects in the metric space:

$$HoldsAt(AT(o, x, y, z), t) \Rightarrow (o = Baseball1 \vee o = Baseball2) \tag{10.42}$$

Without this assumption, either of the baseballs might first collide with some other object.

We start with a lemma that states that, if object o_1 collides with object o_2, then o_2 collides with o_1:

LEMMA 10.1

If Γ = (10.29), then we have

$$CIRC[\Gamma; Happens] \wedge Happens(Collide(o_1, o_2), t) \Rightarrow Happens(Collide(o_2, o_1), t)$$

Proof Let ω_1 and ω_2 be arbitrary objects, and τ be an arbitrary timepoint. We must show

$$CIRC[\Gamma; Happens] \wedge Happens(Collide(\omega_1, \omega_2), \tau) \Rightarrow Happens(Collide(\omega_2, \omega_1), \tau)$$

Suppose the following:

$$CIRC[\Gamma; Happens] \tag{10.43}$$

$$Happens(Collide(\omega_1, \omega_2), \tau) \tag{10.44}$$

From (10.43) and Theorem 2.1, we have

$$\begin{aligned} &Happens(Collide(o_1, o_2), t) \Leftrightarrow \\ &\exists v_{1x}, v_{1y}, v_{1z}, v_{2x}, v_{2y}, v_{2z}, x, y, z \\ &(o_1 \neq o_2 \wedge \\ &HoldsAt(Flying(o_1, v_{1x}, v_{1y}, v_{1z}), t) \wedge \\ &HoldsAt(Flying(o_2, v_{2x}, v_{2y}, v_{2z}), t) \wedge \\ &HoldsAt(AT(o_1, x, y, z), t) \wedge \\ &HoldsAt(AT(o_2, x, y, z), t)) \end{aligned} \tag{10.45}$$

From this and (10.44), we have:

$$\omega_1 \neq \omega_2. \tag{10.46}$$

$$HoldsAt(Flying(\omega_1, V1X, V1Y, V1Z), \tau) \tag{10.47}$$

$$HoldsAt(Flying(\omega_2, V2X, V2Y, V2Z), \tau) \tag{10.48}$$

$$HoldsAt(AT(\omega_1, X, Y, Z), \tau) \tag{10.49}$$

$$HoldsAt(AT(\omega_2, X, Y, Z), \tau) \tag{10.50}$$

for some $V1X$, $V1Y$, $V1Z$, $V2X$, $V2Y$, $V2Z$, X, Y, and Z. From (10.46), (10.47), (10.48), (10.49), (10.50), and (10.45), we have $Happens(Collide(\omega_2, \omega_1), \tau)$, as required. ■

Now we show that the baseballs collide.

PROPOSITION 10.2

Let Σ be the conjunction of (10.21), (10.22), (10.23), (10.24), (10.30), and (10.31). Let Δ be the conjunction of (10.29), (10.39), and (10.40). Let Ω be $U[IN, AT, Flying, Velocity] \wedge U[Throw, Collide] \wedge (10.41)$. Let Ψ be the conjunction of RS1 through RS4, MS1, (10.27), (10.28), and (10.42). Let Π be the conjunction of (10.25) and (10.26). Let Γ be the conjunction of (10.32), (10.33), (10.34), (10.35), (10.36), (10.37), and (10.38). Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge EC \models Happens(Collide(Baseball1, Baseball2), 0.5)$$

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

$$\begin{aligned}
&Initiates(e, f, t) \Leftrightarrow \qquad (10.51)\\
&\exists a, o, v_x, v_y, v_z\, (e = Throw(a, o, v_x, v_y, v_z) \wedge\\
&f = Flying(o, v_x, v_y, v_z) \wedge\\
&HoldsAt(IN(o, a), t)) \vee\\
&\exists o_1, o_2, v_{1x}, v_{1y}, v_{1z}, v_{2x}, v_{2y}, v_{2z}\\
&(e = Collide(o_1, o_2) \wedge\\
&f = Flying(o_1, v_{2x}, v_{2y}, v_{2z}) \wedge\\
&HoldsAt(Velocity(o_1, v_{1x}, v_{1y}, v_{1z}), t) \wedge\\
&HoldsAt(Velocity(o_2, v_{2x}, v_{2y}, v_{2z}), t))
\end{aligned}$$

$$\begin{aligned}
&\mathit{Terminates}(e,f,t) \Leftrightarrow \\
&\exists a,o,v_x,v_y,v_z\,(e = \mathit{Throw}(a,o,v_x,v_y,v_z) \wedge \\
&f = \mathit{IN}(o,a) \wedge \\
&\mathit{HoldsAt}(\mathit{IN}(o,a),t)) \vee \\
&\exists o_1,o_2,v_x,v_y,v_z\,(e = \mathit{Collide}(o_1,o_2) \wedge \\
&f = \mathit{Flying}(o_1,v_x,v_y,v_z) \wedge \\
&\mathit{HoldsAt}(\mathit{Velocity}(o_1,v_x,v_y,v_z),t))
\end{aligned} \tag{10.52}$$

$$\begin{aligned}
&\mathit{Releases}(e,f,t) \Leftrightarrow \\
&\exists a,o,v_x,v_y,v_z\,(e = \mathit{Throw}(a,o,v_x,v_y,v_z) \wedge \\
&f = \mathit{AT}(o,x,y,z) \wedge \\
&\mathit{HoldsAt}(\mathit{IN}(o,a),t)) \vee \\
&\exists a,o,v_{x1},v_{y1},v_{z1},v_{x2},v_{y2},v_{z2} \\
&(e = \mathit{Throw}(a,o,v_{x1},v_{y1},v_{z1}) \wedge \\
&f = \mathit{Velocity}(o,v_{x2},v_{y2},v_{z2}) \wedge \\
&\mathit{HoldsAt}(\mathit{IN}(o,a),t))
\end{aligned} \tag{10.53}$$

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

$$\begin{aligned}
&\mathit{Happens}(e,t) \Leftrightarrow \\
&\exists o_1,o_2,x,y,z,v_{1x},v_{1y},v_{1z},v_{2x},v_{2y},v_{2z} \\
&(e = \mathit{Collide}(o_1,o_2) \wedge \\
&o_1 \neq o_2 \wedge \\
&\mathit{HoldsAt}(\mathit{Flying}(o_1,v_{1x},v_{1y},v_{1z}),t) \wedge \\
&\mathit{HoldsAt}(\mathit{Flying}(o_2,v_{2x},v_{2y},v_{2z}),t) \wedge \\
&\mathit{HoldsAt}(\mathit{AT}(o_1,x,y,z),t) \wedge \\
&\mathit{HoldsAt}(\mathit{AT}(o_2,x,y,z),t)) \vee \\
&(e = \mathit{Throw}(\mathit{Nathan}, \mathit{Baseball1}, 1,0,0) \wedge t = 0) \vee \\
&(e = \mathit{Throw}(\mathit{Ryan}, \mathit{Baseball2}, -1,0,0) \wedge t = 0)
\end{aligned} \tag{10.54}$$

We can show

$$\neg\exists o,t\,(0 < t < 0.5 \wedge \mathit{Happens}(\mathit{Collide}(\mathit{Baseball1},o),t)) \tag{10.55}$$

To see this, suppose to the contrary that

$$\exists o,t\,(0 < t < 0.5 \wedge \mathit{Happens}(\mathit{Collide}(\mathit{Baseball1},o),t))$$

Let $Happens(Collide(Baseball1, \omega), \tau)$ be the first such event. That is, we have

$$0 < \tau < 0.5 \tag{10.56}$$

$$Happens(Collide(Baseball1, \omega), \tau) \tag{10.57}$$

$$\neg\exists\omega', \tau' (0 < \tau' < \tau \wedge Happens(Collide(Baseball1, \omega'), \tau')) \tag{10.58}$$

From (10.57) and (10.54), we have

$$Baseball1 \neq \omega \tag{10.59}$$

$$HoldsAt(AT(Baseball1, X1, Y1, Z1), \tau) \tag{10.60}$$

$$HoldsAt(AT(\omega, X1, Y1, Z1), \tau) \tag{10.61}$$

for some $X1$, $Y1$, and $Z1$. From (10.59), (10.60), (10.61), and (10.42), we have $\omega = Baseball2$. From (10.52), (10.58), and EC3, we have $\neg StoppedIn(0, Flying(Baseball1, 1, 0, 0), \tau)$. From this, (10.39) (which follows from (10.54)), (10.37), (10.21) (which follows from (10.51)), $0 < \tau$ (which follows from (10.56)), (10.33), (10.25), and EC5, we have $HoldsAt(AT(Baseball1, \tau, 0, \frac{1}{2}G(1 - \tau^2)), \tau)$. From this, (10.60), (10.61), and $\omega = Baseball2$, we have

$$HoldsAt(AT(Baseball2, \tau, 0, \frac{1}{2}G(1 - \tau^2)), \tau) \tag{10.62}$$

From (10.58), we have

$$\neg\exists\tau' (0 < \tau' < \tau \wedge Happens(Collide(Baseball1, Baseball2), \tau')) \tag{10.63}$$

From (10.54), we have $CIRC[(10.29); Happens]$. From this, (10.63), and Lemma 10.1, we have

$$\neg\exists\tau' (0 < \tau' < \tau \wedge Happens(Collide(Baseball2, Baseball1), \tau'))$$

From this, (10.52), and EC3, we have $\neg StoppedIn(0, Flying(Baseball2, -1, 0, 0), \tau)$. From this, (10.40) (which follows from (10.54)), (10.38), (10.21) (which follows from (10.51)), $0 < \tau$ (which follows from (10.56)), (10.35), (10.25), and EC5, we have $HoldsAt(AT(Baseball2, 1 - \tau, 0, \frac{1}{2}G(1 - \tau^2)), \tau)$. From this, (10.56), and MS1, we have $\neg HoldsAt(AT(Baseball2, \tau, 0, \frac{1}{2}G(1 - \tau^2)), \tau)$, which contradicts (10.62).

From (10.52), (10.55), and EC3, we have

$$\neg StoppedIn(0, Flying(Baseball1, 1, 0, 0), 0.5) \tag{10.64}$$

From this, (10.39) (which follows from (10.54)), (10.37), (10.21) (which follows from (10.51)), $0 < 0.5$, (10.33), (10.25), and EC5, we have

$$HoldsAt(AT(Baseball1, 0.5, 0, \frac{3}{8}G), 0.5) \tag{10.65}$$

From (10.53) and EC13, we have $\neg ReleasedIn(0, Flying(Baseball1, 1, 0, 0), 0.5)$. From this, (10.39) (which follows from (10.54)), (10.37), (10.21) (which follows from (10.51)), $0 < 0.5$, (10.64), and EC14, we have

$$HoldsAt(Flying(Baseball1, 1, 0, 0), 0.5) \tag{10.66}$$

Using similar arguments we can show

$$HoldsAt(AT(Baseball2, 0.5, 0, \frac{3}{8}G), 0.5) \tag{10.67}$$
$$HoldsAt(Flying(Baseball2, -1, 0, 0), 0.5) \tag{10.68}$$

From (10.41), (10.66), (10.68), (10.65), (10.67), and (10.29) (which follows from (10.54)), we have $Happens(Collide(Baseball1, Baseball2), 0.5)$. ■

10.3 Object Identity

We can express object identity in first-order logic using the atom $o_1 = o_2$, which represents that o_1 and o_2 are the same object. The commonsense world of space and time places certain restrictions on object identity, which we can reason about using the event calculus. In this section, we consider the problem of observing the locations of two objects over time and determining whether the two objects must be the same, may be the same, or must be different.

We use the following spatial theory. The predicate $Adjacent(l_1, l_2)$ represents that location l_1 is adjacent to location l_2. The event $Move(o, l_1, l_2)$ represents that object o moves from location l_1 to location l_2. The fluent $At(o, l)$ represents that object o is located at location l. An object is at exactly one location at a time:

$$HoldsAt(At(o, l_1), t) \wedge HoldsAt(At(o, l_2), t) \Rightarrow l_1 = l_2 \tag{10.69}$$
$$\exists l\ HoldsAt(At(o, l), t) \tag{10.70}$$

Two objects cannot occupy the same location at the same time:

$$HoldsAt(At(o_1, l), t) \wedge HoldsAt(At(o_2, l), t) \Rightarrow o_1 = o_2 \tag{10.71}$$

The *Adjacent* predicate is symmetric:

$$Adjacent(l_1, l_2) \Leftrightarrow Adjacent(l_2, l_1) \tag{10.72}$$

If an object is at location l_1, which is adjacent to location l_2, and the object moves from l_1 to l_2, then the object will be at l_2 and will no longer be at l_1:

$$HoldsAt(At(o, l_1), t) \wedge Adjacent(l_1, l_2) \Rightarrow \tag{10.73}$$
$$Initiates(Move(o, l_1, l_2), At(o, l_2), t)$$

$$HoldsAt(At(o, l_1), t) \wedge Adjacent(l_1, l_2) \Rightarrow \\ Terminates(Move(o, l_1, l_2), At(o, l_1), t) \tag{10.74}$$

10.3.1 Example: One Screen

Consider the following scenario involving three locations: The location *L1* is to the left of *L2*, which is to the left of *L3*:

$$Adjacent(l_1, l_2) \Leftrightarrow \\ (l_1 = L1 \wedge l_2 = L2) \vee \\ (l_1 = L2 \wedge l_2 = L1) \vee \\ (l_1 = L2 \wedge l_2 = L3) \vee \\ (l_1 = L3 \wedge l_2 = L2) \tag{10.75}$$

Our view of location *L2* is blocked by a screen.

Suppose we observe the following: At timepoint 0, we observe an object, let us call it *O1*, at *L1* and nothing at *L3*. At timepoint 1, we observe no objects at *L1* or *L3*. At timepoint 2, we observe an object, let us call it *O2*, at *L3* and nothing at *L1*. We observe nothing about *L2* because it is blocked by a screen.

Thus, we have the following observations:

$$HoldsAt(At(O1, L1), 0) \tag{10.76}$$
$$\neg HoldsAt(At(o, L3), 0) \tag{10.77}$$
$$\neg HoldsAt(At(o, L1), 1) \tag{10.78}$$
$$\neg HoldsAt(At(o, L3), 1) \tag{10.79}$$
$$HoldsAt(At(O2, L3), 2) \tag{10.80}$$
$$\neg HoldsAt(At(o, L1), 2) \tag{10.81}$$
$$\neg ReleasedAt(f, t) \tag{10.82}$$

We can then show that *O1* and *O2* must be the same object.

PROPOSITION 10.3

Let $\Sigma = (10.73) \wedge (10.74)$, $\Psi = (10.69) \wedge (10.70) \wedge (10.71) \wedge (10.72)$, and $\Gamma = (10.75) \wedge (10.76) \wedge (10.77) \wedge (10.78) \wedge (10.79) \wedge (10.80) \wedge (10.81) \wedge (10.82)$. Suppose

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge \Psi \wedge \Gamma \wedge \text{DEC}$$

Then $O1 = O2$.

Proof From *CIRC*[Σ; *Initiates*, *Terminates*, *Releases*], Theorem 2.2, and Theorem 2.1, we have

$$\begin{aligned}&Initiates(e,f,t) \Leftrightarrow \\ &\exists o, l_1, l_2\,(e = Move(o,l_1,l_2) \wedge \\ &f = At(o,l_2) \wedge \\ &HoldsAt(At(o,l_1),t) \wedge \\ &Adjacent(l_1,l_2))\end{aligned} \tag{10.83}$$

$$\begin{aligned}&Terminates(e,f,t) \Leftrightarrow \\ &\exists o, l_1, l_2\,(e = Move(o,l_1,l_2) \wedge \\ &f = At(o,l_1) \wedge \\ &HoldsAt(At(o,l_1),t) \wedge \\ &Adjacent(l_1,l_2))\end{aligned} \tag{10.84}$$

$$\neg Releases(e,f,t) \tag{10.85}$$

From (10.80) and the contrapositive of DEC6, we have

$$\begin{aligned}&HoldsAt(At(O2,L3),1) \vee \\ &ReleasedAt(At(O2,L3),2) \vee \\ &\exists e\,(Happens(e,1) \wedge Initiates(e,At(O2,L3),1))\end{aligned}$$

From this, (10.79), and (10.82), we have

$$\exists e\,(Happens(e,1) \wedge Initiates(e,At(O2,L3),1))$$

From this, (10.83), and (10.75), we have

$$HoldsAt(At(O2,L2),1) \tag{10.86}$$

From (10.78), we have $\neg HoldsAt(At(O1,L1),1)$. From this and the contrapositive of DEC5, we have

$$\begin{aligned}&\neg HoldsAt(At(O1,L1),0) \vee \\ &ReleasedAt(At(O1,L1),1) \vee \\ &\exists e\,(Happens(e,0) \wedge Terminates(e,At(O1,L1),0))\end{aligned}$$

From this, (10.76), and (10.82), we have

$$\exists e\,(Happens(e,0) \wedge Terminates(e,At(O1,L1),0))$$

From this, (10.84), and (10.75), we have $Happens(Move(O1, L1, L2), 0)$. From this, (10.76), (10.75), (10.83), and DEC9, we have

$$HoldsAt(At(O1, L2), 1)$$

From this, (10.86), and (10.71), we have $O1 = O2$. ■

10.3.2 Example: Two Screens

Now consider the following scenario involving five locations: The location $L1$ is to the left of $L2$, which is to the left of $L3$, which is to the left of $L4$, which is to the left of $L5$:

$$\begin{aligned} &Adjacent(l_1, l_2) \Leftrightarrow \\ &(l_1 = L1 \wedge l_2 = L2) \vee \\ &(l_1 = L2 \wedge l_2 = L1) \vee \\ &(l_1 = L2 \wedge l_2 = L3) \vee \\ &(l_1 = L3 \wedge l_2 = L2) \vee \\ &(l_1 = L3 \wedge l_2 = L4) \vee \\ &(l_1 = L4 \wedge l_2 = L3) \vee \\ &(l_1 = L4 \wedge l_2 = L5) \vee \\ &(l_1 = L5 \wedge l_2 = L4) \end{aligned} \tag{10.87}$$

Locations $L2$ and $L4$ are blocked by screens.

This time we observe the following. At timepoint 0, we observe $O1$ at $L1$ and nothing at $L5$. At timepoint 4, we observe $O2$ at $L5$ and nothing at $L1$. We never observe an object at $L3$, and we never observe anything about $L2$ or $L4$.

Thus, we have the following observations:

$$HoldsAt(At(O1, L1), 0) \tag{10.88}$$
$$\neg HoldsAt(At(o, L5), 0) \tag{10.89}$$
$$HoldsAt(At(O2, L5), 4) \tag{10.90}$$
$$\neg HoldsAt(At(o, L1), 4) \tag{10.91}$$
$$\neg HoldsAt(At(o, L3), t) \tag{10.92}$$
$$\neg ReleasedAt(f, t) \tag{10.93}$$

In this case, we can show that $O1$ and $O2$ cannot be the same object (see Exercise 10.5).

Bibliographic Notes

Hayes (1985) describes the following ontology for liquids. Liquids can be lazy or energetic. Lazy liquids can be still or moving. Rain is a lazy moving liquid, and a splash is an energetic liquid. Liquids can be bulk or divided. Bulk and divided liquids can be supported by a two-dimensional object, supported by a three-dimensional object, or unsupported. A mist is a lazy, still, divided, unsupported liquid. A river is a lazy, moving, bulk liquid supported by a three-dimensional object.

E. Davis (1990, pp. 241–310) reviews a number of spatial representations including geometric entities, occupancy arrays, constructive solid geometry, boundary representations, topological route maps, and configuration spaces. E. Davis (1995) describes a formal language for describing objects in two-dimensional space. The language provides the following entities: real number, point, length, angle, direction, area, vector, frame of reference, interval, arc, region, directed curve, scale mapping, and collection of regions. Partial and inexact information can be expressed in the language. For example, the language can be used to represent a circle whose diameter is between 1 and 2 inches, with a triangular hole somewhere inside it.

Randell, Cui, and Cohn (1992) introduce what has become known as the *region connection calculus* (RCC), a logic for spatial reasoning. RCC is based on a single predicate $C(x,y)$, which represents that region x connects with region y. A number of predicates can be defined in terms of C to represent that one region is disconnected from another region, one region is a part of another region, one region overlaps another region, and so forth. Cohn, Bennett, Gooday, and Gotts (1997) provide an introduction to RCC.

Kuipers (2000) describes the *spatial semantic hierarchy*, a collection of interacting representations of large-scale space. Representations are classified along two dimensions: (1) qualitative and quantitative, and (2) sensory, control, causal, topological, and metrical. In this hierarchy, our representation of relational space is classified as qualitative and topological, whereas our representation of metric space is classified as quantitative and metrical. Kuipers discusses several implementations of the spatial semantic hierarchy within real and virtual robots. Topological and metric spaces are discussed by Marsden (1974).

Our axiomatizations of space use techniques developed in previous event calculus axiomatizations of space (Shanahan, 1996, 2004; Morgenstern, 2001). Our representation of metric space in the event calculus is based on that of Shanahan (1996), which is adapted from his previous situation calculus representation (Shanahan, 1995b). Shanahan (1995b, 1996) uses the predicate $Occupies(o,r)$ to represent that object o occupies region r, where a *region* is a set of points in two-dimensional space. Shanahan (1996) represents a robot that can rotate, move in the direction it is facing, and stop moving when it bumps into an object.

Morgenstern (2001) develops spatial representations in the event calculus for reasoning about cracking an egg and pouring its contents into a bowl. She uses the

fluent $shape(o) = r$ to represent that object o occupies region r, the fluent $At(o, l)$ to represent that object o is at location l, and the event $Move(o, l)$ to represent that object o moves to location l. Objects and locations are related by the *Above* fluent. Morgenstern represents object parts, object sets, eggs, eggshells, egg insides, packages, open and closed containers, solid and liquid phases, material, falling, pouring, leaking, breaking, and cracking. Shanahan (2004) provides representations in the event calculus for reasoning about egg cracking, including representations of parts and wholes of objects. Synge (1960, pp. 94–96) provides equations for the velocities of two objects after collision. The object identity examples involving one and two screens are taken from Cassimatis (2002).

Exercises

10.1 Add doors to the formalization in Section 10.1. Modify Axioms RS14 and RS15 so that, in order for an agent to go from one room to another, the two rooms must have a door in common.

10.2 Extend the formalization in Section 10.1 to represent that one physical object is on another physical object. Because *in* and *on* are different spatial relations, one spatial relation *IN* is no longer sufficient. Use IN_1 to represent *in* and IN_2 to represent *on*. Write state constraints across IN_1 and IN_2. For example, an object cannot be both in and on another object.

10.3 Extend the formalization in Section 10.1 to represent that one physical object is a part of another physical object. Use IN_3 to represent "is a part of."

10.4 Extend the formalization in Section 10.1 to represent that an agent is wearing one or more items of clothing.

10.5 In the object identity example in Section 10.3.2, prove that *O1* and *O2* cannot be the same object.

10.6 Create an event calculus formalization of the grid-based space used in ThoughtTreasure (Mueller, 1998). There are many grids, each consisting of a rectangular array of cells. For any given grid, a given object is located at a unique cell $\langle row, column \rangle$ in that grid at a time. (An object may be in several grids at a time.) If an agent walks from one cell in a grid to another cell in that grid, then the agent will be at that other cell. In order for an agent to walk from one cell to another, the two cells must be adjacent.

10.7 Formalize riding in a car. Include getting into the car, getting out of the car, and driving from one location to another along a street.

CHAPTER 11

The Mental States of Agents

Many instances of commonsense reasoning involve the mental states of agents. An *agent* is an entity that performs purposeful actions in the world in which it exists. Examples of agents are people, animals, organizations, robots, and computer programs. In order to represent and reason about agent behavior—in particular, to interpret an agent's actions and make predictions about an agent's future actions—we must model the mental states of agents. In this chapter we first address agent behavior, which requires the modeling of beliefs, goals, and plans. Then, we discuss the modeling of emotions.

11.1 Beliefs, Goals, and Plans

Agent behavior can be reactive or goal-driven.

11.1.1 Reactive Behavior

Consider the following example of commonsense reasoning:

Ryan is driving along when a squirrel leaps out in front of the car. What will Ryan do? He will slam on the brakes.

Here we are reasoning about the *reactive behavior* of an agent, in which an agent immediately performs an action in response to the situation the agent encounters. We may represent reactive behavior in the event calculus using trigger axioms, which are of the form

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

where γ is a condition, α is an event term, and τ is a timepoint term.

11.1.2 Goal-Driven Behavior

Whereas some agent behavior is reactive, much behavior is goal-driven.

Ryan has not eaten for a while. He gets hungry. He walks into the kitchen. The refrigerator is empty. He goes to a sandwich shop across the street. He eats a sandwich.

Here Ryan initiates a goal to eat some food in response to his hunger. He forms a plan to eat some food in the kitchen. We use these notions of goals and plans to interpret Ryan's actions and predict his future actions. We understand that Ryan walks into the kitchen because of his goal to eat. We understand that his plan involves taking some food out of the refrigerator. He is acting on his belief that there is food in the refrigerator. When he learns the refrigerator is empty, we expect him to revise his plan and go somewhere else.

Intuitively, a *belief* of an agent is a property considered by the agent to be true or false. A *goal* of an agent is a property that the agent desires to be true. An agent's *plan* for a goal consists of actions that the agent intends to perform in order to achieve the goal. An agent activates certain goals in certain situations. An agent's goals tend to persist until they are achieved. When an agent has a goal, the agent activates a plan for achieving the goal. When a plan is active, an agent tries to execute the plan. If a plan fails, an agent activates an alternative plan. When a goal succeeds, whether intentionally or accidentally, the goal and plan are dropped.

11.1.3 Formalization

We now formalize these notions. We represent a plan as a list of actions. We introduce a list sort and terms of the form

$$[\tau_1, \ldots, \tau_n] \tag{11.1}$$

where $\tau_1, \ldots, \tau_n$ are terms. Several functions can be applied to a list: The *Head* of (11.1) is τ_1, its *Tail* is $[\tau_2, \ldots, \tau_n]$, and its *Length* is n. An axiomatization of lists is provided in Appendix A.

We then use the following sorts:

- an agent sort, with variables a, a_1, a_2, ...
- a belief sort, which is a subsort of the fluent sort, with variables b, b_1, b_2, ...
- a goal sort, which is a subsort of the fluent sort, with variables g, g_1, g_2, ...
- a plan sort, which is a subsort of the list sort, with variables p, p_1, p_2, ...

In the event calculus, it is natural to represent persistent goals and plans as fluents subject to the commonsense law of inertia. We represent beliefs, goals, and plans using the following fluents.

Believe(a, b): Agent a has belief b; that is, a believes that b is true.

Goal(a, g): Agent a has an active goal g.

Plan(a, g, p): Agent a has an active plan p for an active goal g.

We may thus write, for example, *HoldsAt*(*Believe*(*Agent*, *At*(*Agent*, *Kitchen*)), 0).

Beliefs, goals, and plans may be added (activated) and dropped (deactivated).

AXIOM A1 $Initiates(AddBelief(a, b), Believe(a, b), t)$

AXIOM A2 $Terminates(DropBelief(a, b), Believe(a, b), t)$

AXIOM A3 $Initiates(AddGoal(a, g), Goal(a, g), t)$

AXIOM A4 $Terminates(DropGoal(a, g), Goal(a, g), t)$

AXIOM A5 $Initiates(AddPlan(a, g, p), Plan(a, g, p), t)$

AXIOM A6 $Terminates(DropPlan(a, g, p), Plan(a, g, p), t)$

Plans are activated for goals as follows. If an agent has a goal, the agent does not believe the goal is true, and no plans are yet active for the goal, then the agent selects and activates a plan for the goal.

AXIOM A7

$$HoldsAt(Goal(a, g), t) \wedge \neg HoldsAt(Believe(a, g), t) \wedge SelectedPlan(a, g, p, t) \wedge \neg \exists p_1 \, HoldsAt(Plan(a, g, p_1), t) \Rightarrow Happens(AddPlan(a, g, p), t)$$

We use *SelectedPlan*(a, g, p, t) to mean that at time t, plan p is the plan selected by agent a for goal g. This predicate represents the agent's planning ability. For a particular domain or problem, we must supply a definition for *SelectedPlan* (a, g, p, t).

Plans are executed and managed as follows. At each timepoint, provided that the agent believes the plan to be sound and the agent does not believe the goal

to be true, the agent executes the next action of the plan, and the plan is updated to consist of the remaining actions.

AXIOM A8

$$HoldsAt(Plan(a,g,p),t) \wedge \neg HoldsAt(Believe(a,g),t) \wedge SoundPlan(a,g,p,t) \Rightarrow Happens(Head(p),t)$$

AXIOM A9

$$HoldsAt(Plan(a,g,p),t) \Rightarrow Happens(DropPlan(a,g,p),t)$$

AXIOM A10

$$HoldsAt(Plan(a,g,p),t) \wedge Length(p) > 1 \wedge \neg HoldsAt(Believe(a,g),t) \wedge SoundPlan(a,g,p,t) \Rightarrow Happens(AddPlan(a,g,Tail(p)),t)$$

We use $SoundPlan(a,g,p,t)$ to mean that at time t, agent a believes that p is a sound plan for achieving goal g. Notice what happens if the agent believes the plan to be unsound or believes the goal to be true: The plan is dropped. If the goal is false, then a new plan for the goal is activated by axiom A7. For a particular domain or problem, we must supply a definition for $SoundPlan(a,g,p,t)$.

This formalization does not deal with multiple active goals, which lead to multiple active plans, which lead to the concurrent execution of plan actions, which may lead to inconsistency (see Exercise 11.7).

Goal success is treated as follows. If an agent has an active goal and the agent believes that the goal is true, then the agent drops the goal:

AXIOM A11

$$HoldsAt(Goal(a,g),t) \wedge HoldsAt(Believe(a,g),t) \Rightarrow Happens(DropGoal(a,g),t)$$

In order to reason about an agent in a particular domain or problem using axioms A1 through A11, we must supply the following:

- Event calculus axioms. These represent the domain.
- Belief revision axioms. These specify how an agent's beliefs are updated in response to events and are typically of the form:

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

and

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

where γ is a condition, α is an event term, β is a fluent term, a is an agent term, and τ is a timepoint term.

- Goal activation axioms. These are typically of the form

$$\gamma \Rightarrow Happens(AddGoal(a, \beta), \tau)$$

where γ is a condition, a is an agent term, β is a fluent term, and τ is a timepoint term.

- Planning axioms. These supply definitions for $SelectedPlan(a, g, p, t)$ and $SoundPlan(a, g, p, t)$.

11.1.4 Example: Hungry Cat Scenario

This example is due to Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah. A cat is sitting on the floor. He is hungry and would love to eat a cookie sitting on a nearby table. He plans to jump onto a nearby chair and, from the chair, onto the table. But after the cat jumps onto the chair, someone moves the chair away from the table. The cat then modifies his plan. He plans to jump onto a nearby shelf and then onto the table. When he jumps onto the shelf, he notices a cupcake lying there, so he gobbles it up immediately and is satiated.

We start by providing the event calculus axioms for the domain. We use a simple spatial theory. If an agent is on surface s_1, it is possible to jump from s_1 to surface s_2, and the agent jumps from s_1 to s_2, then the agent will be on s_2:

$$HoldsAt(On(a, s_1), t) \wedge HoldsAt(CanJump(s_1, s_2), t) \Rightarrow Initiates(Jump(a, s_1, s_2), On(a, s_2), t) \quad (11.2)$$

In this case, the agent will no longer be on surface 1:

$$HoldsAt(On(a, s_1), t) \wedge HoldsAt(CanJump(s_1, s_2), t) \Rightarrow Terminates(Jump(a, s_1, s_2), On(a, s_1), t) \quad (11.3)$$

If surface s_1 is moved from surface s_2 to surface s_3, then it will be possible to jump from s_1 to s_3:

$$Initiates(Move(s_1, s_2, s_3), CanJump(s_1, s_3), t) \quad (11.4)$$

In this case, it will no longer be possible to jump from surface s_1 to surface s_2:

$$Terminates(Move(s_1, s_2, s_3), CanJump(s_1, s_2), t) \quad (11.5)$$

We use a simple theory of eating. If an agent is on a surface, some food is on the surface, and the agent eats the food, then the agent will be satiated:

$$HoldsAt(On(a,s),t) \wedge HoldsAt(On(f,s),t) \Rightarrow Initiates(Eat(a,f), Satiated(a), t) \quad (11.6)$$

If an agent eats food that is on a surface, then the food will no longer be on the surface:

$$HoldsAt(On(a,s),t) \wedge HoldsAt(On(f,s),t) \Rightarrow Terminates(Eat(a,f), On(f,s), t) \quad (11.7)$$

We provide axioms for belief revision:

$$HoldsAt(Believe(a, On(a,s_1)),t) \wedge HoldsAt(Believe(a, CanJump(s_1,s_2)),t) \Rightarrow Initiates(Jump(a,s_1,s_2), Believe(a, On(a,s_2)), t) \quad (11.8)$$

$$HoldsAt(Believe(a, On(a,s_1)),t) \wedge HoldsAt(Believe(a, CanJump(s_1,s_2)),t) \Rightarrow Terminates(Jump(a,s_1,s_2), Believe(a, On(a,s_1)), t) \quad (11.9)$$

$$Initiates(Move(s_1,s_2,s_3), Believe(a, CanJump(s_1,s_3)), t) \quad (11.10)$$

$$Terminates(Move(s_1,s_2,s_3), Believe(a, CanJump(s_1,s_2)), t) \quad (11.11)$$

$$HoldsAt(Believe(a, On(a,s)),t) \wedge HoldsAt(Believe(a, On(f,s)),t) \Rightarrow Initiates(Eat(a,f), Believe(a, Satiated(a)), t) \quad (11.12)$$

$$HoldsAt(Believe(a, On(a,s)),t) \wedge HoldsAt(Believe(a, On(f,s)),t) \Rightarrow Terminates(Eat(a,f), Believe(a, On(f,s)), t) \quad (11.13)$$

We provide planning axioms. We give a definition for $SelectedPlan(a,g,p,t)$ that enables the agent to plan trips consisting of one and two jumps in order to obtain food:

$$SelectedPlan(a,g,p,t) \Leftrightarrow \quad (11.14)$$
$$\exists s_1, s_2, f\ (HoldsAt(Believe(a, On(a,s_1)),t) \wedge$$

$$
\begin{aligned}
&\mathit{HoldsAt}(\mathit{Believe}(a, \mathit{CanJump}(s_1, s_2)), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Believe}(a, \mathit{On}(f, s_2)), t) \wedge \\
&g = \mathit{Satiated}(a) \wedge \\
&p = [\mathit{Jump}(a, s_1, s_2), \mathit{Eat}(a, f)]) \vee \\
&\exists s_1, s_2, s_3, f\,(\mathit{HoldsAt}(\mathit{Believe}(a, \mathit{On}(a, s_1)), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Believe}(a, \mathit{CanJump}(s_1, s_2)), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Believe}(a, \mathit{CanJump}(s_2, s_3)), t) \wedge \\
&\mathit{HoldsAt}(\mathit{Believe}(a, \mathit{On}(f, s_3)), t) \wedge \\
&g = \mathit{Satiated}(a) \wedge \\
&p = [\mathit{Jump}(a, s_1, s_2), \mathit{Wait}(a), \mathit{Jump}(a, s_2, s_3), \mathit{Eat}(a, f)])
\end{aligned}
$$

We give a definition for $\mathit{SoundPlan}(a, g, p, t)$ that deals with the plans selected by (11.14). An agent's plan is sound if and only if for every jumping action in the plan, the agent believes it is possible to jump from the source surface to the destination surface:

$$
\begin{aligned}
&\mathit{SoundPlan}(a, g, p, t) \Leftrightarrow \\
&\forall s_1, s_2\,(\mathit{Jump}(a, s_1, s_2) \in p \Rightarrow \\
&\mathit{HoldsAt}(\mathit{Believe}(a, \mathit{CanJump}(s_1, s_2)), t))
\end{aligned}
\tag{11.15}
$$

In addition to the goal-driven behavior, the agent also has a reactive behavior. If the agent is not satiated, the agent is on a surface, and some food is on the surface, then the agent will eat the food:

$$
\begin{aligned}
&\neg\mathit{HoldsAt}(\mathit{Satiated}(a), t) \wedge \\
&\mathit{HoldsAt}(\mathit{On}(a, s), t) \wedge \\
&\mathit{HoldsAt}(\mathit{On}(f, s), t) \Rightarrow \\
&\mathit{Happens}(\mathit{Eat}(a, f), t)
\end{aligned}
\tag{11.16}
$$

We specify some observations. The cat has the goal of being satiated, and no plans are active:

$$\neg\mathit{ReleasedAt}(f, t) \tag{11.17}$$

$$
\begin{aligned}
&\mathit{HoldsAt}(\mathit{Goal}(a, g), 0) \Leftrightarrow \\
&a = \mathit{Cat} \wedge g = \mathit{Satiated}(\mathit{Cat})
\end{aligned}
\tag{11.18}
$$

$$\neg\mathit{HoldsAt}(\mathit{Plan}(a, g, p), 0) \tag{11.19}$$

The cat is on the floor, food is on the table and the shelf, and it is possible to jump from the floor to the chair, from the chair to the table, and from the shelf to the

table:

$$\begin{aligned} &HoldsAt(On(o,s),0) \Leftrightarrow \\ &(o = Cat \wedge s = Floor) \vee \\ &(o = Food1 \wedge s = Table) \vee \\ &(o = Food2 \wedge s = Shelf) \end{aligned} \tag{11.20}$$

$$\begin{aligned} &HoldsAt(CanJump(s_1,s_2),0) \Leftrightarrow \\ &(s_1 = Floor \wedge s_2 = Chair) \vee \\ &(s_1 = Chair \wedge s_2 = Table) \vee \\ &(s_1 = Shelf \wedge s_2 = Table) \end{aligned} \tag{11.21}$$

The cat believes the cat is on the floor, believes food is on the table, does not believe food is on the shelf, and believes it is possible to jump from the floor to the chair, from the chair to the table, and from the shelf to the table:

$$\begin{aligned} &HoldsAt(Believe(a, On(o,s)),0) \Leftrightarrow \\ &(a = Cat \wedge o = Cat \wedge s = Floor) \vee \\ &(a = Cat \wedge o = Food1 \wedge s = Table) \end{aligned} \tag{11.22}$$

$$\begin{aligned} &HoldsAt(Believe(a, CanJump(s_1,s_2)),0) \Leftrightarrow \\ &(a = Cat \wedge s_1 = Floor \wedge s_2 = Chair) \vee \\ &(a = Cat \wedge s_1 = Chair \wedge s_2 = Table) \vee \\ &(a = Cat \wedge s_1 = Shelf \wedge s_2 = Table) \end{aligned} \tag{11.23}$$

The cat does not believe the cat is satiated:

$$\neg HoldsAt(Believe(Cat, Satiated(Cat)),0) \tag{11.24}$$

The narrative consists of someone moving the chair at timepoint 2:

$$Happens(Move(Chair, Table, Shelf),2) \tag{11.25}$$

We can show that the cat will eat the food on the shelf.

PROPOSITION 11.1 Let Σ be the conjunction of A1 through A6, (11.2), (11.3), (11.4), (11.5), (11.6), (11.7), (11.8), (11.9), (11.10), (11.11), (11.12), and (11.13). Let Δ be the conjunction of A7 through A11, (11.16), and (11.25). Let $\Omega = U[AddBelief, DropBelief, AddGoal, DropGoal, AddPlan, DropPlan, Jump, Move, Eat] \wedge U[Believe, Goal, Plan, On, CanJump, Satiated]$. Let Γ be the

conjunction of L1 through L8 (the axiomatization of lists of Appendix A), (11.14), (11.15), (11.17), (11.18), (11.19), (11.20), (11.21), (11.22), (11.23), and (11.24). Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Gamma \wedge \mathrm{DEC} \models Happens(Eat(Cat, Food2), 6)$$

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

$$\begin{aligned}
&Initiates(e,f,t) \Leftrightarrow \qquad (11.26)\\
&\exists a,b\,(e = AddBelief(a,b) \wedge f = Believe(a,b)) \vee\\
&\exists a,g\,(e = AddGoal(a,g) \wedge f = Goal(a,g)) \vee\\
&\exists a,g,p\,(e = AddPlan(a,g,p) \wedge f = Plan(a,g,p)) \vee\\
&\exists a,s_1,s_2\,(e = Jump(a,s_1,s_2) \wedge\\
&f = On(a,s_2) \wedge\\
&HoldsAt(On(a,s_1),t) \wedge\\
&HoldsAt(CanJump(s_1,s_2),t)) \vee\\
&\exists s_1,s_2,s_3\,(e = Move(s_1,s_2,s_3) \wedge f = CanJump(s_1,s_3)) \vee\\
&\exists a,f,s\,(e = Eat(a,f) \wedge\\
&f = Satiated(a) \wedge\\
&HoldsAt(On(a,s),t) \wedge\\
&HoldsAt(On(f,s),t)) \vee\\
&\exists a,s_1,s_2\,(e = Jump(a,s_1,s_2) \wedge\\
&f = Believe(a, On(a,s_2)) \wedge\\
&HoldsAt(Believe(a, On(a,s_1)),t) \wedge\\
&HoldsAt(Believe(a, CanJump(s_1,s_2)),t)) \vee\\
&\exists a,s_1,s_2,s_3\,(e = Move(s_1,s_2,s_3) \wedge f = Believe(a, CanJump(s_1,s_3))) \vee\\
&\exists a,f\,(e = Eat(a,f) \wedge\\
&f = Believe(a, Satiated(a)) \wedge\\
&HoldsAt(Believe(a, On(a,s)),t) \wedge\\
&HoldsAt(Believe(a, On(f,s)),t))
\end{aligned}$$

$$\begin{aligned}
&Terminates(e,f,t) \Leftrightarrow \qquad (11.27)\\
&\exists a,b\,(e = DropBelief(a,b) \wedge f = Believe(a,b)) \vee\\
&\exists a,g\,(e = DropGoal(a,g) \wedge f = Goal(a,g)) \vee
\end{aligned}$$

$$
\begin{aligned}
&\exists a, g, p\,(e = DropPlan(a, g, p) \wedge f = Plan(a, g, p)) \vee \\
&\exists a, s_1, s_2 \\
&(e = Jump(a, s_1, s_2) \wedge \\
&f = On(a, s_1) \wedge \\
&HoldsAt(On(a, s_1), t) \wedge \\
&HoldsAt(CanJump(s_1, s_2), t)) \vee \\
&\exists s_1, s_2, s_3\,(e = Move(s_1, s_2, s_3) \wedge f = CanJump(s_1, s_2)) \vee \\
&\exists a, f, s \\
&(e = Eat(a, f) \wedge \\
&f = On(f, s) \wedge \\
&HoldsAt(On(a, s), t) \wedge \\
&HoldsAt(On(f, s), t)) \vee \\
&\exists a, s_1, s_2\,(e = Jump(a, s_1, s_2) \wedge \\
&f = Believe(a, On(a, s_1)) \wedge \\
&HoldsAt(Believe(a, On(a, s_1)), t) \wedge \\
&HoldsAt(Believe(a, CanJump(s_1, s_2)), t)) \vee \\
&\exists a, s_1, s_2, s_3\,(e = Move(s_1, s_2, s_3) \wedge f = Believe(a, CanJump(s_1, s_2))) \vee \\
&\exists a, f, s\,(e = Eat(a, f) \wedge \\
&f = Believe(a, On(f, s)) \wedge \\
&HoldsAt(Believe(a, On(a, s)), t) \wedge \\
&HoldsAt(Believe(a, On(f, s)), t)) \\
\\
&\neg Releases(e, f, t)
\end{aligned}
\tag{11.28}
$$

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

$$
\begin{aligned}
&Happens(e, t) \Leftrightarrow \\
&\exists a, g, p\,(e = AddPlan(a, g, p) \wedge \\
&\neg HoldsAt(Believe(a, g), t) \wedge \\
&HoldsAt(Goal(a, g), t) \wedge \\
&SelectedPlan(a, g, p, t) \wedge \\
&\neg \exists p_1 HoldsAt(Plan(a, g, p_1), t)) \vee \\
&\exists a, g, p\,(e = Head(p) \wedge \\
&HoldsAt(Plan(a, g, p), t) \wedge \\
&\neg HoldsAt(Believe(a, g), t) \wedge \\
&SoundPlan(a, g, p, t)) \vee
\end{aligned}
\tag{11.29}
$$

$$\exists a, g, p\,(e = DropPlan(a, g, p) \wedge HoldsAt(Plan(a, g, p), t)) \vee$$
$$\exists a, g, p\,(e = AddPlan(a, g, Tail(p)) \wedge$$
$$HoldsAt(Plan(a, g, p), t) \wedge$$
$$Length(p) > 1 \wedge$$
$$\neg HoldsAt(Believe(a, g), t) \wedge$$
$$SoundPlan(a, g, p, t)) \vee$$
$$\exists a, g\,(e = DropGoal(a, g) \wedge$$
$$HoldsAt(Goal(a, g), t) \wedge$$
$$HoldsAt(Believe(a, g), t)) \vee$$
$$\exists a, f, s\,(e = Eat(a, f) \wedge$$
$$\neg HoldsAt(Satiated(a), t) \wedge$$
$$HoldsAt(On(a, s), t) \wedge$$
$$HoldsAt(On(f, s), t)) \vee$$
$$(e = Move(Chair, Table, Shelf) \wedge t = 2)$$

We proceed one timepoint at a time, starting with timepoint 0. We start by determining what events occur at timepoint 0.

Timepoint 0: Events Let $P_1 = [Jump(Cat, Floor, Chair), Wait(Cat), Jump(Cat, Chair, Table), Eat(Cat, Food1)]$. Using (11.29), we proceed to show that one event occurs:

$$Happens(e, 0) \Leftrightarrow \quad (11.30)$$
$$e = AddPlan(Cat, Satiated(Cat), P_1)$$

From (11.19), we have

$$\neg\exists e, a, g, p\,(e = Head(p) \wedge \quad (11.31)$$
$$HoldsAt(Plan(a, g, p), 0) \wedge$$
$$\neg HoldsAt(Believe(a, g), 0) \wedge$$
$$SoundPlan(a, g, p, 0))$$

$$\neg\exists e, a, g, p\,(e = DropPlan(a, g, p) \wedge HoldsAt(Plan(a, g, p), 0)) \quad (11.32)$$

$$\neg\exists e, a, g, p\,(e = AddPlan(a, g, Tail(p)) \wedge \quad (11.33)$$
$$HoldsAt(Plan(a, g, p), 0) \wedge$$
$$Length(p) > 1 \wedge$$
$$\neg HoldsAt(Believe(a, g), 0) \wedge$$
$$SoundPlan(a, g, p, 0))$$

From (11.18) and (11.24), we have

$$\neg\exists e, a, g\,(e = DropGoal(a, g) \wedge HoldsAt(Goal(a, g), 0) \wedge HoldsAt(Believe(a, g), 0)) \tag{11.34}$$

From (11.20), we have

$$\neg\exists e, a, f, s\,(e = Eat(a, f) \wedge \neg HoldsAt(Satiated(a), 0) \wedge HoldsAt(On(a, s), 0) \wedge HoldsAt(On(f, s), 0)) \tag{11.35}$$

From (11.22), (11.23), and (11.14), we have

$$SelectedPlan(a, g, p, 0) \Leftrightarrow a = Cat \wedge g = Satiated(Cat) \wedge p = P_1$$

From this, (11.18), (11.19), (11.24), (11.31), (11.32), (11.33), (11.34), (11.35), and (11.29), we have (11.30).

Next we determine the values of relevant fluents at timepoint 1.

Timepoint 1: Fluents From (11.30) and (11.27), we have

$$\neg\exists e, o, s\,(Happens(e, 0) \wedge Terminates(e, On(o, s), 0))$$

From this, (11.20), (11.17), and DEC5, we have

$$HoldsAt(On(Cat, Floor), 1) \tag{11.36}$$

$$HoldsAt(On(Food1, Table), 1) \tag{11.37}$$

$$HoldsAt(On(Food2, Shelf), 1) \tag{11.38}$$

From (11.30) and (11.26), we have

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

From this, (11.20), (11.17), and DEC6, we have

$$\neg(o = Cat \wedge s = Floor) \wedge \neg(o = Food1 \wedge s = Table) \wedge \neg(o = Food2 \wedge s = Shelf) \Rightarrow \neg HoldsAt(On(o, s), 1) \tag{11.39}$$

From (11.30) and (11.26), we have

$$\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, Believe(Cat, Satiated(Cat)), 0))$$

From this, (11.24), (11.17), and DEC6, we have

$$\neg HoldsAt(Believe(Cat, Satiated(Cat)), 1) \tag{11.40}$$

From (11.30) and (11.27), we have

$$\neg\exists e, a, s_1, s_2\,(Happens(e, 0) \wedge Terminates(e, Believe(a, CanJump(s_1, s_2)), 0))$$

From this, (11.23), (11.17), and DEC5, we have

$$HoldsAt(Believe(Cat, CanJump(Floor, Chair)), 1) \tag{11.41}$$
$$HoldsAt(Believe(Cat, CanJump(Chair, Table)), 1) \tag{11.42}$$
$$HoldsAt(Believe(Cat, CanJump(Shelf, Table)), 1) \tag{11.43}$$

From (11.30), A5 (which follows from (11.26)), and DEC9, we have

$$HoldsAt(Plan(Cat, Satiated(Cat), P_1), 1) \tag{11.44}$$

We proceed to show the following:

$$a \neq Cat \vee g \neq Satiated(Cat) \vee p \neq P_1 \Rightarrow \neg HoldsAt(Plan(a, g, p), 1) \tag{11.45}$$

Let α be an arbitrary agent, γ be an arbitrary fluent, and π be an arbitrary list. We must show

$$\alpha \neq Cat \vee \gamma \neq Satiated(Cat) \vee \pi \neq P_1 \Rightarrow \neg HoldsAt(Plan(\alpha, \gamma, \pi), 1)$$

Suppose $\alpha \neq Cat \vee \gamma \neq Satiated(Cat) \vee \pi \neq P_1$. From this, (11.30), and (11.26), we have

$$\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, Plan(\alpha, \gamma, \pi), 0))$$

From this, (11.17), (11.19), and DEC6, we have $\neg HoldsAt(Plan(\alpha, \gamma, \pi), 1)$ as required.

From (11.30) and (11.27), we have

$$\neg\exists e\,(Happens(e, 0) \wedge Terminates(e, Goal(Cat, Satiated(Cat)), 0))$$

From this, (11.18), (11.17), and DEC5, we have

$$HoldsAt(Goal(Cat, Satiated(Cat)), 1) \tag{11.46}$$

We proceed to show

$$a \neq Cat \vee g \neq Satiated(Cat) \Rightarrow \neg HoldsAt(Goal(a, g), 1) \tag{11.47}$$

Let α be an arbitrary agent and γ be an arbitrary fluent. We must show

$$\alpha \neq Cat \vee \gamma \neq Satiated(Cat) \Rightarrow \neg HoldsAt(Goal(\alpha, \gamma), 1)$$

Suppose $\alpha \neq Cat \vee \gamma \neq Satiated(Cat)$. From (11.30) and (11.26), we have

$$\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, Goal(\alpha, \gamma), 0))$$

From this, (11.18), $\alpha \neq Cat \vee \gamma \neq Satiated(Cat)$, (11.17), and DEC6, we have $\neg HoldsAt(Goal(\alpha, \gamma), 1)$ as required.

Timepoint 1: Events Using (11.29), we proceed to show

$$\begin{aligned} &Happens(e, 1) \Leftrightarrow \\ &e = Jump(Cat, Floor, Chair) \vee \\ &e = DropPlan(Cat, Satiated(Cat), P_1) \vee \\ &e = AddPlan(Cat, Satiated(Cat), Tail(P_1)) \end{aligned} \tag{11.48}$$

From (11.46), (11.47), and (11.44), we have

$$\begin{aligned} \neg\exists a, g, p\,(&e = AddPlan(a, g, p) \wedge \\ &\neg HoldsAt(Believe(a, g), 1) \wedge \\ &HoldsAt(Goal(a, g), 1) \wedge \\ &SelectedPlan(a, g, p, 1) \wedge \\ \neg\exists p_1\ &HoldsAt(Plan(a, g, p_1), 1)) \end{aligned} \tag{11.49}$$

From (11.40) and (11.47), we have

$$\begin{aligned} \neg\exists e, a, g\,(&e = DropGoal(a, g) \wedge \\ &HoldsAt(Goal(a, g), 1) \wedge \\ &HoldsAt(Believe(a, g), 1)) \end{aligned} \tag{11.50}$$

From (11.36), (11.37), (11.38), and (11.39), we have

$$\begin{aligned} \neg\exists e, a, f, s\,(&e = Eat(a, f) \wedge \\ &\neg HoldsAt(Satiated(a), 1) \wedge \\ &HoldsAt(On(a, s), 1) \wedge \\ &HoldsAt(On(f, s), 1)) \end{aligned} \tag{11.51}$$

From (11.41), (11.42), (11.43), (11.15), L5, and L6, we have

$$SoundPlan(Cat, Satiated(Cat), P_1, 1)$$

From this, (11.49), (11.50), (11.51), (11.45), (11.44), (11.40), (11.29), L1, L3, and L4, we have (11.48).

Timepoint 2: Fluents Using similar arguments we can show

$$\begin{aligned} &HoldsAt(On(o,s),2) \Leftrightarrow \\ &(o = Cat \wedge s = Chair) \vee \\ &(o = Food1 \wedge s = Table) \vee \\ &(o = Food2 \wedge s = Shelf) \end{aligned} \tag{11.52}$$

$$\begin{aligned} &HoldsAt(Believe(a, CanJump(s_1, s_2)), 1) \Leftrightarrow \\ &(a = Cat \wedge s_1 = Floor \wedge s_2 = Chair) \vee \\ &(a = Cat \wedge s_1 = Chair \wedge s_2 = Table) \vee \\ &(a = Cat \wedge s_1 = Shelf \wedge s_2 = Table) \end{aligned} \tag{11.53}$$

$$\neg HoldsAt(Believe(Cat, Satiated(Cat)), 2) \tag{11.54}$$

$$HoldsAt(Plan(Cat, Satiated(Cat), Tail(P_1)), 2) \tag{11.55}$$

$$\begin{aligned} &a \neq Cat \vee g \neq Satiated(Cat) \vee p \neq Tail(P_1) \Rightarrow \\ &\neg HoldsAt(Plan(a, g, p), 2) \end{aligned} \tag{11.56}$$

$$HoldsAt(Goal(Cat, Satiated(Cat)), 2) \tag{11.57}$$

$$a \neq Cat \vee g \neq Satiated(Cat) \Rightarrow \neg HoldsAt(Goal(a, g), 2) \tag{11.58}$$

Timepoint 2: Events We proceed to show

$$\begin{aligned} &Happens(e, 2) \Leftrightarrow \\ &e = Move(Chair, Table, Shelf) \vee \\ &e = Wait(Cat) \vee \\ &e = DropPlan(Cat, Satiated(Cat), Tail(P_1)) \vee \\ &e = AddPlan(Cat, Satiated(Cat), Tail(Tail(P_1))) \end{aligned} \tag{11.59}$$

From (11.57), (11.58), and (11.55), we have

$$\begin{aligned} &\neg\exists a, g, p\, (e = AddPlan(a, g, p) \wedge \\ &\neg HoldsAt(Believe(a, g), 2) \wedge \\ &HoldsAt(Goal(a, g), 2) \wedge \\ &SelectedPlan(a, g, p, 2) \wedge \\ &\neg\exists p_1\ HoldsAt(Plan(a, g, p_1), 2)) \end{aligned} \tag{11.60}$$

From (11.54) and (11.58), we have

$$\neg\exists e, a, g\,(e = DropGoal(a, g) \wedge HoldsAt(Goal(a, g), 2) \wedge HoldsAt(Believe(a, g), 2)) \tag{11.61}$$

From (11.52), we have

$$\neg\exists e, a, f, s\,(e = Eat(a, f) \wedge \neg HoldsAt(Satiated(a), 2) \wedge HoldsAt(On(a, s), 2) \wedge HoldsAt(On(f, s), 2)) \tag{11.62}$$

From (11.53), (11.15), L5, and L6, we have

$$SoundPlan(Cat, Satiated(Cat), Tail(P_1), 2)$$

From this, (11.60), (11.61), (11.62), (11.56), (11.55), (11.54), (11.29), L1, L2, L3, and L4, we have (11.59).

Timepoint 3: Fluents Using similar arguments we can show

$$HoldsAt(On(o, s), 3) \Leftrightarrow (o = Cat \wedge s = Chair) \vee (o = Food1 \wedge s = Table) \vee (o = Food2 \wedge s = Shelf) \tag{11.63}$$

$$HoldsAt(Believe(a, CanJump(s_1, s_2)), 3) \Leftrightarrow (a = Cat \wedge s_1 = Floor \wedge s_2 = Chair) \vee (a = Cat \wedge s_1 = Chair \wedge s_2 = Shelf) \vee (a = Cat \wedge s_1 = Shelf \wedge s_2 = Table) \tag{11.64}$$

$$\neg HoldsAt(Believe(Cat, Satiated(Cat)), 3) \tag{11.65}$$

$$HoldsAt(Plan(Cat, Satiated(Cat), Tail(Tail(P_1))), 3) \tag{11.66}$$

$$a \neq Cat \vee g \neq Satiated(Cat) \vee p \neq Tail(Tail(P_1)) \Rightarrow \neg HoldsAt(Plan(a, g, p), 3) \tag{11.67}$$

$$HoldsAt(Goal(Cat, Satiated(Cat)), 3) \tag{11.68}$$

$$a \neq Cat \vee g \neq Satiated(Cat) \Rightarrow \neg HoldsAt(Goal(a, g), 3) \tag{11.69}$$

Timepoint 3: Events At timepoint 3, the cat no longer believes that the plan is sound. So the cat does not act on the plan, and the plan is dropped. We proceed to show

$$Happens(e, 3) \Leftrightarrow \\ e = DropPlan(Cat, Satiated(Cat), Tail(Tail(P_1))) \tag{11.70}$$

From (11.68), (11.69), and (11.66), we have

$$\neg\exists a, g, p\,(e = AddPlan(a, g, p) \wedge \\ \neg HoldsAt(Believe(a, g), 3) \wedge \\ HoldsAt(Goal(a, g), 3) \wedge \\ SelectedPlan(a, g, p, 3) \wedge \\ \neg\exists p_1\ HoldsAt(Plan(a, g, p_1), 3)) \tag{11.71}$$

From (11.64), (11.15), L5, and L6, we have

$$\neg SoundPlan(Cat, Satiated(Cat), Tail(Tail(P_1)), 3)$$

From this, (11.66), and (11.67), we have

$$\neg\exists e, a, g, p\,(e = Head(p) \wedge \\ HoldsAt(Plan(a, g, p), 3) \wedge \\ \neg HoldsAt(Believe(a, g), 3) \wedge \\ SoundPlan(a, g, p, 3)) \tag{11.72}$$

$$\neg\exists e, a, g, p\,(e = AddPlan(a, g, Tail(p)) \wedge \\ HoldsAt(Plan(a, g, p), 3) \wedge \\ Length(p) > 1 \wedge \\ \neg HoldsAt(Believe(a, g), 3) \wedge \\ SoundPlan(a, g, p, 3)) \tag{11.73}$$

From (11.65) and (11.69), we have

$$\neg\exists e, a, g\,(e = DropGoal(a, g) \wedge \\ HoldsAt(Goal(a, g), 3) \wedge \\ HoldsAt(Believe(a, g), 3)) \tag{11.74}$$

From (11.63), we have

$$\neg\exists e, a, f, s\,(e = Eat(a, f) \wedge \\ \neg HoldsAt(Satiated(a), 3) \wedge \\ HoldsAt(On(a, s), 3) \wedge \\ HoldsAt(On(f, s), 3)) \tag{11.75}$$

From (11.71), (11.72), (11.73), (11.74), (11.75), (11.67), (11.66), and (11.29), we have (11.70).

Timepoint 4: Fluents Using similar arguments we can show

$$\begin{aligned} &HoldsAt(On(o,s),4) \Leftrightarrow \\ &(o = Cat \wedge s = Chair) \vee \\ &(o = Food1 \wedge s = Table) \vee \\ &(o = Food2 \wedge s = Shelf) \end{aligned} \tag{11.76}$$

$$\begin{aligned} &HoldsAt(Believe(a, On(o,s)),4) \Leftrightarrow \\ &(a = Cat \wedge o = Cat \wedge s = Chair) \vee \\ &(a = Cat \wedge o = Food1 \wedge s = Table) \end{aligned} \tag{11.77}$$

$$\begin{aligned} &HoldsAt(Believe(a, CanJump(s_1,s_2)),4) \Leftrightarrow \\ &(a = Cat \wedge s_1 = Floor \wedge s_2 = Chair) \vee \\ &(a = Cat \wedge s_1 = Chair \wedge s_2 = Shelf) \vee \\ &(a = Cat \wedge s_1 = Shelf \wedge s_2 = Table) \end{aligned} \tag{11.78}$$

$$\neg HoldsAt(Believe(Cat, Satiated(Cat)),4) \tag{11.79}$$

$$\neg HoldsAt(Plan(a,g,p),4) \tag{11.80}$$

$$HoldsAt(Goal(Cat, Satiated(Cat)),4) \tag{11.81}$$

$$a \neq Cat \vee g \neq Satiated(Cat) \Rightarrow \neg HoldsAt(Goal(a,g),4) \tag{11.82}$$

Timepoint 4: Events Because no plans are active at timepoint 4, a new plan is activated. Let $P_2 = [Jump(Cat, Chair, Shelf), Wait(Cat), Jump(Cat, Shelf, Table), Eat(Cat, Food1)]$. We proceed to show

$$\begin{aligned} &Happens(e,4) \Leftrightarrow \\ &e = AddPlan(Cat, Satiated(Cat), P_2) \end{aligned} \tag{11.83}$$

From (11.80), we have

$$\begin{aligned} &\neg\exists e,a,g,p\,(e = Head(p) \wedge \\ &HoldsAt(Plan(a,g,p),4) \wedge \\ &\neg HoldsAt(Believe(a,g),4) \wedge \\ &SoundPlan(a,g,p,4)) \end{aligned} \tag{11.84}$$

$$\neg\exists e,a,g,p\,(e = DropPlan(a,g,p) \wedge HoldsAt(Plan(a,g,p),4)) \tag{11.85}$$

$$\neg\exists e, a, g, p\,(e = AddPlan(a, g, Tail(p)) \wedge HoldsAt(Plan(a, g, p), 4) \wedge Length(p) > 1 \wedge \neg HoldsAt(Believe(a, g), 4) \wedge SoundPlan(a, g, p, 4)) \tag{11.86}$$

From (11.81), (11.82), and (11.79), we have

$$\neg\exists e, a, g\,(e = DropGoal(a, g) \wedge HoldsAt(Goal(a, g), 4) \wedge HoldsAt(Believe(a, g), 4)) \tag{11.87}$$

From (11.76), we have

$$\neg\exists e, a, f, s\,(e = Eat(a, f) \wedge \neg HoldsAt(Satiated(a), 4) \wedge HoldsAt(On(a, s), 4) \wedge HoldsAt(On(f, s), 4)) \tag{11.88}$$

From (11.77), (11.78), and (11.14), we have

$$SelectedPlan(a, g, p, 4) \Leftrightarrow a = Cat \wedge g = Satiated(Cat) \wedge p = P_2$$

From this, (11.81), (11.82), (11.80), (11.79), (11.84), (11.85), (11.86), (11.87), (11.88), and (11.29), we have (11.83).

Timepoint 5: Fluents Using similar arguments we can show

$$HoldsAt(On(o, s), 5) \Leftrightarrow (o = Cat \wedge s = Chair) \vee (o = Food1 \wedge s = Table) \vee (o = Food2 \wedge s = Shelf) \tag{11.89}$$

$$HoldsAt(Believe(a, CanJump(s_1, s_2)), 5) \Leftrightarrow (a = Cat \wedge s_1 = Floor \wedge s_2 = Chair) \vee (a = Cat \wedge s_1 = Chair \wedge s_2 = Shelf) \vee (a = Cat \wedge s_1 = Shelf \wedge s_2 = Table) \tag{11.90}$$

$$\neg HoldsAt(Believe(Cat, Satiated(Cat)), 5) \tag{11.91}$$

$$HoldsAt(Plan(Cat, Satiated(Cat), P_2), 5) \tag{11.92}$$

$$a \neq Cat \vee g \neq Satiated(Cat) \vee p \neq P_2 \Rightarrow \neg HoldsAt(Plan(a, g, p), 5) \tag{11.93}$$

$$HoldsAt(Goal(Cat, Satiated(Cat)), 5) \tag{11.94}$$

$$a \neq Cat \vee g \neq Satiated(Cat) \Rightarrow \neg HoldsAt(Goal(a, g), 4) \tag{11.95}$$

Timepoint 5: Events The new plan is acted on. We proceed to show

$$\begin{aligned} Happens(e, 5) \Leftrightarrow \\ e = Jump(Cat, Chair, Shelf) \vee \\ e = DropPlan(Cat, Satiated(Cat), P_2) \vee \\ e = AddPlan(Cat, Satiated(Cat), Tail(P_2)) \end{aligned} \tag{11.96}$$

From (11.94), (11.95), and (11.92), we have

$$\begin{aligned} \neg\exists a, g, p\, (e = AddPlan(a, g, p) \wedge \\ \neg HoldsAt(Believe(a, g), 5) \wedge \\ HoldsAt(Goal(a, g), 5) \wedge \\ SelectedPlan(a, g, p, 5) \wedge \\ \neg\exists p_1\ HoldsAt(Plan(a, g, p_1), 5)) \end{aligned} \tag{11.97}$$

From (11.91) and (11.95), we have

$$\begin{aligned} \neg\exists e, a, g\, (e = DropGoal(a, g) \wedge \\ HoldsAt(Goal(a, g), 5) \wedge \\ HoldsAt(Believe(a, g), 5)) \end{aligned} \tag{11.98}$$

From (11.89), we have

$$\begin{aligned} \neg\exists e, a, f, s\, (e = Eat(a, f) \wedge \\ \neg HoldsAt(Satiated(a), 5) \wedge \\ HoldsAt(On(a, s), 5) \wedge \\ HoldsAt(On(f, s), 5)) \end{aligned} \tag{11.99}$$

From (11.90), (11.15), L5, and L6, we have

$$SoundPlan(Cat, Satiated(Cat), P_2, 5)$$

From this, (11.97), (11.98), (11.99), (11.93), (11.92), (11.91), (11.29), L1, L3, and L4, we have (11.96).

Timepoint 6: Fluents Using similar arguments we can show

$$\begin{aligned} &HoldsAt(On(o,s),6) \Leftrightarrow \\ &(o = Cat \wedge s = Shelf) \vee \\ &(o = Food1 \wedge s = Table) \vee \\ &(o = Food2 \wedge s = Shelf) \end{aligned} \tag{11.100}$$

$$\neg HoldsAt(Believe(Cat, Satiated(Cat)), 6) \tag{11.101}$$

Timepoint 6: Events From (11.101), (11.100), (11.16) (which follows from (11.29)), we have $Happens(Eat(Cat, Food2), 6)$ as required. (At timepoint 7, the cat is satiated, the plan is not further acted upon, and the goal to be satiated is dropped.) ■

11.2 Emotions

We use the theory of human emotions and emotional reactions to events proposed by Andrew Ortony, Gerald L. Clore, and Allan Collins.

11.2.1 Emotion Theory

The emotion theory specifies several emotion classes, each consisting of several emotions or emotional reactions. For each emotion, *eliciting conditions* determine under what circumstances the emotion is elicited.

Well-being emotions are reactions to desirable or undesirable events. For example, if a desirable event occurs, then a *joy* emotion is elicited, whereas if an undesirable event occurs, then a *distress* emotion is elicited.

Fortunes-of-others emotions are reactions to events involving others. They vary according to whether they are desirable or undesirable for the self and whether they are believed to be desirable or undesirable for the other person. For example, if an event occurs that is undesirable for the self and desirable for the other person, then a *resentment* emotion is elicited.

Prospect-based emotions are reactions to events that are likely to occur but have not yet occurred and reactions to anticipated events that occur or fail to occur. They vary according to whether they are desirable or undesirable and whether they have not yet occurred, they have occurred, or they have failed to occur. For example, if a desirable event is likely to occur, then a

hope emotion is elicited; if an undesirable event has failed to occur, then the *relief* emotion is elicited.

Attribution emotions are reactions to actions performed by the self or others of which the self approves or disapproves. For example, if the self performs an action approved by the self, then the *pride* emotion is elicited; if another person performs an action approved by the self, then the *appreciation* emotion is elicited.

Compound emotions are combinations of well-being and attribution emotions. For example, *gratitude* is a combination of joy and appreciation.

Attraction emotions are positive and negative reactions to objects. For example, a person *likes* or *dislikes* various objects.

11.2.2 Formalization

We now formalize in the event calculus (parts of) the theory of Ortony, Clore, and Collins. We have the following sorts:

- a belief sort, with variables b, b_1, b_2, ...
- an event sort, which is a subsort of the belief sort
- a negated event sort, which is also a subsort of the belief sort
- a fluent sort, with variables f, f_1, f_2, ...
- an object sort, with variables o, o_1, o_2, ...
- an agent sort, which is a subsort of the object sort, with variables a, a_1, a_2, ...
- a real number sort, with variables x, x_1, x_2, ...
- an event sort, with variables e, e_1, e_2, ...

We also introduce a function symbol *Not*, which denotes a function that maps events to their negations.

We start with the following fluents that represent factors used to specify the eliciting conditions of emotions.

Believe(a, b): If b is an event term, then agent a believes that b has occurred. If b is a negated event term *Not*(e), then agent a believes that e has not occurred.

Desirability(a_1, a_2, e, x): Agent a_1 believes that the desirability of event e to agent a_2 is x, where $-1 \leq x \leq 1$. Agents a_1 and a_2 may be the same.

Praiseworthiness(a_1, a_2, e, x): Agent a_1 believes that the praiseworthiness of an action e performed by agent a_2 is x, where $-1 \leq x \leq 1$. Agents a_1 and a_2 may be the same.

Anticipate(a, e, x): Agent a anticipates that event e will occur with likelihood x, where $0 \leq x \leq 1$.

We have the following fluents for well-being emotions.

Joy(a, e): Agent a is joyful about event e.

Distress(a, e): Agent a is distressed about event e.

We have the following fluents for fortunes-of-others emotions.

HappyFor(a_1, a_2, e): Agent a_1 is happy for agent a_2 regarding event e.

SorryFor(a_1, a_2, e): Agent a_1 is sorry for agent a_2 regarding event e.

Resentment(a_1, a_2, e): Agent a_1 is resentful of agent a_2 regarding event e.

Gloating(a_1, a_2, e): Agent a_1 gloats toward agent a_2 regarding event e.

We have the following fluents for prospect-based emotions.

Hope(a, e): Agent a is hopeful about an anticipated event e.

Fear(a, e): Agent a is fearful about an anticipated event e.

Satisfaction(a, e): Agent a is satisfied that event e occurred.

Disappointment(a, e): Agent a is disappointed that event e did not occur.

Relief(a, e): Agent a is relieved that event e did not occur.

FearsConfirmed(a, e): The fears of a are confirmed by the occurrence of event e.

We have the following fluents for attribution emotions.

Pride(a, e): Agent a is proud of performing action e.

SelfReproach(a, e): Agent a feels self-reproach for performing action e.

Appreciation(a_1, a_2, e): Agent a_1 is appreciative of agent a_2 for performing action e.

Reproach(a_1, a_2, e): Agent a_1 is reproachful of agent a_2 for performing action e.

We have the following fluents for compound emotions.

Gratitude(a_1, a_2, e): Agent a_1 is grateful toward agent a_2 regarding event e.

Anger(a_1, a_2, e): Agent a_1 is angry at agent a_2 regarding event e.

Gratification(a, e): Agent a is gratified about event e.

Remorse(a, e): Agent a is remorseful about event e.

We have the following fluents for attraction emotions.

Like(a, o): Agent a likes object o.
Dislike(a, o): Agent a dislikes object o.

We now present a number of axioms, starting with those for adding and dropping these fluents.

AXIOM E1 *Initiates*(*AddBelief*(a, b), *Believe*$(a, b), t)$

AXIOM E2 *Terminates*(*DropBelief*(a, b), *Believe*$(a, b), t)$

AXIOM E3 *Initiates*(*AddDesirability*(a_1, a_2, e, x), *Desirability*$(a_1, a_2, e, x), t)$

AXIOM E4 *Terminates*(*DropDesirability*(a_1, a_2, e, x), *Desirability*$(a_1, a_2, e, x), t)$

AXIOM E5 *Initiates*(*AddPraiseworthiness*(a_1, a_2, e, x), *Praiseworthiness*$(a_1, a_2, e, x), t)$

AXIOM E6 *Terminates*(*DropPraiseworthiness*(a_1, a_2, e, x), *Praiseworthiness*$(a_1, a_2, e, x), t)$

AXIOM E7 *Initiates*(*AddAnticipate*(a, e, x), *Anticipate*$(a, e, x), t)$

AXIOM E8 *Terminates*(*DropAnticipate*(a, e, x), *Anticipate*$(a, e, x), t)$

AXIOM E9 *Initiates*(*AddJoy*(a, e), *Joy*$(a, e), t)$

AXIOM E10 *Terminates*(*DropJoy*(a, e), *Joy*$(a, e), t)$

AXIOM E11 *Initiates*(*AddDistress*(a, e), *Distress*$(a, e), t)$

AXIOM E12 *Terminates*(*DropDistress*(a, e), *Distress*$(a, e), t)$

AXIOM E13 *Initiates*(*AddHappyFor*(a_1, a_2, e), *HappyFor*$(a_1, a_2, e), t)$

AXIOM E14 *Terminates*(*DropHappyFor*(a_1, a_2, e), *HappyFor*$(a_1, a_2, e), t)$

AXIOM E15 *Initiates*(*AddSorryFor*(a_1, a_2, e), *SorryFor*$(a_1, a_2, e), t)$

AXIOM E16 *Terminates*(*DropSorryFor*(a_1, a_2, e), *SorryFor*$(a_1, a_2, e), t)$

AXIOM E17 $Initiates(AddResentment(a_1, a_2, e), Resentment(a_1, a_2, e), t)$

AXIOM E18 $Terminates(DropResentment(a_1, a_2, e), Resentment(a_1, a_2, e), t)$

AXIOM E19 $Initiates(AddGloating(a_1, a_2, e), Gloating(a_1, a_2, e), t)$

AXIOM E20 $Terminates(DropGloating(a_1, a_2, e), Gloating(a_1, a_2, e), t)$

AXIOM E21 $Initiates(AddHope(a, e), Hope(a, e), t)$

AXIOM E22 $Terminates(DropHope(a, e), Hope(a, e), t)$

AXIOM E23 $Initiates(AddFear(a, e), Fear(a, e), t)$

AXIOM E24 $Terminates(DropFear(a, e), Fear(a, e), t)$

AXIOM E25 $Initiates(AddSatisfaction(a, e), Satisfaction(a, e), t)$

AXIOM E26 $Terminates(DropSatisfaction(a, e), Satisfaction(a, e), t)$

AXIOM E27 $Initiates(AddDisappointment(a, e), Disappointment(a, e), t)$

AXIOM E28 $Terminates(DropDisappointment(a, e), Disappointment(a, e), t)$

AXIOM E29 $Initiates(AddRelief(a, e), Relief(a, e), t)$

AXIOM E30 $Terminates(DropRelief(a, e), Relief(a, e), t)$

AXIOM E31 $Initiates(AddFearsConfirmed(a, e), FearsConfirmed(a, e), t)$

AXIOM E32 $Terminates(DropFearsConfirmed(a, e), FearsConfirmed(a, e), t)$

AXIOM E33 $Initiates(AddPride(a, e), Pride(a, e), t)$

AXIOM E34 $Terminates(DropPride(a, e), Pride(a, e), t)$

AXIOM E35 $Initiates(AddSelfReproach(a, e), SelfReproach(a, e), t)$

AXIOM E36 $Terminates(DropSelfReproach(a, e), SelfReproach(a, e), t)$

AXIOM E37 $Initiates(AddAppreciation(a_1, a_2, e), Appreciation(a_1, a_2, e), t)$

AXIOM E38 $Terminates(DropAppreciation(a_1, a_2, e), Appreciation(a_1, a_2, e), t)$

AXIOM E39 $Initiates(AddReproach(a_1, a_2, e), Reproach(a_1, a_2, e), t)$

AXIOM E40 $Terminates(DropReproach(a_1, a_2, e), Reproach(a_1, a_2, e), t)$

AXIOM E41 $Initiates(AddGratitude(a_1, a_2, e), Gratitude(a_1, a_2, e), t)$

AXIOM E42 $Terminates(DropGratitude(a_1, a_2, e), Gratitude(a_1, a_2, e), t)$

AXIOM E43 $Initiates(AddAnger(a_1, a_2, e), Anger(a_1, a_2, e), t)$

AXIOM E44 $Terminates(DropAnger(a_1, a_2, e), Anger(a_1, a_2, e), t)$

AXIOM E45 $Initiates(AddGratification(a, e), Gratification(a, e), t)$

AXIOM E46 $Terminates(DropGratification(a, e), Gratification(a, e), t)$

AXIOM E47 $Initiates(AddRemorse(a, e), Remorse(a, e), t)$

AXIOM E48 $Terminates(DropRemorse(a, e), Remorse(a, e), t)$

AXIOM E49 $Initiates(AddLike(a, o), Like(a, o), t)$

AXIOM E50 $Terminates(DropLike(a, o), Like(a, o), t)$

AXIOM E51 $Initiates(AddDislike(a, o), Dislike(a, o), t)$

AXIOM E52 $Terminates(DropDislike(a, o), Dislike(a, o), t)$

We continue with state constraints on desirability, praiseworthiness, and anticipation.

AXIOM E53
$$HoldsAt(Desirability(a_1, a_2, e, x_1), t) \wedge HoldsAt(Desirability(a_1, a_2, e, x_2), t) \Rightarrow x_1 = x_2$$

AXIOM E54
$$HoldsAt(Desirability(a_1, a_2, e, x), t) \Rightarrow -1 \leq x \leq 1$$

AXIOM E55
$$HoldsAt(Praiseworthiness(a_1, a_2, e, x_1), t) \wedge HoldsAt(Praiseworthiness(a_1, a_2, e, x_2), t) \Rightarrow x_1 = x_2$$

AXIOM E56
$$HoldsAt(Praiseworthiness(a_1, a_2, e, x), t) \Rightarrow -1 \leq x \leq 1$$

AXIOM E57
$$HoldsAt(Anticipate(a, e, x_1), t) \wedge HoldsAt(Anticipate(a, e, x_2), t) \Rightarrow x_1 = x_2$$

AXIOM E58
$$HoldsAt(Anticipate(a, e, x), t) \Rightarrow 0 \leq x \leq 1$$

We now present axioms for eliciting emotions.

AXIOM E59
$$\neg HoldsAt(Joy(a, e), t) \wedge HoldsAt(Desirability(a, a, e, x), t) \wedge x > 0 \wedge HoldsAt(Believe(a, e), t) \Rightarrow Happens(AddJoy(a, e), t)$$

AXIOM E60
$$\neg HoldsAt(Distress(a, e), t) \wedge HoldsAt(Desirability(a, a, e, x), t) \wedge x < 0 \wedge HoldsAt(Believe(a, e), t) \Rightarrow Happens(AddDistress(a, e), t)$$

AXIOM E61
$$\neg HoldsAt(HappyFor(a_1, a_2, e), t) \wedge HoldsAt(Desirability(a_1, a_2, e, x_1), t) \wedge x_1 > 0 \wedge HoldsAt(Desirability(a_1, a_1, e, x_2), t) \wedge x_2 > 0 \wedge HoldsAt(Like(a_1, a_2), t) \wedge HoldsAt(Believe(a_1, e), t) \wedge a_1 \neq a_2 \Rightarrow Happens(AddHappyFor(a_1, a_2, e), t)$$

AXIOM E62
$$\neg HoldsAt(SorryFor(a_1, a_2, e), t) \wedge HoldsAt(Desirability(a_1, a_2, e, x_1), t) \wedge x_1 < 0 \wedge HoldsAt(Desirability(a_1, a_1, e, x_2), t) \wedge x_2 < 0 \wedge$$

$$HoldsAt(Like(a_1, a_2), t) \wedge$$
$$HoldsAt(Believe(a_1, e), t) \wedge$$
$$a_1 \neq a_2 \Rightarrow$$
$$Happens(AddSorryFor(a_1, a_2, e), t)$$

AXIOM E63

$$\neg HoldsAt(Resentment(a_1, a_2, e), t) \wedge$$
$$HoldsAt(Desirability(a_1, a_2, e, x_1), t) \wedge x_1 > 0 \wedge$$
$$HoldsAt(Desirability(a_1, a_1, e, x_2), t) \wedge x_2 < 0 \wedge$$
$$HoldsAt(Dislike(a_1, a_2), t) \wedge$$
$$HoldsAt(Believe(a_1, e), t) \wedge$$
$$a_1 \neq a_2 \Rightarrow$$
$$Happens(AddResentment(a_1, a_2, e), t)$$

AXIOM E64

$$\neg HoldsAt(Gloating(a_1, a_2, e), t) \wedge$$
$$HoldsAt(Desirability(a_1, a_2, e, x_1), t) \wedge x_1 < 0 \wedge$$
$$HoldsAt(Desirability(a_1, a_1, e, x_2), t) \wedge x_2 > 0 \wedge$$
$$HoldsAt(Dislike(a_1, a_2), t) \wedge$$
$$HoldsAt(Believe(a_1, e), t) \wedge$$
$$a_1 \neq a_2 \Rightarrow$$
$$Happens(AddGloating(a_1, a_2, e), t)$$

AXIOM E65

$$\neg HoldsAt(Hope(a, e), t) \wedge$$
$$HoldsAt(Desirability(a, a, e, x_1), t) \wedge x_1 > 0 \wedge$$
$$HoldsAt(Anticipate(a, e, x_2), t) \wedge x_2 > 0 \wedge$$
$$a_1 \neq a_2 \Rightarrow$$
$$Happens(AddHope(a, e), t)$$

AXIOM E66

$$\neg HoldsAt(Fear(a, e), t) \wedge$$
$$HoldsAt(Desirability(a, a, e, x_1), t) \wedge x_1 < 0 \wedge$$
$$HoldsAt(Anticipate(a, e, x_2), t) \wedge x_2 > 0 \Rightarrow$$
$$Happens(AddFear(a, e), t)$$

AXIOM E67

$$\neg HoldsAt(Satisfaction(a, e), t) \wedge$$
$$HoldsAt(Desirability(a, a, e, x_1), t) \wedge x_1 > 0 \wedge$$

$$HoldsAt(Anticipate(a, e, x_2), t) \wedge x_2 > 0 \wedge$$
$$HoldsAt(Believe(a, e), t) \Rightarrow$$
$$Happens(AddSatisfaction(a, e), t)$$

AXIOM E68

$$\neg HoldsAt(Disappointment(a, e), t) \wedge$$
$$HoldsAt(Desirability(a, a, e, x_1), t) \wedge x_1 > 0 \wedge$$
$$HoldsAt(Anticipate(a, e, x_2), t) \wedge x_2 > 0 \wedge$$
$$HoldsAt(Believe(a, \neg e), t) \Rightarrow$$
$$Happens(AddDisappointment(a, e), t)$$

AXIOM E69

$$\neg HoldsAt(Relief(a, e), t) \wedge$$
$$HoldsAt(Desirability(a, a, e, x_1), t) \wedge x_1 < 0 \wedge$$
$$HoldsAt(Anticipate(a, e, x_2), t) \wedge x_2 > 0 \wedge$$
$$HoldsAt(Believe(a, \neg e), t) \Rightarrow$$
$$Happens(AddRelief(a, e), t)$$

AXIOM E70

$$\neg HoldsAt(FearsConfirmed(a, e), t) \wedge$$
$$HoldsAt(Desirability(a, a, e, x_1), t) \wedge x_1 < 0 \wedge$$
$$HoldsAt(Anticipate(a, e, x_2), t) \wedge x_2 > 0 \wedge$$
$$HoldsAt(Believe(a, e), t) \Rightarrow$$
$$Happens(AddFearsConfirmed(a, e), t)$$

AXIOM E71

$$\neg HoldsAt(Pride(a, e), t) \wedge$$
$$HoldsAt(Praiseworthiness(a, a, e, x), t) \wedge x > 0 \wedge$$
$$HoldsAt(Believe(a, e), t) \Rightarrow$$
$$Happens(AddPride(a, e), t)$$

AXIOM E72

$$\neg HoldsAt(SelfReproach(a, e), t) \wedge$$
$$HoldsAt(Praiseworthiness(a, a, e, x), t) \wedge x < 0 \wedge$$
$$HoldsAt(Believe(a, e), t) \Rightarrow$$
$$Happens(AddSelfReproach(a, e), t)$$

AXIOM E73

$$\neg HoldsAt(Appreciation(a_1, a_2, e), t) \wedge$$
$$HoldsAt(Praiseworthiness(a_1, a_2, e, x), t) \wedge x > 0 \wedge$$

$$HoldsAt(Believe(a_1, e), t) \wedge a_1 \neq a_2 \Rightarrow$$
$$Happens(AddAppreciation(a_1, a_2, e), t)$$

AXIOM E74

$$\neg HoldsAt(Reproach(a_1, a_2, e), t) \wedge$$
$$HoldsAt(Praiseworthiness(a_1, a_2, e, x), t) \wedge x < 0 \wedge$$
$$HoldsAt(Believe(a_1, e), t) \wedge a_1 \neq a_2 \Rightarrow$$
$$Happens(AddReproach(a_1, a_2, e), t)$$

We specify compound emotions using state constraints.

AXIOM E75

$$HoldsAt(Gratitude(a_1, a_2, e), t) \Leftrightarrow$$
$$HoldsAt(Joy(a_1, e), t) \wedge$$
$$HoldsAt(Appreciation(a_1, a_2, e), t)$$

AXIOM E76

$$HoldsAt(Anger(a_1, a_2, e), t) \Leftrightarrow$$
$$HoldsAt(Distress(a_1, e), t) \wedge$$
$$HoldsAt(Reproach(a_1, a_2, e), t)$$

AXIOM E77

$$HoldsAt(Gratification(a, e), t) \Leftrightarrow$$
$$HoldsAt(Joy(a, e), t) \wedge$$
$$HoldsAt(Pride(a, e), t)$$

AXIOM E78

$$HoldsAt(Remorse(a, e), t) \Leftrightarrow$$
$$HoldsAt(Distress(a, e), t) \wedge$$
$$HoldsAt(SelfReproach(a, e), t)$$

11.2.3 Example: Lottery

If we are told that Lisa likes Kate, who just won the lottery, we should infer that Kate will be happy and Lisa will be happy for Kate. We assume that when any agent wins the lottery, all agents are aware of it:

$$Initiates(WinLottery(a_1), Believe(a_2, WinLottery(a_1)), t) \quad (11.102)$$

We have the following observations and narrative. Initially, Kate is not joyful about any events, and Lisa is not happy for Kate regarding any events:

$$\neg ReleasedAt(f, t) \quad (11.103)$$
$$\neg HoldsAt(Joy(Kate, e), 0) \quad (11.104)$$
$$\neg HoldsAt(HappyFor(Lisa, Kate, e), 0) \quad (11.105)$$

Kate and Lisa do not believe any events have occurred:

$$\neg HoldsAt(Believe(Kate, e), 0) \tag{11.106}$$
$$\neg HoldsAt(Believe(Lisa, e), 0) \tag{11.107}$$

Lisa likes Kate, Kate believes that her winning the lottery is highly desirable to herself, Lisa believes that Kate winning the lottery is highly desirable to Kate, and Lisa believes that Kate winning the lottery is highly desirable to Lisa:

$$HoldsAt(Like(Lisa, Kate), t) \tag{11.108}$$
$$HoldsAt(Desirability(Kate, Kate, WinLottery(Kate), 1), t) \tag{11.109}$$
$$HoldsAt(Desirability(Lisa, Kate, WinLottery(Kate), 1), t) \tag{11.110}$$
$$HoldsAt(Desirability(Lisa, Lisa, WinLottery(Kate), 1), t) \tag{11.111}$$

Kate wins the lottery:

$$Happens(WinLottery(Kate), 0) \tag{11.112}$$

We can show that Kate will be happy, and Lisa will be happy for Kate.

PROPOSITION 11.2 Let Σ be the conjunction of E9, E13, and (11.102). Let Δ be the conjunction of E59, E61, and (11.112). Let Ω be $U[AddJoy, AddHappyFor, WinLottery] \wedge U[Joy, HappyFor, Believe, Desirability, Like]$. Let Ψ be the conjunction of E53 and E54. Let Γ be the conjunction of (11.103), (11.104), (11.105), (11.106), (11.107), (11.108), (11.109), (11.110), and (11.111). Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Omega \wedge \Psi \wedge \Gamma \wedge \text{DEC} \models HoldsAt(Joy(Kate, WinLottery(Kate)), 2) \wedge HoldsAt(HappyFor(Lisa, Kate, WinLottery(Kate)), 2)$$

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

$$Initiates(e, f, t) \Leftrightarrow \tag{11.113}$$
$$\exists a, e_1 (e = AddJoy(a, e_1) \wedge f = Joy(a, e_1)) \vee$$
$$\exists a_1, a_2, e_1 (e = AddHappyFor(a_1, a_2, e_1) \wedge f = HappyFor(a_1, a_2, e_1)) \vee$$
$$\exists a_1, a_2 (e = WinLottery(a_1) \wedge f = Believe(a_2, WinLottery(a_1)))$$

$$\neg Terminates(e, f, t) \tag{11.114}$$

$$\neg Releases(e, f, t) \tag{11.115}$$

From *CIRC*[Δ; *Happens*] and Theorem 2.1, we have

$$\begin{aligned}
&Happens(e,t) \Leftrightarrow \\
&\exists a, e_1, x\,(e = AddJoy(a, e_1) \wedge \\
&\neg HoldsAt(Joy(a, e_1), t) \wedge \\
&HoldsAt(Desirability(a, a, e_1, x), t) \wedge \\
&x > 0 \wedge \\
&HoldsAt(Believe(a, e_1), t)) \vee \\
&\exists a_1, a_2, e_1, x_1, x_2\,(e = AddHappyFor(a_1, a_2, e_1) \wedge \\
&\neg HoldsAt(HappyFor(a_1, a_2, e_1), t) \wedge \\
&HoldsAt(Desirability(a_1, a_2, e_1, x_1), t) \wedge \\
&x_1 > 0 \wedge \\
&HoldsAt(Desirability(a_1, a_1, e_1, x_2), t) \wedge \\
&x_2 > 0 \wedge \\
&HoldsAt(Like(a_1, a_2), t) \wedge \\
&HoldsAt(Believe(a_1, e_1), t) \wedge \\
&a_1 \neq a_2) \vee \\
&(e = WinLottery(Kate) \wedge t = 0)
\end{aligned} \tag{11.116}$$

From (11.112) (which follows from (11.116)), (11.102) (which follows from (11.113)), and DEC9, we have the following:

$$HoldsAt(Believe(Kate, WinLottery(Kate)), 1) \tag{11.117}$$

$$HoldsAt(Believe(Lisa, WinLottery(Kate)), 1) \tag{11.118}$$

From (11.113), (11.116), and (11.106), we have $\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, Joy(Kate, e_1), 0))$. From this, (11.104), (11.103), and DEC6, we have $\neg HoldsAt(Joy(Kate, e), 1)$. From this, (11.109), $1 > 0$, (11.117), and E59 (which follows from (11.116)), we have

$$Happens(AddJoy(Kate, WinLottery(Kate)), 1) \tag{11.119}$$

From (11.113), (11.116), and (11.107), we have $\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, HappyFor(Lisa, Kate, e_1), 0))$. From this, (11.105), (11.103), and DEC6, we have $\neg HoldsAt(HappyFor(Lisa, Kate, e), 1)$. From this, (11.110), (11.111), (11.108), (11.118), $Lisa \neq Kate$, and E61 (which follows from (11.116)), we have

$$Happens(AddHappyFor(Lisa, Kate, WinLottery(Kate)), 1) \tag{11.120}$$

From (11.119), E9 (which follows from (11.113)), and DEC9, we have $HoldsAt(Joy(Kate, WinLottery(Kate)), 2)$. From (11.120), E13 (which follows from (11.113)), and DEC9, we have $HoldsAt(HappyFor(Lisa, Kate, WinLottery(Kate)), 2)$. ■

Bibliographic Notes

Beliefs and Knowledge

Formal models of beliefs and knowledge are provided by R. C. Moore (1980), Morgenstern (1988), and E. Davis (1990, pp. 351–393). Fagin, Halpern, Moses, and Vardi (1995) provide a book-length treatment of reasoning about knowledge. Joseph Halpern, John McCarthy, and Heinrich Wansing debate the merits of modal logic versus first-order logic for representing mental states (McCarthy, 1997, 1999, 2000; Wansing, 1998; Halpern, 1999, 2000). McCarthy (1998b) discusses giving robots knowledge of their own mental states. Elgot-Drapkin and Perlis (1990) and Elgot-Drapkin et al. (1999) describe a logic called active logic for modeling the step-by-step reasoning processes of agents. Bratman (1987) treats the notions of beliefs, desires, and intentions.

Our axiomatization of beliefs in the event calculus is a simplified version of the one of Lévy and Quantz (1998). They develop a variant of the event calculus called the situated event calculus, which adds a situation argument to the event calculus predicates. They provide belief axioms such as

$$Initiates(adopt(p, s_f), believes(p, f, t_f), t_b, s_b) \leftarrow Holds\_at(f, t_f, s_f)$$

which represents that, if a person p adopts a situation s_f at timepoint t_b in situation s_b, and fluent f is true at timepoint t_f in situation s_f, then after t_b in s_b person p will believe that f is true at timepoint t_f.

Reactive Behavior

Our representation of reactive behavior is similar to the stimulus-response rules of behaviorism (Watson, 1924). Brooks (1990) provides a language for reactive behaviors that compiles into code to be run on an architecture for controlling mobile robots (Brooks, 1985).

Goals and Plans

Our event calculus axiomatization of goal-driven behavior is loosely based on agent control mechanisms such as those of Simon (1967); Fikes et al. (1972a); Wilensky (1983); Genesereth and Nilsson (1987, pp. 325–327); Bratman, Israel, and Pollack (1988); Mueller (1990); Rao and Georgeff (1992); Shoham (1993); Beaudoin (1994); Wooldridge (2000); and Winikoff et al. (2002). The hungry cat scenario is taken from Winikoff et al. (2002).

The persistence of plans and goals was pointed out by Lewin (1926/1951) and G. A. Miller, Galanter, and Pribram (1960). The study of planning is a large area within the field of artificial intelligence (Fikes and Nilsson, 1971; Newell and Simon, 1972; Sacerdoti, 1974, 1977; Wilensky, 1983; Ghallab, Nau, and Traverso, 2004). A collection of readings on planning is provided by Allen,

Hendler, and Tate (1990). Logical formalizations of goals and plans as mental states are provided by Morgenstern (1988), Cohen and Levesque (1990), and E. Davis (1990, pp. 413–430).

Commonsense Psychology

A number of researchers have analyzed the notions of the mental states and the events people use to model themselves and others. These notions are variously known as commonsense psychology, folk psychology, and theory of mind. Heider (1958) treats interpersonal relations and notions of perception, action, desire, pleasure, and sentiment. He provides relations such as "*p* SIMILAR TO *o* INDUCES *p* LIKES *o*, OR *p* TENDS TO LIKE A SIMILAR *o*" (p. 184).

Schank and Abelson (1977) provide a taxonomy of human goals and plans for achieving those goals. They classify goals according to whether they are to be repeatedly satisfied (S-HUNGER, S-SEX, S-SLEEP), are pursued for enjoyment (E-TRAVEL, E-ENTERTAINMENT, E-COMPETITION), are to be achieved (A-GOOD-JOB, A-POSSESSIONS, A-SOCIAL-RELATIONSHIPS), are states to be preserved (P-HEALTH, P-JOB, P-POSSESSIONS, P-SOCIAL-RELATIONSHIP), involve imminent threats (C-HEALTH, C-FIRE, C-STORM), or are instrumental to other goals (I-PREP, D-KNOW, D-PROX, D-CONT). They describe named plans and planboxes for achieving goals. For instance, a named plan USE(FOOD) for the goal of satisfying hunger (S-HUNGER) consists of knowing (D-KNOW) the location of some food, being near (D-PROX) the food, having physical control (D-CONT) over the food, and eating the food.

Smedslund (1997) provides a system of commonsense psychological propositions (45 definitions and 56 axioms). The propositions are broken up into the following categories: persons, acting, wanting, believing, feeling, interpersonal processes, intrapersonal processes, and personal change. Sample axioms are "*P wants X, if, and only if, other things equal, P prefers X to not-X*" (p. 29) and "*P is happy, if, and only if, and to the extent that, P believes that at least one of P's wants is being, or is going to be, fulfilled*" (p. 51). Gordon (2004) provides pre-formal representations of 372 strategies from 10 planning domains.

Emotions

Dyer (1983, pp. 103–139) presents a model of human emotions for story understanding. The representation of an emotion consists of a sign (positive or negative), story character experiencing the emotion, goal situation giving rise to the emotion, character toward whom the emotion is directed, strength, and expectation (expected or unexpected). A number of emotions can be represented using this scheme. Gratitude, for example, is represented as a positive emotion directed toward a character that caused a goal success for the character experiencing the emotion. Hope is represented as a positive emotion associated with an active goal that is expected to succeed.

Mueller (1990, pp. 53–56) extends Dyer's representation to model the influence of emotions on daydreaming and the influence of daydreaming on emotions.

For example, an emotion of interest associated with an active goal initiates a stream of thought involving various scenarios for achieving the goal. If success of the goal is imagined, then an emotion of hope is initiated. Certain emotions are associated with certain personal goals. For example, embarrassment is defined as a negative emotion that results from a failed social esteem goal.

We formalize in the event calculus the model of emotions of Ortony, Clore, and Collins (1988). Our formalization is loosely based on the situation calculus formalization of O'Rorke and Ortony (1994). Some differences are as follows. O'Rorke and Ortony (p. 296) define emotional reactions using state constraints such as

$$joy(P, F, S) \leftarrow wants(P, F, S) \wedge holds(F, S)$$

rather than using trigger axioms and effect axioms. They represent that emotional reactions result directly from the fact that a fluent is true rather than from the fact that an agent believes that an event has occurred. They do not use real-valued functions for desirability, praiseworthiness, and anticipation, as suggested by Ortony et al. (1988, pp. 181–190).

Sanders (1989) presents a formalization of emotions using an extended version of Shoham's (1988) temporal logic. She treats emotions involving obligation, permission, and prohibition, namely, the emotions of anger, gratitude, approval, disapproval, shame, and guilt. For example, an axiom states that x is grateful to y if and only if (1) x believes that y performed an action a, (2) x wanted y to perform a, (3) x believes that a benefits x, and (4) x believes that y performed a without expecting anything in return. As in our axiomatization, the emotions of an agent arise from the beliefs of the agent. Unlike in our axiomatization, Sanders defines emotions using state constraints rather than trigger axioms and effect axioms.

Exercises

11.1 Formalize the reactive behavior of a driver. If a driver sees a squirrel in front of the car, then the driver will slam on the brakes.

11.2 If we remove the reactive behavior axiom (11.16), then when the cat jumps onto the shelf the cat sticks to his original plan, jumping from the shelf to the table, ignoring the cookie on the shelf. Prove that if this axiom is removed, then the cat eats the food on the table.

11.3 Suppose that we remove axiom (11.16), but we modify the axiomatization so that the agent drops plans that are nonoptimal. A plan is nonoptimal if a shorter plan exists. Provide appropriate modifications to axioms A1–A11 so that nonoptimal plans are dropped. Note that when a nonoptimal plan is dropped, a new plan should be activated by axiom A7. Modify other axioms as needed. Prove that the cat eats the food on the shelf.

11.4 Discuss the advantages and disadvantages of reactive rules versus rules that look for nonoptimal plans.

11.5 Extend the formalization in Section 11.2 so that (1) each emotion has an associated strength, (2) the strength is set appropriately based on desirability, praiseworthiness, and the likelihood of the anticipated event, (3) the strength of an emotion decays over time, and (4) an emotion is dropped when its strength drops below a certain threshold.

11.6 Formalize the operation of mental defense mechanisms (Freud, 1936/1946; Suppes and Warren, 1975). Use trigger axioms and determining fluents to apply transformations to the beliefs of an agent at each timepoint.

11.7 (Research Problem) Extend the formalization in Section 11.1 to deal with multiple active goals and plans. Add priorities to active goals and a mechanism to select which active goal to work on at any moment.

11.8 (Research Problem) McCarthy (1990) discusses a news story about a furniture salesman who is robbed. Formalize the knowledge necessary to represent and reason about this story.

11.9 (Research Problem) Formalize processes of human daydreaming (Mueller, 1990) in the event calculus. Daydreaming consists of recalling past experiences, imagining alternative versions of past experiences, and imagining possible future experiences.

CHAPTER 12

Default Reasoning

Commonsense reasoning requires default reasoning. As we engage in commonsense reasoning, we make certain assumptions so that we can proceed. If we later gain more information, then we may have to revise those assumptions. For example, if we are told that Nathan went to sleep, we might assume that he went to sleep in his home. If we are told that he cooked breakfast, we might imagine him in his kitchen. If we later learn that he was actually at a hotel, our understanding of these events will have to be modified.

We have so far considered two special cases of default reasoning: assuming that unexpected events do not occur and assuming that events do not have unexpected effects. In this chapter, we consider default reasoning in general. We discuss atemporal default reasoning and then temporal default reasoning. We describe a general method for default reasoning about time based on the circumscription of abnormality predicates and how this method can be used to address the qualification problem. We then discuss the representation of default events and properties.

12.1 Atemporal Default Reasoning

We start by introducing default reasoning with some examples that do not involve time. Consider the following formula:

$$Apple(x) \Rightarrow Red(x)$$

This states that apples are always red. If we wish to represent that apples are sometimes red, we may use a formula such as the following:

$$Apple(x) \wedge \neg Ab_1(x) \Rightarrow Red(x) \tag{12.1}$$

The predicate $Ab_1(x)$ is called an *abnormality predicate*. It represents that x is abnormal in a way that prevents us from concluding that it is red given that it is an apple. Thus, (12.1) states that all apples that are not abnormal in fashion 1 are red. How do we specify when an apple is abnormal in fashion 1?

One possibility is that we could specify the necessary and sufficient conditions. We could represent that an apple is abnormal in fashion 1 if and only if it is a Granny Smith apple or it is rotten:

$$Ab_1(x) \Leftrightarrow GrannySmith(x) \vee Rotten(x) \tag{12.2}$$

But what about other conditions that might make an apple abnormal in fashion 1? If we wished to add other conditions, then we would have to modify (12.2):

$$Ab_1(x) \Leftrightarrow \\ GrannySmith(x) \vee Rotten(x) \vee NewtownPippin(x) \tag{12.3}$$

An alternative is to use the following approach, which is more elaboration tolerant. We start by specifying some sufficient conditions for an apple being abnormal in fashion 1:

$$GrannySmith(x) \Rightarrow Ab_1(x) \tag{12.4}$$
$$Rotten(x) \Rightarrow Ab_1(x) \tag{12.5}$$
$$NewtownPippin(x) \Rightarrow Ab_1(x) \tag{12.6}$$

Then we circumscribe Ab_1 in formulas (12.4) to (12.6). Let Θ be the conjunction of (12.4), (12.5), and (12.6). By theorem 2.1, $CIRC[\Theta; Ab_1]$ is equivalent to (12.3). This circumscription allows us to jump to the conclusion that the necessary and sufficient conditions are specified by (12.3), given that the sufficient conditions are specified by Θ.

Now suppose that a particular apple A is not a Granny Smith apple, not rotten, and not a Newtown Pippin apple:

$$Apple(A)$$
$$\neg GrannySmith(A)$$
$$\neg Rotten(A)$$
$$\neg NewtownPippin(A)$$

We can conclude from (12.3) that A is not abnormal in fashion 1:

$$\neg Ab_1(A)$$

From this and (12.1), we can conclude that A is red:

$$Red(A)$$

We jump to the conclusion that A is red given that it is not a Granny Smith apple, not rotten, and not a Newtown Pippin apple.

12.2 Temporal Default Reasoning

Now we introduce time. In fact, previous chapters have already discussed a form of default reasoning about time. The event calculus assumes by default that (1) unexpected events do not occur and (2) events have no unexpected effects.

Event Occurrences

Although it is abnormal for an event to occur at any given timepoint, events sometimes do occur. We specify what events are known to occur in the conjunction Δ of a domain description, which is described in Section 2.7. We write event calculus formulas containing the predicate $Happens(e, t)$, which represents that timepoint t is abnormal with respect to event e. We then use circumscription to minimize the extension of *Happens* in the conjunction Δ, which minimizes event occurrences.

Event Effects

It is also abnormal for an event to have any given effect. We specify the known effects of events in the conjunction Σ of a domain description. We write event calculus formulas containing the abnormality predicates *Initiates*, *Terminates*, and *Releases*. We then circumscribe these predicates in the conjunction Σ in order to minimize event effects.

Using Minimized Events and Effects

Having minimized *Happens*, *Initiates*, *Terminates*, and *Releases*, we then write formulas containing these predicates that are outside the scope of the circumscription of these predicates. An example is axiom DEC5 of the discrete event calculus:

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

This axiom states that a fluent f that is true at timepoint t is also true at $t+1$, unless f is released from the commonsense law of inertia at $t+1$ or there is an event e that occurs at t and terminates f at t.

12.3 Default Reasoning Method

We now present the following general method for default reasoning about time based on the method just discussed:

1. We specify a set of abnormality predicates. Typically we use predicate symbols of the form Ab_i where i is a positive integer.

2. We use the abnormality predicates within various event calculus formulas such as effect axioms. (Table 2.1 provides a list of types of event calculus formulas.)
3. We form a conjunction Θ of cancellation axioms. If γ is a condition, ρ is an abnormality predicate symbol of arity $n+1$, $\tau_1, \ldots, \tau_n$ are terms, and τ is a timepoint term, then

$$\gamma \Rightarrow \rho(\tau_1, \ldots, \tau_n, \tau)$$

   is a *cancellation axiom* for ρ.
4. We compute the circumscription of the abnormality predicates in the conjunction of cancellation axioms Θ.

This method requires that we add the following clause to the definition of a condition in Section 2.5: If ρ is an abnormality predicate symbol of arity $n+1$, $\tau_1, \ldots, \tau_n$ are terms, and τ is a timepoint term, then $\rho(\tau_1, \ldots, \tau_n, \tau)$ is a condition.

The domain description is

$$CIRC[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge CIRC[\Delta_1 \wedge \Delta_2; \mathit{Happens}] \wedge$$
$$CIRC[\Theta; \rho_1, \ldots, \rho_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge \mathrm{E}$$

where $\rho_1, \ldots, \rho_n$ are the abnormality predicate symbols, Θ is a conjunction of cancellation axioms containing $\rho_1, \ldots, \rho_n$, and Σ, Δ_1, Δ_2, Ω, Ψ, Π, Γ, and E are as defined in Section 2.7.

12.4 Defaults and the Qualification Problem

Default reasoning can be used to address the qualification problem (introduced in Section 3.3). Default reasoning provides a more elaboration-tolerant way of specifying qualifications that prevent events from having their intended effects.

First, we add abnormality predicates to effect axioms, which are the same as fluent precondition axioms. We write effect axioms of the form

$$\gamma \wedge \neg\rho(\tau_1, \ldots, \tau_n, \tau) \Rightarrow \mathit{Initiates}(\alpha, \beta, \tau)$$

or

$$\gamma \wedge \neg\rho(\tau_1, \ldots, \tau_n, \tau) \Rightarrow \mathit{Terminates}(\alpha, \beta, \tau)$$

where γ is a condition, ρ is an abnormality predicate symbol of arity $n+1$, $\tau_1, \ldots, \tau_n$ are terms, τ is a timepoint term, α is an event term, and β is a fluent term. Second, whenever we wish to add a qualification, we add a cancellation axiom for ρ.

Default reasoning can also be used to represent qualifications that prevent events from occurring. But action precondition axioms (discussed in Section 3.3.2) already provide an elaboration tolerant way of expressing such qualifications.

12.4.1 Example: Device Revisited

Consider the example of turning on a device. We introduce the abnormality predicate $Ab_1(d, t)$, which represents that device d is abnormal in some way at timepoint t. We use this predicate in an effect axiom. If a device is not Ab_1 and an agent turns on the device, then the device will be on:

$$\neg Ab_1(d, t) \Rightarrow Initiates(TurnOn(a, d), On(d), t) \tag{12.7}$$

We add cancellation axioms that represent qualifications. A device is Ab_1 if its switch is broken:

$$HoldsAt(BrokenSwitch(d), t) \Rightarrow Ab_1(d, t) \tag{12.8}$$

A device is Ab_1 if it is not plugged in:

$$\neg HoldsAt(PluggedIn(d), t) \Rightarrow Ab_1(d, t) \tag{12.9}$$

A particular antique device never works:

$$Ab_1(AntiqueDevice1, t) \tag{12.10}$$

Now consider the following observations and narrative in which the device is initially off and then the device is turned on:

$$\neg HoldsAt(On(Device1), 0) \tag{12.11}$$
$$\neg ReleasedAt(f, t) \tag{12.12}$$
$$Happens(TurnOn(Nathan, Device1), 0) \tag{12.13}$$

Suppose that the device's switch is not broken, the device is plugged in, and the device is not the antique device:

$$\neg HoldsAt(BrokenSwitch(Device1), 0) \tag{12.14}$$
$$HoldsAt(PluggedIn(Device1), 0) \tag{12.15}$$
$$Device1 \neq AntiqueDevice1 \tag{12.16}$$

We can then show that the device will be on at timepoint 1.

PROPOSITION 12.1

Let Σ = (12.7), Δ = (12.13), Θ = (12.8) $\wedge$ (12.9) $\wedge$ (12.10), Ω = (12.16) $\wedge$ U[*On, BrokenSwitch, PluggedIn*], and Γ = (12.11) $\wedge$ (12.12) $\wedge$ (12.14) $\wedge$ (12.15). Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge CIRC[\Theta; Ab_1] \wedge \Omega \wedge \Gamma \wedge \text{DEC} \models HoldsAt(On(Device1), 1)$$

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

$$Initiates(e,f,t) \Leftrightarrow \exists a, d\,(e = TurnOn(a,d) \wedge f = On(d) \wedge \neg Ab_1(d,t)) \tag{12.17}$$

$$\neg Terminates(e,f,t) \tag{12.18}$$

$$\neg Releases(e,f,t) \tag{12.19}$$

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

$$Happens(e,t) \Leftrightarrow (e = TurnOn(Nathan, Device1) \wedge t = 0) \tag{12.20}$$

From $CIRC[\Theta; Ab_1]$ and Theorem 2.1, we have

$$Ab_1(d,t) \Leftrightarrow HoldsAt(BrokenSwitch(d),t) \vee \neg HoldsAt(PluggedIn(d),t) \vee d = AntiqueDevice1 \tag{12.21}$$

From this, (12.14), (12.15), and (12.16), we have $\neg Ab_1(Device1, 0)$. From this, (12.13) (which follows from (12.20)), (12.7) (which follows from (12.17)), and DEC9, we have $HoldsAt(On(Device1), 1)$. ■

12.4.2 Example: Broken Device

Suppose that instead of (12.14), (12.15), and (12.16), we have the fact that the device's switch is broken:

$$HoldsAt(BrokenSwitch(Device1), 0) \tag{12.22}$$

We can then show that the device will *not* be on at timepoint 1.

PROPOSITION 12.2

Let Σ = (12.7), Δ = (12.13), Θ = (12.8) $\wedge$ (12.9) $\wedge$ (12.10), $\Omega = U[On, BrokenSwitch, PluggedIn]$, and Γ = (12.11) $\wedge$ (12.12) $\wedge$ (12.22). Then we have

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge CIRC[\Theta; Ab_1] \wedge \Omega \wedge \Gamma \wedge \text{DEC} \models \neg HoldsAt(On(Device1), 1)$$

Proof From $CIRC[\Sigma; Initiates, Terminates, Releases]$, Theorem 2.2, and Theorem 2.1, we have (12.17), (12.18), and (12.19). From $CIRC[\Delta; Happens]$ and Theorem 2.1, we have (12.20). From $CIRC[\Theta; Ab_1]$ and Theorem 2.1, we have (12.21). From (12.21) and (12.22), we have $Ab_1(Device1, 0)$. From this and (12.17), we have $\neg\exists e\,(Happens(e, 0) \wedge Initiates(e, On(Device1), 0))$. From this, (12.11), (12.12), and DEC6, we have $\neg HoldsAt(On(Device1), 1)$. ■

12.4.3 Strong and Weak Qualifications

Patrick Doherty and Jonas Kvarnström proposed a distinction between strong and weak qualifications. If an event is *strongly qualified*, then it will not have its intended effects; if an event is *weakly qualified*, then it may or may not have its intended effects. A weakly qualified event results in two classes of models: one in which the event has its intended effects and one in which the event does not. Thus, weakly qualified events have nondeterministic effects. Weak qualifications can be represented using determining fluents (introduced in Section 9.1). A determining fluent is introduced for each weak qualification.

12.4.4 Example: Erratic Device

Let us extend our device example to erratic devices. If a device is erratic, then when it is turned on, it may or may not go on. We introduce the determining fluent $DeterminingFluent(d)$. We add a cancellation axiom that states that if a device is erratic and the determining fluent is true, then the device is Ab_1:

$$HoldsAt(Erratic(d), t) \wedge HoldsAt(DeterminingFluent(d), t) \Rightarrow Ab_1(d, t) \tag{12.23}$$

Now suppose that the device's switch is not broken, the device is plugged in, the device is erratic, and Nathan turns on the device at timepoint 0. We can show that if $DeterminingFluent(Device1)$ is false at timepoint 0, then the device will be on at timepoint 1. But if $DeterminingFluent(Device1)$ is true at timepoint 0, then

the device will not be on at timepoint 1. We use the following observations. The device is erratic:

$$HoldsAt(Erratic(Device1), 0) \tag{12.24}$$

The determining fluent is always released from the commonsense law of inertia, and *On*, *BrokenSwitch*, *PluggedIn*, and *Erratic* are never released from this law:

$$ReleasedAt(DeterminingFluent(d), t) \tag{12.25}$$
$$\neg ReleasedAt(On(d), t) \tag{12.26}$$
$$\neg ReleasedAt(BrokenSwitch(d), t) \tag{12.27}$$
$$\neg ReleasedAt(PluggedIn(d), t) \tag{12.28}$$
$$\neg ReleasedAt(Erratic(d), t) \tag{12.29}$$

PROPOSITION 12.3

Let Σ = (12.7), Δ = (12.13), Θ = (12.8) $\wedge$ (12.9) $\wedge$ (12.23), $\Omega = U[On, BrokenSwitch, PluggedIn, Erratic, DeterminingFluent]$, and Γ = (12.11) $\wedge$ (12.14) $\wedge$ (12.15) $\wedge$ (12.24) $\wedge$ (12.25) $\wedge$ (12.26) $\wedge$ (12.27) $\wedge$ (12.28) $\wedge$ (12.29). Then we have

$$\begin{aligned} &CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ &CIRC[\Theta; Ab_1] \wedge \Omega \wedge \Gamma \wedge \text{DEC} \\ &\models (\neg HoldsAt(DeterminingFluent(Device1), 0) \Rightarrow \\ &\quad HoldsAt(On(Device1), 1)) \wedge \\ &\quad (HoldsAt(DeterminingFluent(Device1), 0) \Rightarrow \\ &\quad \neg HoldsAt(On(Device1), 1)) \end{aligned}$$

Proof See Exercise 12.1. ■

12.5 Default Events and Properties

Default reasoning can be used to represent events that occur by default, as well as properties that are true or false by default.

12.5.1 Default Events

A default event is represented by an axiom of the form

$$\gamma \wedge \neg\rho(\tau_1, \ldots, \tau_n, \tau) \Rightarrow Happens(\alpha, \tau)$$

where γ is a condition, ρ is an abnormality predicate symbol of arity $n + 1$, $\tau_1, \ldots, \tau_n$ are terms, τ is a timepoint term, and α is an event term.

We may specify that normally a watch beeps every 3600 timepoints, starting at timepoint 0:

$$Remainder(t, 3600) = 0 \wedge \neg Ab_1(w, t) \Rightarrow Happens(Beep(w), t)$$

Using the alarm clock axiomatization in Section 4.1.1, we may rewrite the trigger axiom (4.8) as follows:

$$HoldsAt(AlarmTime(c, t), t) \wedge HoldsAt(AlarmOn(c), t) \wedge \neg Ab_1(c, t) \Rightarrow Happens(StartBeeping(c), t)$$

This states that normally a clock starts beeping when its alarm time is the current time.

12.5.2 Default Properties

A default property such as a typical color or location is represented by a formula of the form

$$\neg\rho(\tau_1, \ldots, \tau_n, \tau) \Rightarrow HoldsAt(\beta, \tau)$$

where ρ is an abnormality predicate symbol of arity $n+1$, $\tau_1, \ldots, \tau_n$ are terms, τ is a timepoint term, and β is a fluent term.

Suppose we have an apple sort with variable a. We represent the typical color of apples as follows:

$$\neg Ab_1(a, t) \Rightarrow HoldsAt(Red(a), t)$$

We add cancellation axioms representing when apples are not red:

$$HoldsAt(GrannySmith(a), t) \Rightarrow Ab_1(a, t)$$
$$HoldsAt(Rotten(a), t) \Rightarrow Ab_1(a, t)$$
$$HoldsAt(NewtownPippin(a), t) \Rightarrow Ab_1(a, t)$$

Suppose we have a living room sort with variable l and a television sort with variable v. We represent the typical location of a television as follows:

$$\neg Ab_1(l, t) \Rightarrow \exists v\, HoldsAt(InRoom(v, l), t)$$

Bibliographic Notes

Frames and Inheritance Networks

Minsky (1974) proposed a knowledge structure called *frames* for representing typical situations. The slots of a frame have default assignments that "can be easily displaced by new items that fit better the current situation" (p. 2). A number of framelike knowledge representation languages with defaults have been developed (Bobrow and Winograd, 1977; Roberts and Goldstein, 1977; Charniak, Riesbeck, and McDermott, 1980, chap. 15). Frames are reviewed by Brachman and Levesque (2004, chap. 8). Work on semantic networks (Quillian, 1968; Winston, 1970) and frames led to the development of inheritance networks (Fahlman, 1979; Horty, Thomason, and Touretzky, 1990; Stein, 1992), which can be used for default reasoning; inheritance networks are reviewed by Brachman and Levesque (2004, chap. 10).

Benchmark Problems

Three benchmark problems were important in the development of logic-based methods for default reasoning: the Tweety example, the Nixon diamond, and the Yale shooting scenario. Reiter (1978, p. 215; 1980b, pp. 82–83) introduced the Tweety example, which can be traced back to work on semantic networks (A. M. Collins and Quillian, 1969). The example is as follows. All birds fly except for penguins, which do not fly. If we are told that Tweety is a bird, then we should conclude that Tweety flies. But if we are told that Tweety is a penguin, then we should retract the conclusion that Tweety flies and conclude that Tweety does not fly. Reiter (1980b) introduced default logic to deal with examples of default reasoning such as this. Lifschitz (1999) discusses the success of default logic.

The Nixon diamond, introduced by Reiter and Criscuolo (1981, p. 274), is an example of conflicting defaults. The example is as follows. Quakers are typically pacifists, and Republicans are typically not pacifists. If we are told that John is both a Quaker and a Republican, what shall we conclude? The Yale shooting scenario is discussed in Section 5.2.1.

Logic-based methods for default reasoning are reviewed by Brachman and Levesque (2004, chap. 11) and Russell and Norvig (2003, pp. 354–360). See also the discussion of nonmonotonic reasoning methods in the Bibliographic Notes of Chapter 2.

Methods Based on Circumscription

McCarthy (1977) proposed circumscription as a "candidate for solving the qualification problem" (p. 1040). McCarthy (1980, pp. 36–37) introduced the method of circumscription of abnormality for default reasoning using a predicate called *prevents*. McCarthy (1984a, 1986, 1987) and Grosof (1984) further developed the method and introduced the *ab* predicate. Levesque (1981, chap. 6) proposed a similar method involving *prototypical predicates*, or the negations of abnormality

predicates. The prototypical predicate $\nabla\rho : \mathrm{i}(x)$ represents that x is prototypical with respect to the ith default property of ρ. For example, we can write the following:

$$\nabla Apple : 1(x) \Rightarrow Red(x)$$
$$GrannySmith(x) \Rightarrow \neg\nabla Apple : 1(x)$$

The analogous formulas using abnormality predicates are:

$$Apple(x) \wedge \neg Ab_1(x) \Rightarrow Red(x)$$
$$GrannySmith(x) \Rightarrow Ab_1(x)$$

Defaults are applied by a function $\Delta Ext(m)$, which extends a knowledge base m with formulas such as

$$Apple(x) \wedge \neg K\neg\nabla Apple : 1(x) \Rightarrow \nabla Apple : 1(x)$$

which states that if x is an apple and it is not known that x is not prototypical with respect to the first default property of apples, then x is prototypical with respect to this property.

The method of default reasoning used in this chapter is one of a number of proposed methods based on circumscription, which are summarized in Table 12.1. For each method, the table lists the techniques used in the method, an example of a domain theory, the circumscriptions of the domain theory, and the results entailed by the circumscriptions. Methods (A) and (B) address default reasoning problems such as the Tweety example. Methods (C), (D), and (E) address problems of conflicting defaults such as the Nixon diamond. Method (F) was developed to deal with problems involving action and change such as the Yale shooting scenario.

Methods (A) through (E) use circumscription with varied constants (Lifschitz, 1994, sec. 2.3). In basic circumscription (defined in Appendix A), the extension of a predicate is minimized without allowing the extensions of other predicates to vary. The extensions of other predicates are fixed. In *circumscription with varied constants,* the extension of one predicate is minimized while allowing the extensions of certain other predicates to vary. If Γ is a formula containing the predicate symbols $\rho, \psi_1, \ldots, \psi_n$, then the circumscription of ρ in Γ with $\psi_1, \ldots, \psi_n$ varied, written $CIRC[\Gamma; \rho; \psi_1, \ldots, \psi_n]$, is the formula of second-order logic

$$\Gamma \wedge \neg\exists\phi, \nu_1, \ldots, \nu_n \left(\Gamma(\phi, \nu_1, \ldots, \nu_n) \wedge \phi < \rho\right)$$

where ϕ and $\nu_1, \ldots, \nu_n$ are distinct predicate variables with the same arities and argument sorts as ρ and $\psi_1, \ldots, \psi_n$, respectively, and $\Gamma(\phi, \nu_1, \ldots, \nu_n)$ is the formula obtained from Γ by replacing each occurrence of ρ with ϕ and each occurrence of ψ_i with ν_i for each $i \in \{1, \ldots, n\}$.

Methods (C) through (F) use conjunctions of circumscriptions. Method (C) adopts the skeptical stance that, because the defaults about $Q(A)$ conflict, neither $Q(A)$ nor $\neg Q(A)$ should be concluded.

Table 12.1 Default reasoning methods based on circumscription[a]

Method	*Example*	*CIRC*	*Result*
(A) var (McCarthy, 1986; Lifschitz, 1994)	$\Phi =$ $(P(x) \wedge \neg Ab(x) \Rightarrow Q(x)) \wedge$ $\neg Q(A) \wedge P(A) \wedge P(B) \wedge$ $A \neq B$	$CIRC[\Phi; Ab; Q] \models$ $Ab(x) \Leftrightarrow x = A$	$Q(B)$
(B) cancellation + var (McCarthy, 1986; Lifschitz, 1994)	$\Phi =$ $(P(x) \wedge \neg Ab(x) \Rightarrow Q(x)) \wedge$ $Ab(A) \wedge P(A) \wedge P(B) \wedge$ $A \neq B$	$CIRC[\Phi; Ab; Q] \models$ $Ab(x) \Leftrightarrow x = A$	$Q(B)$
(C) conjunction + var (McCarthy, 1986; Lifschitz, 1994)	$\Phi =$ $(P(x) \wedge \neg Ab_1(x) \Rightarrow Q(x)) \wedge$ $(R(x) \wedge \neg Ab_2(x) \Rightarrow \neg Q(x)) \wedge$ $P(A) \wedge R(A)$	$CIRC[\Phi; Ab_1; Q] \wedge$ $CIRC[\Phi; Ab_2; Q] \models$ $(Ab_1(x) \Leftrightarrow P(x) \wedge$ $R(x) \wedge \neg Ab_2(x)) \wedge$ $(Ab_2(x) \Leftrightarrow P(x) \wedge$ $R(x) \wedge \neg Ab_1(x))$	
(D) cancellation + conjunction + var (McCarthy, 1986; Lifschitz, 1994)	$\Phi =$ $(P(x) \wedge \neg Ab_1(x) \Rightarrow Q(x)) \wedge$ $(R(x) \wedge \neg Ab_2(x) \Rightarrow \neg Q(x)) \wedge$ $(R(x) \Rightarrow P(x)) \wedge R(A) \wedge$ $P(B) \wedge \neg R(B) \wedge A \neq B \wedge$ $(R(x) \Rightarrow Ab_1(x))$	$CIRC[\Phi; Ab_1; Q] \wedge$ $CIRC[\Phi; Ab_2; Q] \models$ $(Ab_1(x) \Leftrightarrow R(x)) \wedge$ $\neg Ab_2(x)$	$Q(B)$ $\neg Q(A)$
(E) conjunction + priorities + var (McCarthy, 1986; Lifschitz, 1994)	$\Phi =$ $(P(x) \wedge \neg Ab_1(x) \Rightarrow Q(x)) \wedge$ $(R(x) \wedge \neg Ab_2(x) \Rightarrow \neg Q(x)) \wedge$ $(R(x) \Rightarrow P(x)) \wedge R(A) \wedge$ $P(B) \wedge \neg R(B) \wedge A \neq B$	$CIRC[\Phi; Ab_1; Q] \wedge$ $CIRC[\Phi; Ab_2; Q, Ab_1]$ $\models (Ab_1(x) \Leftrightarrow R(x)) \wedge$ $\neg Ab_2(x)$	$Q(B)$ $\neg Q(A)$
(F) cancellation + conjunction + filtering + parallel = forced separation (Shanahan, 1997b)	$\Phi_1 = (P(x) \Rightarrow Ab_1(x)) \wedge$ $(Q(x) \Rightarrow Ab_2(x))$ $\Phi_2 = (R(x) \Rightarrow Ab_3(x))$ $\Phi_3 =$ $(S(x) \wedge \neg Ab_1(x) \Rightarrow T(x)) \wedge$ $(U(x) \wedge \neg Ab_2(x) \Rightarrow V(x)) \wedge$ $(W(x) \wedge \neg Ab_3(x) \Rightarrow Y(x)) \wedge$ $S(A) \wedge \neg P(A)$	$CIRC[\Phi_1; Ab_1, Ab_2] \wedge$ $CIRC[\Phi_2; Ab_3] \wedge$ $\Phi_3 \models$ $(Ab_1(x) \Leftrightarrow P(x)) \wedge$ $(Ab_2(x) \Leftrightarrow Q(x)) \wedge$ $(Ab_3(x) \Leftrightarrow R(x))$	$T(A)$

[a] cancellation = cancellation of inheritance axioms; conjunction = conjunction of circumscriptions; parallel = parallel circumscription; var = circumscription with varied constants.

Methods (B), (D), and (F) use cancellation of inheritance axioms (McCarthy, 1986, p. 93; Genesereth and Nilsson, 1987, p. 129; Lifschitz, 1994, sec. 4.1), which are called cancellation axioms in this chapter. Antoniou (1997, pp. 158–159) and Brewka et al. (1997, pp. 18–19) criticize the use of cancellation of inheritance axioms on elaboration tolerance grounds. Consider the example given for (D). Notice that whenever we add an axiom $\rho_1(x) \wedge \neg\rho_2(x) \Rightarrow \neg Q(x)$ to the domain theory, where ρ_1 is a predicate symbol and ρ_2 is an abnormality predicate symbol, we must also remember to add a cancellation of inheritance axiom $\rho_1(x) \Rightarrow Ab_1(x)$. But also notice that this is not the only possible order of elaboration of the domain theory. After writing the axiom $P(x) \wedge \neg Ab_1(x) \Rightarrow Q(x)$, we might then think of a situation in which this default does not apply. We would then write a cancellation axiom $\rho_1(x) \Rightarrow Ab_1(x)$. Having canceled the default, we would then proceed to write an axiom describing what the new default is in this situation: $\rho_1(x) \wedge \neg\rho_2(x) \Rightarrow \neg Q(x)$.

Antoniou (1997) and Brewka et al. (1997) advocate replacing cancellation of inheritance axioms with prioritized circumscription. Method (E) replaces the cancellation of inheritance axiom in method (D) with the use of priorities on abnormality predicate symbols. The abnormality predicate symbol Ab_2 is given a higher priority than Ab_1 by allowing Ab_1 to vary in the circumscription of Ab_2. This method can be generalized. For example, we may specify that Ab_3 has a higher priority than Ab_2, which has a higher priority than Ab_1, as follows:

$$\begin{gathered} CIRC[\Phi; Ab_1; Q] \wedge \\ CIRC[\Phi; Ab_2; Q, Ab_1] \wedge \\ CIRC[\Phi; Ab_3; Q, Ab_1, Ab_2] \end{gathered}$$

Etherington (1988, p. 47) criticizes the use of priorities on the grounds that it is difficult to determine what the priorities should be.

Method (F) incorporates the technique of filtering (Sandewall, 1989b, 1994; Doherty and Łukaszewicz, 1994; Doherty, 1994), in which the domain theory is partitioned and only some parts of the partition are circumscribed. Filtering is related to the process of delimited completion (Genesereth and Nilsson, 1987, p. 132). Method (F), which is used in this chapter, is called forced separation in Shanahan (1996, 1997b). In forced separation, a domain theory is partitioned into formulas $\Phi_1, \ldots, \Phi_n$, disjoint sets of predicates are circumscribed in Φ_i for each $i \in \{1, \ldots, n-1\}$, and Φ_n is not circumscribed. The combination of cancellation, conjunction, and filtering from method (F) can be used to solve problems addressed by methods (B) and (D), as shown in Table 12.2.

Our treatment of the qualification problem is patterned after that of E. Giunchiglia et al. (2004, pp. 67–68). Doherty and Kvarnström (1997) propose the distinction between strong and weak qualifications. See also the discussion of Thielscher (2001, pp. 27–29). Williams (1997) describes a framework for modeling revision of beliefs.

Table 12.2 Default reasoning using basic circumscription and filtering

Method	*Example*	*CIRC*	*Result*
(B′)	$\Phi_1 = Ab(A)$	$CIRC[\Phi_1; Ab] \wedge$	$Q(B)$
cancellation +	$\Phi_2 =$	$\Phi_2 \models$	
filtering	$(P(x) \wedge \neg Ab(x) \Rightarrow Q(x)) \wedge$	$Ab(x) \Leftrightarrow x = A$	
	$P(A) \wedge P(B) \wedge A \neq B$		
(D′)	$\Phi_1 = (R(x) \Rightarrow Ab_1(x))$	$CIRC[\Phi_1; Ab_1] \wedge$	$Q(B)$
cancellation +	$\Phi_2 = (\bot \Rightarrow Ab_2(x))$	$CIRC[\Phi_2; Ab_2] \wedge$	$\neg Q(A)$
conjunction +	$\Phi_3 =$	$\Phi_3 \models$	
filtering	$(P(x) \wedge \neg Ab_1(x) \Rightarrow Q(x)) \wedge$	$(Ab_1(x) \Leftrightarrow R(x)) \wedge$	
	$(R(x) \wedge \neg Ab_2(x) \Rightarrow \neg Q(x)) \wedge$	$\neg Ab_2(x)$	
	$(R(x) \Rightarrow P(x)) \wedge R(A) \wedge$		
	$P(B) \wedge \neg R(B) \wedge A \neq B$		

Exercises

12.1 Prove Proposition 12.3.

12.2 Formalize the slippery block scenario described by Thielscher (2001, p. 27). A surface is a block or a table. Normally when a block is moved from one surface to another, the block ends up on the second surface. If the block is slippery, however, then the block may (or may not) end up on the table rather than the second surface.

12.3 Formalize Lifschitz's (2003) problem of choosing a place for dinner. Each night, Peter eats at either the Chinese restaurant or the Italian restaurant. On Mondays, the Italian restaurant is closed. Peter prefers not to eat at the same restaurant two nights in a row. Use the formalization to determine where Peter eats on Sundays.

12.4 Formalize the Russian airplane hijack scenario of Kvarnström and Doherty (2000a). Boris, Dimiter, and Erik are at their respective homes getting ready for a business trip. Dimiter gets drunk. Erik grabs a comb. They each stop off at the office on their way to the airport. While at the office, Boris picks up a gun. They each try to board a plane, which then flies to Stockholm. Normally, if a person is at the airport, then the person can board the plane. However, people carrying guns are not permitted to board the plane, and drunk people may or may not be permitted to board the plane. The problem is then to show that Erik and the comb will be in Stockholm, Boris will still be at the airport, and Dimiter may or may not be at Stockholm.

12.5 (Research Problem) In Section 11.2, we introduce the predicates $Like(a, o)$ and $Dislike(a, o)$. Formalize the default rules given by Heider (1958, chap. 7) regarding the ways people tend to evaluate or feel about people or things.

CHAPTER 13

The Discrete Event Calculus Reasoner

This chapter describes the Discrete Event Calculus Reasoner, which uses satisfiability (SAT) to solve event calculus problems. The program can be used to perform automated deduction, abduction, postdiction, and model finding. We discuss the architecture of the program and the encoding of SAT problems. We present some simple examples of how the program is used and then present a more complicated example. We discuss the language used to describe commonsense reasoning problems to the program. The Discrete Event Calculus Reasoner can be downloaded from decreasoner.sourceforge.net.

13.1 Discrete Event Calculus Reasoner Architecture

The architecture of the Discrete Event Calculus Reasoner is shown in Figure 13.1. The program works as follows:

1. A domain description is provided as input to the program.
2. The main program sends the domain description to the SAT encoder.
3. The SAT encoder encodes the domain description as a SAT problem.
4. The SAT encoder sends the SAT problem back to the main program.
5. The main program sends the SAT problem to the SAT solver. The SAT problem is expressed in the standard DIMACS format used by SAT solvers.
6. A complete SAT solver is run in order to find all models. The Discrete Event Calculus Reasoner uses the Relsat SAT solver. If Relsat does not find any models, then the Walksat solver is run in order to find near-miss models.

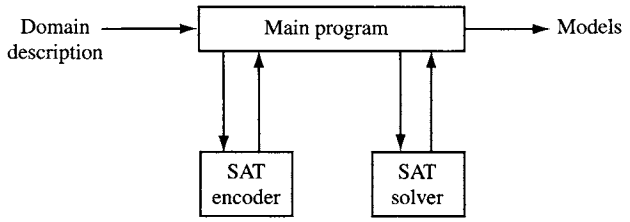


Figure 13.1 Discrete Event Calculus Reasoner architecture.

7. The SAT solver sends models back to the main program.
8. The main program decodes the models from the SAT solver and produces them as output.

13.2 Encoding Satisfiability Problems

In order to encode the domain description as a SAT problem, the domain of every sort is limited to a finite set. The timepoint sort is restricted to the integers $\{0, \ldots, n\}$ for some $n \geq 0$. An existential quantification $\exists v\, \Phi(v)$ is replaced with $\bigvee_i \Phi(v_i)$, and a universal quantification $\forall v\, \Phi(v)$ is replaced with $\bigwedge_i \Phi(v_i)$, where v_i are the constants of the sort of v. The unique names assumption is used.

SAT solving is facilitated by DEC. (The efficient solution of event calculus problems using SAT was the original motivation for the development of DEC.) DEC uses integers for timepoints, unlike EC, which uses real numbers. DEC replaces triply quantified time in many of the EC axioms with doubly quantified time, resulting in smaller SAT encodings.

13.3 Simple Examples

In this section we present some simple examples of how the Discrete Event Calculus Reasoner is used to perform various types of reasoning.

13.3.1 Deduction

The first example is a deduction problem. We create an input file containing the following domain description:

```
sort agent

fluent Awake(agent)
event WakeUp(agent)
```

```
[agent,time] Initiates(WakeUp(agent),Awake(agent),time).

agent James
!HoldsAt(Awake(James),0).
Delta: Happens(WakeUp(James),0).

completion Delta Happens

range time 0 1
```

We define sorts and predicates. We define an agent sort, a fluent *Awake*, and an event *WakeUp*. The fluent and the event both take a single argument that is an agent. Next, we provide an axiom stating that, if an agent wakes up, then that agent will be awake. The symbols [and] indicate universal quantification. Next we specify the observations and narrative. We define a constant *James* whose sort is agent. We specify that James is not awake at timepoint 0 and that he wakes up at timepoint 0. The symbol ! indicates negation. For deduction we specify completion of the *Happens* predicate in the formulas labeled `Delta`. Finally, we specify the range of the time sort. There are two timepoints: 0 and 1.

When we run the program on this input, it produces the following output:

```
model 1:
0
Happens(WakeUp(James), 0).
1
+Awake(James).
```

For each model, the fluents that are true at timepoint 0 are shown, followed by differences in the truth values of fluents from one timepoint to the next. Fluents that become true are preceded by + (plus sign), and fluents that become false are preceded by − (minus sign). There is one model, and *HoldsAt*(*Awake*(*James*), 1) is true in that model. Thus, the domain description entails that James is awake at timepoint 1.

An alternative output format is available that shows the truth values of all fluents at all timepoints. This format is obtained by adding the following line to the input file:

```
option timediff off
```

The program then produces the following output:

```
model 1:
0
!HoldsAt(Awake(James), 0).
!ReleasedAt(Awake(James),0).
Happens(WakeUp(James), 0).
1
!Happens(WakeUp(James), 1).
```

```
!ReleasedAt(Awake(James), 1).
HoldsAt(Awake(James), 1).
```

13.3.2 Abduction

We can turn the deduction problem into an abduction or planning problem by making several changes. First, we specify a *goal* for James to be awake at timepoint 1. Second, we no longer specify that James wakes up because this is the *plan* we would like the program to find. Third, we do not specify completion of the *Happens* predicate. Our new domain description is as follows:

```
sort agent

fluent Awake(agent)
event WakeUp(agent)

[agent,time] Initiates(WakeUp(agent),Awake(agent),time).

agent James
!HoldsAt(Awake(James),0).
HoldsAt(Awake(James),1).

range time 0 1
```

When we run this input through the program, we get a single model in which James wakes up:

```
model 1:
0
Happens(WakeUp(James), 0).
1
+Awake(James).
```

13.3.3 Postdiction

Next, we turn the deduction problem into a postdiction problem. First, we add an action precondition axiom stating that, in order for an agent to wake up, the agent must not be awake. Second, we remove the fact that James is not awake at timepoint 0 because this is the fact we would like the program to find. This gives us the following domain description:

```
sort agent

fluent Awake(agent)
event WakeUp(agent)

[agent,time] Initiates(WakeUp(agent),Awake(agent),time). [agent,time]
Happens(WakeUp(agent),time) -> !HoldsAt(Awake(agent),time).
```

```
agent James
Delta: Happens(WakeUp(James),0).
HoldsAt(Awake(James),1).

completion Delta Happens

range time 0 1
```

From this, the program produces a single model in which James is not awake at timepoint 0:

```
model 1:
0
Happens(WakeUp(James), 0).
1
+Awake(James).
```

13.3.4 Model Finding

We can turn the postdiction problem into a model-finding problem by removing the observations, narrative, and completion of *Happens* in `Delta`:

```
sort agent

fluent Awake(agent)
event WakeUp(agent)

[agent,time] Initiates(WakeUp(agent),Awake(agent),time).
[agent,time] Happens(WakeUp(agent),time) -> !HoldsAt(Awake(agent),time).

agent James

range time 0 1
```

We then get three models:

```
model 1:
0
Happens(WakeUp(James), 0).
1
+Awake(James).
---
model 2:
0
1
---
model 3:
0
Awake(James).
1
```

That is, James is either awake or not awake at timepoint 0. If he is not awake at timepoint 0, then he may wake up (model 1) or may not wake up (model 2). If he is awake at timepoint 0 (model 3), then he cannot wake up at timepoint 0. (Event occurrences are disallowed at the final timepoint, which is 1 in this case.)

13.4 Example: Telephone

In this section, we run the telephone example in Section 3.1.1 through the Discrete Event Calculus Reasoner. The domain description is as follows:

```
sort agent
sort phone

agent Agent1, Agent2
phone Phone1, Phone2

fluent Ringing(phone,phone)
fluent DialTone(phone)
fluent BusySignal(phone)
fluent Idle(phone)
fluent Connected(phone,phone)
fluent Disconnected(phone)

event PickUp(agent,phone)
event SetDown(agent,phone)
event Dial(agent,phone,phone)

[agent,phone,time]
HoldsAt(Idle(phone),time) ->
Initiates(PickUp(agent,phone),DialTone(phone),time).

[agent,phone,time]
HoldsAt(Idle(phone),time) ->
Terminates(PickUp(agent,phone),Idle(phone),time).

[agent,phone,time]
HoldsAt(DialTone(phone),time) ->
Initiates(SetDown(agent,phone),Idle(phone),time).

[agent,phone,time]
HoldsAt(DialTone(phone),time) ->
Terminates(SetDown(agent,phone),DialTone(phone),time).

[agent,phone1,phone2,time]
HoldsAt(DialTone(phone1),time) &
HoldsAt(Idle(phone2),time) ->
Initiates(Dial(agent,phone1,phone2),Ringing(phone1,phone2),time).

[agent,phone1,phone2,time]
HoldsAt(DialTone(phone1),time) &
```

```
HoldsAt(Idle(phone2),time) ->
Terminates(Dial(agent,phone1,phone2),DialTone(phone1),time).

[agent,phone1,phone2,time]
HoldsAt(DialTone(phone1),time) &
HoldsAt(Idle(phone2),time) ->
Terminates(Dial(agent,phone1,phone2),Idle(phone2),time).

[agent,phone1,phone2,time]
HoldsAt(DialTone(phone1),time) &
!HoldsAt(Idle(phone2),time) ->
Initiates(Dial(agent,phone1,phone2),BusySignal(phone1),time).

[agent,phone1,phone2,time]
HoldsAt(DialTone(phone1),time) &
!HoldsAt(Idle(phone2),time) ->
Terminates(Dial(agent,phone1,phone2),DialTone(phone1),time).

[agent,phone,time]
HoldsAt(BusySignal(phone),time) ->
Initiates(SetDown(agent,phone),Idle(phone),time).

[agent,phone,time]
HoldsAt(BusySignal(phone),time) ->
Terminates(SetDown(agent,phone),BusySignal(phone),time).

[agent,phone1,phone2,time]
HoldsAt(Ringing(phone1,phone2),time) ->
Initiates(SetDown(agent,phone1),Idle(phone1),time).

[agent,phone1,phone2,time]
HoldsAt(Ringing(phone1,phone2),time) ->
Initiates(SetDown(agent,phone1),Idle(phone2),time).

[agent,phone1,phone2,time]
HoldsAt(Ringing(phone1,phone2),time) ->
Terminates(SetDown(agent,phone1),Ringing(phone1,phone2),time).

[agent,phone1,phone2,time]
HoldsAt(Ringing(phone1,phone2),time) ->
Initiates(PickUp(agent,phone2),Connected(phone1,phone2),time).

[agent,phone1,phone2,time]
HoldsAt(Ringing(phone1,phone2),time) ->
Terminates(PickUp(agent,phone2),Ringing(phone1,phone2),time).

[agent,phone1,phone2,time]
HoldsAt(Connected(phone1,phone2),time) ->
Initiates(SetDown(agent,phone1),Idle(phone1),time).

[agent,phone1,phone2,time]
HoldsAt(Connected(phone1,phone2),time) ->
Initiates(SetDown(agent,phone1),Disconnected(phone2),time).
```

```
[agent,phone1,phone2,time]
HoldsAt(Connected(phone1,phone2),time)->
Terminates(SetDown(agent,phone1),Connected(phone1,phone2),time).

[agent,phone1,phone2,time]
HoldsAt(Connected(phone1,phone2),time)->
Initiates(SetDown(agent,phone2),Idle(phone2),time).

[agent,phone1,phone2,time]
HoldsAt(Connected(phone1,phone2),time)->
Initiates(SetDown(agent,phone2),Disconnected(phone1),time).

[agent,phone1,phone2,time]
HoldsAt(Connected(phone1,phone2),time)->
Terminates(SetDown(agent,phone2),Connected(phone1,phone2),time).

[agent,phone,time]
HoldsAt(Disconnected(phone),time) ->
Initiates(SetDown(agent,phone),Idle(phone),time).

[agent,phone,time]
HoldsAt(Disconnected(phone),time) ->
Terminates(SetDown(agent,phone),Disconnected(phone),time).

Delta: Happens(PickUp(Agent1,Phone1),0).
Delta: Happens(Dial(Agent1,Phone1,Phone2),1).
Delta: Happens(PickUp(Agent2,Phone2),2).

[phone] HoldsAt(Idle(phone),0).
[phone] !HoldsAt(DialTone(phone),0).
[phone] !HoldsAt(BusySignal(phone),0).
[phone1,phone2] !HoldsAt(Ringing(phone1,phone2),0).
[phone1,phone2] !HoldsAt(Connected(phone1,phone2),0).
[phone] !HoldsAt(Disconnected(phone),0).

completion Delta Happens

range time 0 3
```

The output is as follows:

```
model 1:
0
Idle(Phone1).
Idle(Phone2).
Happens(PickUp(Agent1, Phone1), 0).
1
-Idle(Phone1).
+DialTone(Phone1).
Happens(Dial(Agent1, Phone1, Phone2), 1).
2
-DialTone(Phone1).
-Idle(Phone2).
```

```
  Term ::= Constant | Variable | FunctionSymbol(Term, ..., Term) |
           Term + Term | Term - Term | Term * Term | Term / Term |
           Term % Term | - Term | (Term)
  Atom ::= PredicateSymbol(Term,...,Term) | Term < Term |
           Term <= Term | Term = Term | Term >= Term |
           Term > Term | Term != Term
Formula ::= Atom | ! Formula | Formula & Formula |
           Formula | Formula | Formula -> Formula |
           Formula <-> Formula | {Variable,...,Variable} Formula |
           [Variable,...,Variable] Formula | (Formula)
```

Figure 13.2 Discrete Event Calculus Reasoner formula syntax.

```
+Ringing(Phone1, Phone2).
Happens(PickUp(Agent2, Phone2), 2).
3
-Ringing(Phone1, Phone2).
+Connected(Phone1, Phone2).
```

13.5 Discrete Event Calculus Reasoner Language

Domain descriptions are described using the Discrete Event Calculus Reasoner language. A domain description consists of zero or more formulas, and zero or more statements. The syntax for formulas, which is based on that of the Bliksem theorem prover, is shown in Figure 13.2. The meanings of the symbols used in formulas are shown in Table 13.1. As in this book, predicate symbols, function symbols, and nonnumeric constants start with an uppercase letter and variables start with a lowercase letter. The sort of a variable is determined by removing trailing digits from the variable. For example, the sort of the variable `time1` is time. Some Discrete Event Calculus Reasoner statements are shown in Table 13.2. A *noninertial fluent* is one that is released from the commonsense law of inertia at all timepoints.

Bibliographic Notes

The Discrete Event Calculus Reasoner was introduced by Mueller (2004a, 2004b). DIMACS format is specified by DIMACS (1993). The Relsat SAT solver is discussed by Bayardo and Schrag (1997) and Walksat is discussed by Selman, Kautz, and Cohen (1993). The technique of restricting the event calculus to a finite universe to enable SAT solving was introduced by Shanahan and Witkowski (2004).

Table 13.1 Discrete Event Calculus Reasoner formula symbols

Symbol	*Meaning*	*Symbol*	*Meaning*
+	Addition	!=	Not equal to
−	Subtraction, negation	!	Logical negation
*	Multiplication	&	Conjunction (AND, ∧)
/	Division	|	Disjunction (OR, ∨)
%	Remainder	->	Implication
<	Less than	<->	Bi-implication
<=	Less than or equal to	{ }	Existential quantifier (∃)
=	Equal to	[ ]	Universal quantifier (∀)
>=	Greater than or equal to	( )	Grouping
>	Greater than	,	Separator

Table 13.2 Discrete Event Calculus Reasoner statements

Statement	*Meaning*
`sort` Sort,...,Sort	Define sorts
`function` FunctionSymbol(Sort,...,Sort) : Sort	Define function
`predicate` PredicateSymbol(Sort,...,Sort)	Define predicate
`event` EventSymbol(Sort,...,Sort)	Define event
`fluent` FluentSymbol(Sort,...,Sort)	Define fluent
`noninertial` FluentSymbol,...,FluentSymbol	Define noninertial fluents
Sort Constant	Define constant
`completion` Label PredicateSymbol	Specify completion
`load` Filename	Load file
`option` OptionName OptionValue	Specify option
`range` Sort Integer Integer	Specify range

The restriction of first-order logic to a finite universe is discussed by Kautz and Selman (1992) and Jackson (2000). To produce a compact conjunctive normal form, the Discrete Event Calculus Reasoner uses the technique of renaming subformulas, which is described by Plaisted and Greenbaum (1986) and E. Giunchiglia and Sebastiani (1999). The Bliksem theorem prover is described by de Nivelle (1999).

Exercises

13.1 Formalize and run the Tweety example in the Bibliographic Notes of Chapter 12 in the Discrete Event Calculus Reasoner.

13.2 Using the Discrete Event Calculus Reasoner, formalize and run Shanahan's (1997b) kitchen sink scenario, which is described in Chapter 1.

13.3 Formalize the WALK x-schema of Narayanan (1997), and use the Discrete Event Calculus Reasoner to perform temporal projection for a scenario involving walking.

13.4 (Research Problem) Formalize scuba diving, and use the Discrete Event Calculus Reasoner to understand scuba-diving incident reports. See Section 14.2 for a discussion of the use of the event calculus in natural language understanding.

13.5 (Research Problem) The Discrete Event Calculus Reasoner currently produces models as output. Extend the Discrete Event Calculus Reasoner and a SAT solver to produce proofs for deduction problems using the techniques discussed by McMillan and Amla (2003, sec. 2).

CHAPTER 14

Applications

The availability of techniques for commonsense reasoning presents an enormous opportunity in that there are many areas that have only begun to be explored that can benefit from the use of commonsense reasoning. Commonsense reasoning provides two important benefits to computer applications:

- Commonsense reasoning can be used to give applications a greater degree of understanding about the human world. Users can be freed from having to state everything explicitly because commonsense reasoning can be used to infer what is obvious to a person.
- Commonsense reasoning can be used to give computer programs more flexibility. An application can adapt to an unexpected real-world problem by reasoning about it in order to come up with a solution.

In this chapter, we describe several applications that have already been built using commonsense reasoning in the event calculus, in business systems, natural language understanding, and vision.

14.1 Business Systems

An important area of application of the event calculus is business systems. In this section, we review two areas in which the event calculus has been applied: payment protocols and workflow modeling.

14.1.1 Payment Protocols

Pınar Yolum and Munindar P. Singh have used the event calculus to incorporate commonsense knowledge and reasoning about *commitments* into electronic

payment systems. Using Shanahan's (2000a) event calculus planner, they have implemented a mechanism that enables customer and merchant applications to (1) keep track of their commitments, (2) determine what actions are currently possible, and (3) plan actions to achieve goals.

Instead of describing a payment protocol as a finite state machine (FSM), as is traditionally done, the commitments inherent in the protocol are extracted and described using the event calculus. Applications that implement the protocol can then use the event calculus description to reason about their commitments. This makes the applications more flexible than those based on FSMs. An FSM describes only which actions are currently possible, not why they are possible. By using commonsense reasoning to reason about commitments, applications can generate their own plans for achieving their goals and cope with unanticipated situations.

NetBill Protocol

The mechanism is implemented for the NetBill protocol, which is used for the online purchase of digital products. This protocol works as follows:

1. A customer requests a price quote for a product from a merchant.
2. The merchant provides a price quote to the customer.
3. The customer sends a purchase request to the merchant.
4. The merchant delivers the product, in an encrypted form, to the customer.
5. The customer sends an electronic payment order (EPO) to the merchant.
6. The merchant sends the EPO to the NetBill server.
7. The NetBill server debits the customer's account and credits the merchant's account.
8. The NetBill server sends a receipt to the merchant.
9. The merchant sends the receipt and a decryption key for the product to the customer.
10. The customer decrypts the product.

Commitments are created at various points in this protocol. When the merchant provides a price quote, the merchant makes a commitment to deliver the product if the customer sends a purchase request. When the customer sends a purchase request, the customer makes a commitment to pay for the product if it is delivered.

Suppose that a customer has not yet purchased a certain product the merchant application wishes to sell. The merchant application can reason that it is possible to send an unsolicited price quote to the customer. There is nothing in the protocol that prohibits this, and it might lead to a sale. Similarly, a customer application can reason that it is possible to send a purchase request without having first received a

price quote. In this case, the customer makes a commitment to pay if the product is delivered, which the merchant has not promised to do.

Formalization

We now describe part of the event calculus formalization of these notions. We start with the representation of commitments. We use the following fluents to represent commitments.

C(a_1, a_2, f): Agent a_1 is committed to agent a_2 to bring about fluent f. This is called a *base-level commitment.*

CC(a_1, a_2, c, f): If fluent c is true, then agent a_1 is committed to a_2 to bring about fluent f. This is called a *conditional commitment.*

We have effect axioms for creating and discharging base-level and conditional commitments:

$$Initiates(CreateC(a_1, a_2, f), C(a_1, a_2, f), t) \tag{14.1}$$

$$Initiates(CreateCC(a_1, a_2, c, f), CC(a_1, a_2, c, f), t) \tag{14.2}$$

$$Terminates(DischargeC(a_1, a_2, f), C(a_1, a_2, f), t) \tag{14.3}$$

$$Terminates(DischargeCC(a_1, a_2, c, f), CC(a_1, a_2, c, f), t) \tag{14.4}$$

We have a trigger axiom that, states that, if a_1 is committed to a_2 to bring about f and f is true, then the commitment is discharged:

$$HoldsAt(C(a_1, a_2, f), t) \wedge HoldsAt(f, t) \Rightarrow Happens(DischargeC(a_1, a_2, f), t) \tag{14.5}$$

We also have trigger axioms stating that, if a_1 has a conditional commitment to a_2 to bring about f if c is true and c is true, then a base-level commitment of a_1 to a_2 to bring about f is created and the conditional commitment is discharged:

$$HoldsAt(CC(a_1, a_2, c, f), t) \wedge HoldsAt(c, t) \Rightarrow Happens(CreateC(a_1, a_2, f), t) \tag{14.6}$$

$$HoldsAt(CC(a_1, a_2, c, f), t) \wedge HoldsAt(c, t) \Rightarrow Happens(DischargeCC(a_1, a_2, c, f), t) \tag{14.7}$$

Next we represent part of the NetBill protocol and associated commitments. We use the following events.

RequestQuote(c, m, p): Customer c sends merchant m a request for a price quote for product p.

SendQuote(m, c, p, a): Merchant m sends customer c a quote of amount a for product p.

RequestPurchase(c, m, p, a): Customer c sends merchant m a request to purchase product p for amount a.

Deliver(m, c, p): Merchant m delivers product p to customer c.

SendEPO(c, m, a): Customer c sends merchant m an EPO for amount a.

We use the following fluents:

QuoteRequested(c, m, p): Customer c has sent merchant m a request for a price quote for product p.

QuoteSent(m, c, p, a): Merchant m has sent customer c a quote of amount a for product p.

PurchaseRequested(c, m, p, a): Customer c has sent merchant m a request to purchase product p for amount a.

Delivered(m, c, p): Merchant m has delivered product p to customer c.

EPOSent(c, m, a): Customer c has sent merchant m an EPO for amount a.

Effect axioms relate these events and fluents:

$$Initiates(RequestQuote(c, m, p), QuoteRequested(c, m, p), t) \quad (14.8)$$

$$Initiates(SendQuote(m, c, p, a), QuoteSent(m, c, p, a), t) \quad (14.9)$$

$$Initiates(RequestPurchase(c, m, p, a), PurchaseRequested(c, m, p, a), t) \quad (14.10)$$

$$Initiates(Deliver(m, c, p), Delivered(m, c, p), t) \quad (14.11)$$

$$Initiates(SendEPO(c, m, a), EPOSent(c, m, a), t) \quad (14.12)$$

We now use effect axioms to describe the effects of NetBill events on commitments. If the merchant sends the customer a price quote for a product, then the merchant will have a conditional commitment to deliver the product if the customer promises to pay the quoted price for the product:

$$Initiates(SendQuote(m, c, p, a), \quad (14.13)$$
$$CC(m, c, PurchaseRequested(c, m, p, a), Delivered(m, c, p)), t)$$

If the merchant has yet not delivered a given product to the customer and the customer sends a purchase request for the product to the merchant, then the customer will have a conditional commitment to pay for the product if the merchant delivers it:

$$\neg HoldsAt(Delivered(m, c, p), t) \Rightarrow \quad (14.14)$$
$$Initiates(RequestPurchase(c, m, p, a),$$
$$CC(c, m, Delivered(m, c, p), EPOSent(c, m, a)), t)$$

Scenario

Let us now consider a specific scenario. Suppose that a particular customer has not yet purchased anything from a particular music store:

$$\neg HoldsAt(PurchaseRequested(Jen, MusicStore, p, a), 0) \quad (14.15)$$
$$\neg HoldsAt(Delivered(MusicStore, Jen, p), 0) \quad (14.16)$$
$$\neg HoldsAt(EPOSent(Jen, MusicStore, a), 0) \quad (14.17)$$

Further suppose that the music store has the goal to sell the customer a particular CD:

$$\exists t\,(HoldsAt(Delivered(MusicStore, Jen, BritneyCD), t) \wedge HoldsAt(EPOSent(Jen, MusicStore, 14.99), t)) \quad (14.18)$$

We can then show that there are several ways this goal can be achieved; see Exercises 14.1, 14.2, and 14.3. The music store can start by sending an unsolicited price quote:

$$Happens(SendQuote(MusicStore, Jen, BritneyCD, 14.99), 1) \quad (14.19)$$
$$Happens(RequestPurchase(Jen, MusicStore, BritneyCD, 14.99), 2) \quad (14.20)$$
$$Happens(Deliver(MusicStore, Jen, BritneyCD), 3) \quad (14.21)$$
$$Happens(SendEPO(Jen, MusicStore, 14.99), 4) \quad (14.22)$$

The customer can start by sending an unsolicited purchase request:

$$Happens(RequestPurchase(Jen, MusicStore, BritneyCD, 14.99), 1) \quad (14.23)$$
$$Happens(Deliver(MusicStore, Jen, BritneyCD), 2) \quad (14.24)$$
$$Happens(SendEPO(Jen, MusicStore, 14.99), 3) \quad (14.25)$$

The music store can even start by delivering the product:

$$Happens(Deliver(MusicStore, Jen, BritneyCD), 1) \quad (14.26)$$
$$Happens(SendEPO(Jen, MusicStore, 14.99), 2) \quad (14.27)$$

Thus, the event calculus provides the music store with several possible ways a sale could occur. The music store could perform additional commonsense reasoning to determine which course of action is best.

14.1.2 Workflow Modeling

A method for using the event calculus to specify, simulate, and execute workflows has been defined by Nihan Kesim Cicekli and Yakup Yildirim. A workflow consists

of a set of business activities and dependencies among the activities. For example, the workflow in a newspaper might consist of (1) a set of activities $w_1, \ldots, w_n$ in which reporters write stories; (2) a set of activities $e_1, \ldots, e_n$ in which editors edit stories written by reporters, where for each $i \in \{1, \ldots, n\}$, e_i depends on w_i; and (3) a pagination activity p in which the stories are put into place on pages, which depends on all of the activities $e_1, \ldots, e_n$.

The Workflow Management Coalition defines several types of dependencies that can be part of a workflow.

In a *sequence*, one activity follows another activity; this is the case for e_i and w_i, just described.

In an *AND-split*, several activities follow one activity, and these activities are performed in parallel.

In an *AND-join*, an activity starts after several other activities have all ended; this is the case for p.

In an *XOR-split*, one of several activities follows an activity, depending on what condition is true.

In an *XOR-join*, an activity starts after any of several other activities have ended.

In an *iteration*, one or more activities are repeatedly performed while a condition is true.

The basic method for modeling workflows in the event calculus is as follows. We use DEC. The fluent $Active(a)$ represents that activity a is active, and the fluent $Completed(a)$ represents that activity a is completed. The event $Start(a)$ represents that activity a starts, and the event $End(a)$ represents that activity a ends. Effect axioms state that, if an activity starts, then it will be active and no longer completed, and, if an activity ends, then it will be completed and no longer active:

$$Initiates(Start(a), Active(a), t)$$
$$Terminates(Start(a), Completed(a), t)$$
$$Initiates(End(a), Completed(a), t)$$
$$Terminates(End(a), Active(a), t)$$

Dependencies are represented using trigger axioms that start activities in appropriate situations. The following trigger axiom represents a sequence in which activity A is followed by activity B.

$$\neg HoldsAt(Active(B), t) \wedge$$
$$\neg HoldsAt(Completed(A), t - 1) \wedge$$
$$HoldsAt(Completed(A), t) \Rightarrow$$
$$Happens(Start(B), t)$$

The following trigger axioms represent an AND-split in which *C1*, *C2*, and *C3* follow *B*.

$$\neg HoldsAt(Active(C1), t) \wedge$$
$$\neg HoldsAt(Completed(B), t-1) \wedge$$
$$HoldsAt(Completed(B), t) \Rightarrow$$
$$Happens(Start(C1), t)$$

$$\neg HoldsAt(Active(C2), t) \wedge$$
$$\neg HoldsAt(Completed(B), t-1) \wedge$$
$$HoldsAt(Completed(B), t) \Rightarrow$$
$$Happens(Start(C2), t)$$

$$\neg HoldsAt(Active(C3), t) \wedge$$
$$\neg HoldsAt(Completed(B), t-1) \wedge$$
$$HoldsAt(Completed(B), t) \Rightarrow$$
$$Happens(Start(C3), t)$$

The following trigger axiom represents an AND-join in which *D* starts after *C1*, *C2*, and *C3* end.

$$\neg HoldsAt(Active(D), t) \wedge$$
$$((\neg HoldsAt(Completed(C1), t-1) \wedge HoldsAt(Completed(C1), t)) \vee$$
$$(\neg HoldsAt(Completed(C2), t-1) \wedge HoldsAt(Completed(C2), t)) \vee$$
$$(\neg HoldsAt(Completed(C3), t-1) \wedge HoldsAt(Completed(C3), t))) \wedge$$
$$HoldsAt(Completed(C1), t) \wedge$$
$$HoldsAt(Completed(C2), t) \wedge$$
$$HoldsAt(Completed(C3), t) \Rightarrow$$
$$Happens(Start(D), t)$$

The following trigger axioms represent an XOR-split in which *E1* follows *D* if *E1C* is true, in which *E2* follows *D* if *E2C* is true, and in which *E3* follows *D* if *E3C* is true.

$$\neg HoldsAt(Active(E1), t) \wedge$$
$$\neg HoldsAt(Completed(D), t-1) \wedge$$
$$HoldsAt(Completed(D), t) \wedge$$
$$HoldsAt(E1C, t) \Rightarrow$$
$$Happens(Start(E1), t)$$

$$\neg HoldsAt(Active(E2), t) \wedge$$
$$\neg HoldsAt(Completed(D), t-1) \wedge$$
$$HoldsAt(Completed(D), t) \wedge$$
$$HoldsAt(E2C, t) \Rightarrow$$
$$Happens(Start(E2), t)$$

$$\neg HoldsAt(Active(E3), t) \wedge$$
$$\neg HoldsAt(Completed(D), t-1) \wedge$$
$$HoldsAt(Completed(D), t) \wedge$$
$$HoldsAt(E3C, t) \Rightarrow$$
$$Happens(Start(E3), t)$$

The following trigger axioms represent an XOR-join in which F starts after $E1$, $E2$, or $E3$ ends.

$$\neg HoldsAt(Active(F), t) \wedge$$
$$((\neg HoldsAt(Completed(E1), t-1) \wedge HoldsAt(Completed(E1), t)) \vee$$
$$(\neg HoldsAt(Completed(E2), t-1) \wedge HoldsAt(Completed(E2), t)) \vee$$
$$(\neg HoldsAt(Completed(E3), t-1) \wedge HoldsAt(Completed(E3), t))) \Rightarrow$$
$$Happens(Start(F), t)$$

The following trigger axioms represent an iteration in which activity F is repeatedly performed while a condition FC is true. When FC is false, activity G is performed.

$$\neg HoldsAt(Active(F), t) \wedge$$
$$\neg HoldsAt(Completed(F), t-1) \wedge$$
$$HoldsAt(Completed(F), t) \wedge$$
$$HoldsAt(FC, t) \Rightarrow$$
$$Happens(Start(F), t).$$

$$\neg HoldsAt(Active(G), t) \wedge$$
$$\neg HoldsAt(Completed(F), t-1) \wedge$$
$$HoldsAt(Completed(F), t) \wedge$$
$$\neg HoldsAt(FC, t) \Rightarrow$$
$$Happens(Start(G), t)$$

Once a workflow is modeled using the event calculus, temporal projection can be used to simulate the workflow. The model can also be used by a workflow

manager in a workflow management system to manage the execution of a workflow by the workflow participants. The event calculus enables the past, present, and possible future states of a workflow to be queried at any point in the simulation or execution of the workflow.

14.2 Natural Language Understanding

Understanding natural language text or speech involves building representations of the meaning of that text or speech. The event calculus can be used to perform commonsense reasoning in order to build representations of meaning, and formulas of the event calculus can be used to represent meaning.

For example, we can use a semantic parser to convert the English sentence *Nathan wakes up* into the event calculus formula

$$Happens(WakeUp(Nathan), 0)$$

Then, suppose we have the following representations of commonsense knowledge:

$$Initiates(WakeUp(a), Awake(a), t)$$
$$Happens(WakeUp(a), t) \Rightarrow \neg HoldsAt(Awake(a), t)$$

We can reason using the event calculus in order to build the following representation of the sentence's meaning.

$$\neg HoldsAt(Awake(Nathan), 0)$$
$$Happens(WakeUp(Nathan), 0)$$
$$HoldsAt(Awake(Nathan), 1)$$

In general, the event calculus can be used in a natural language understanding system as follows:

1. The set of domains to be understood by the system is determined.
2. Commonsense knowledge about the domains is represented using the event calculus.
3. For each natural language input:
   (a) The input is parsed by syntactic and/or semantic parsers into predicate-argument structure representations, which resemble event calculus *Happens* and *HoldsAt* formulas.
   (b) The predicate-argument structure representations are converted into event calculus *Happens* and *HoldsAt* formulas.

(c) The event calculus formulas are fed to an event calculus reasoning program, which uses the commonsense knowledge to produce additional event calculus formulas, or inferences.

(d) The inferences are produced as output.

The output can be used for various purposes, including dialog management, information retrieval, question answering, summarization, updating a database, and user modeling.

14.2.1 Story Understanding

The event calculus can be used to address the problem of *story understanding*, which consists of taking a story as input, understanding it, and then answering questions about it. Commonsense reasoning can be used to fill in details not explicitly stated in the input story. The Discrete Event Calculus Reasoner program can be used to build detailed models of a story, which represent the events that occur and the properties that are true or false at various times.

For example, the children's story by Raymond Briggs (1999, pp. 4–8) begins as follows:

Hooray! It is snowing! James gets dressed. He runs outside. He makes a pile of snow. He makes it bigger and bigger. He puts a big snowball on top.

Given a domain description representing this text consisting of (1) a basic axiomatization of space and intentions, (2) observations of the initial state, and (3) a narrative of events, the Discrete Event Calculus Reasoner produces the following model.[1] The first input event is that it starts snowing outside James's house at timepoint 0:

```
0
Happens(StartSnowing(JamesOutside),0).
```

From this input event, it is inferred that it is snowing outside James's house at timepoint 1:

```
1
+Snowing(JamesOutside).
```

The next input event is that James wakes up:

```
Happens(WakeUp(James), 1).
```

1. This model is taken from Mueller (2003, sec. 4), published by the Association for Computational Linguistics.

It is inferred that he is no longer asleep and is awake:

```
2
-Asleep(James).
+Awake(James).
```

Next, it is inferred that James becomes happy:

```
Happens(BecomeHappy(James), 2).
```

A trigger axiom states that if an agent is awake, likes snow, and is in a room that looks out onto a location where it is snowing, then the agent will become happy. It is also inferred that James intends to play outside his house:

```
Happens(IntendToPlay(James, JamesOutside), 2).
```

A trigger axiom activates an intention for an agent to play when the agent has an unsatisfied need for play, likes snow, is awake, and is in a room that looks out onto an outside area where it is snowing. From these events, it is inferred that James is no longer calm, is happy, and intends to play outside his house:

```
3
-Calm(James).
+Happy(James).
+IntentionToPlay(James, JamesOutside).
```

The next input events are that James cries for joy and gets up from his bed:

```
Happens(CryForJoy(James), 3).
4
Happens(RiseFrom(James, JamesBed), 4).
```

It is inferred that James is no longer lying on his bed, is no longer lying down, and is standing up:

```
5
-LyingOn(James, JamesBed).
-Lying(James).
+Standing(James).
```

Next, we have the input event that James gets dressed, from which it is inferred that he is dressed:

```
Happens(GetDressed(James), 5).
6
+Dressed(James).
```

The next input event is that James walks through the door of his bedroom, from which it is inferred that he is no longer in his bedroom, is in the hallway, and is near

the staircase.

```
Happens(WalkThroughDoor12(James, JamesDoor2Fl), 6).
7
-At(James, JamesBedroom2Fl).
+At(James, JamesHallway2Fl).
+NearPortal(James, JamesStaircase1To2).
```

We have the input event that James walks down the staircase, from which it is inferred that he is no longer in the hallway, is no longer near the door of his bedroom, is in the foyer, and is near the front door and the kitchen door:

```
Happens(WalkDownStaircase(James, JamesStaircase1To2), 7).
8
-At(James, JamesHallway2Fl).
-NearPortal(James, JamesDoor2Fl).
+At(James, JamesFoyer1Fl).
+NearPortal(James, JamesFrontDoor1Fl).
+NearPortal(James, JamesKitchenDoor1Fl).
```

The next input event is that James unlocks the front door, from which it is inferred that the front door is unlocked:

```
Happens(DoorUnlock(James, JamesFrontDoor1Fl), 8).
9
+DoorUnlocked(JamesFrontDoor1Fl).
```

The input is provided that James opens the front door, and it is inferred that the door is open:

```
Happens(DoorOpen(James, JamesFrontDoor1Fl), 9).
10
+DoorIsOpen(JamesFrontDoor1Fl).
```

The input is then provided that James walks through the front door, from which it is inferred that he is no longer in the foyer, is no longer near the kitchen door or the staircase, and is outside:

```
Happens(WalkThroughDoor21(James, JamesFrontDoor1Fl), 10).
11
-At(James, JamesFoyer1Fl).
-NearPortal(James, JamesKitchenDoor1Fl).
-NearPortal(James, JamesStaircase1To2).
+At(James,JamesOutside).
```

The input is provided that James plays at timepoint 11:

```
+ActOnIntentionToPlay(James, JamesOutside).
Happens(Play(James, JamesOutside), 11).
```

It is inferred that the need to play is satisfied:

```
12
-ActOnIntentionToPlay(James, JamesOutside).
-IntentionToPlay(James, JamesOutside).
+SatiatedFromPlay(James).
```

It is provided as input that James picks up some snow, rolls it into a ball, and lets go of it:

```
Happens(HoldSome(James, Snowball1, Snow1), 12).
13
+Holding(James, Snowball1).
Happens(RollAlong(James, Snowball1, Snow1), 13).
14
-Diameter(Snowball1, 1).
+Diameter(Snowball1, 2).
Happens(LetGoOf(James, Snowball1), 14).
15
-Holding(James, Snowball1).
```

James does the same to make another snowball:

```
Happens(HoldSome(James, Snowball2, Snow1), 15).
16
+Holding(James, Snowball2).
Happens(RollAlong(James, Snowball2, Snow1), 16).
17
-Diameter(Snowball2, 1).
+Diameter(Snowball2, 2).
```

Finally, we have the input event that James places the second snowball on top of the first:

```
Happens(PlaceOn(James, Snowball2, Snowball1), 17).
18
-Holding(James, Snowball2).
+On(Snowball2, Snowball1).
```

14.3 Vision

Murray Shanahan and David Randell use the event calculus to implement the higher-level vision component of Ludwig, an upper-torso humanoid robot. Ludwig has two arms, each with three degrees of freedom, and a stereo camera hooked up to a head with two degrees of freedom (pan and tilt). The low-level vision component uses off-the-shelf edge detection software to map raw images from the camera into a list of edges. The list of edges is fed to the higher-level vision component,

which is responsible for recognizing shapes. This component is implemented using the event calculus in Prolog.

The higher-level vision component consists of three layers. The first layer generates hypotheses about what regions are in view, based on the input list of edges. The second layer generates hypotheses about what aspects are in view, based on what regions are in view. The *aspects* of a shape are the various ways it can appear when viewed from different angles; for example, a wedge viewed from above appears to be a rectangle, but a wedge viewed from the side appears to be a triangle. The third layer generates hypotheses about what shapes are in view, based on the aspects that are in view over time.

We present here a simplified version of Ludwig's third layer. The predicate $Arc(s, a_1, a_2)$ represents that, by gradually changing the orientation of a shape s with respect to the camera, it is possible for the appearance of s to change from aspect a_1 to aspect a_2. For example, the appearance of a wedge can change from a rectangle to a rectangle plus an adjacent triangle, but it cannot change immediately from a rectangle to a triangle—it must first appear as a rectangle plus an adjacent triangle. The predicate $Shape(o, s)$ represents that object o has shape s. The fluent $Aspect(o, a)$ represents that the appearance of object o is aspect a. The event $Change(o, a_1, a_2)$ represents that the appearance of object o changes from aspect a_1 to aspect a_2.

We start with state constraints that say that an object has unique shape and aspect:

$$Shape(o, s_1) \wedge Shape(o, s_2) \Rightarrow s_1 = s_2$$
$$HoldsAt(Aspect(o, a_1), t) \wedge HoldsAt(Aspect(o, a_2), t) \Rightarrow a_1 = a_2$$

We have effect axioms that state that if the aspect of an object is a_1, the shape of the object is s, it is possible for the appearance of s to change from aspect a_1 to aspect a_2, and the object changes from a_1 to a_2, then the aspect of the object will be a_2 and will no longer be a_1:

$$\begin{gathered} HoldsAt(Aspect(o, a_1), t) \wedge \\ Shape(o, s) \wedge \\ (Arc(s, a_1, a_2) \vee Arc(s, a_2, a_1)) \Rightarrow \\ Initiates(Change(o, a_1, a_2), Aspect(o, a_2), t) \end{gathered}$$

$$\begin{gathered} HoldsAt(Aspect(o, a_1), t) \wedge \\ Shape(o, s) \wedge \\ (Arc(s, a_1, a_2) \vee Arc(s, a_2, a_1)) \Rightarrow \\ Terminates(Change(o, a_1, a_2), Aspect(o, a_1), t) \end{gathered}$$

Now, suppose we have the following visual knowledge about two shapes *Shape1* and *Shape2*:

$$Arc(Shape1, a_1, a_2) \Leftrightarrow$$
$$(a_1 = Aspect1 \wedge a_2 = Aspect2)$$

$$Arc(Shape2, a_1, a_2) \Leftrightarrow$$
$$((a_1 = Aspect1 \wedge a_2 = Aspect3) \vee$$
$$(a_1 = Aspect3 \wedge a_2 = Aspect2))$$

Further, suppose we observe at timepoint 0 that *Object1* has *Aspect1* and at timepoint 1 that *Object1* has *Aspect2*:

$$HoldsAt(Aspect(Object1, Aspect1), 0)$$
$$HoldsAt(Aspect(Object1, Aspect2), 1)$$

We can then show that the shape of *Object1* must be *Shape1* and not *Shape2*:

$$Shape(Object1, Shape1)$$

Note that this inference depends crucially on the sequence of images over time rather than on a single image. These sorts of inferences are supported by the ability of the event calculus to reason about time.

Bibliographic Notes

Lenat and Guha (1991) and Lieberman, Liu, Singh, and Barry (2004) argue for the use of commonsense reasoning in applications and present numerous examples. The payment protocol application of the classical logic event calculus is from Yolum and Singh (2004); our formalization is adapted from theirs. NetBill is described by Sirbu and Tygar (1995). The workflow modeling application is from Cicekli and Yildirim (2000). They use a logic programming version of the event calculus; the axioms we present are adapted to the classical logic event calculus. Terminology relating to workflows is defined by the Workflow Management Coalition (1999). Others have used the event calculus for workflow modeling; Wilk (2004) discusses the use of the classical logic event calculus in a hospital workflow management system.

Textbooks on natural language processing are by Allen (1995), Manning and Schütze (1999), and Jurafsky and Martin (2000). Deep natural language understanding systems are discussed by Ram and Moorman (1999). Commonsense reasoning is used by a number of deep natural language and discourse understanding systems (Charniak, 1972; Schank and Abelson, 1977; Schank and Riesbeck,

1981; Dyer, 1983; Dahlgren, McDowell, and Stabler, 1989; Norvig, 1989; Hobbs et al. 1993; Schank, Kass, and Riesbeck, 1994; J. D. Moore, 1995; Mueller, 1998). Halpin, J. D. Moore, and Robertson (2004) use a simplified event calculus to represent event occurrences in stories. van Lambalgen and Hamm (2005) use the classical logic event calculus to represent the semantics of tense and aspect in natural language. Predicate-argument structure representations are used in the Proposition Bank (Palmer, Gildea, and Kingsbury, 2005) annotated corpus. A modern syntactic parser is that of M. Collins (1999, 2003). Some semantic parsers are those of Alshawi (1992), Beale, Nirenburg, and Mahesh (1995), and Gildea and Jurafsky (2002).

The problem of story understanding and systems for story understanding are reviewed by Domeshek, Jones, and Ram (1999) and Mueller (2002). The use of the Discrete Event Calculus Reasoner for modeling children's stories is discussed by Mueller (2003). This program has also been used to model stories involving scripts, such as eating in a restaurant and kidnapping (Mueller, 2004c).

The use of the event calculus for robot perception is discussed by Shanahan (1999c, 2005). The Ludwig robot is discussed by Shanahan and Randell (2004) and Shanahan (2005, pp. 116–125). The axioms we present are simplified versions of axioms in the third layer of Ludwig's vision system. Koenderink and Doorn (1979) discuss the notion of an aspect of an object and the representation of the shape of an object as a connected graph whose nodes represent aspects and whose links represent visual events.

Exercises

14.1 In the scenario in Section 14.1.1, prove that the sequence of events starting with *SendQuote* achieves the goal of selling the customer the CD.

14.2 In the scenario in Section 14.1.1, prove that the sequence of events starting with *RequestPurchase* achieves the goal of selling the customer the CD.

14.3 In the scenario in Section 14.1.1, prove that the sequence of events starting with *Deliver* achieves the goal of selling the customer the CD.

14.4 (Research Problem) Use the event calculus to model the semantics of your favorite programming language.

14.5 (Research Problem) Use the event calculus to build an automated advisor for your favorite computer program.

14.6 (Research Problem) Use the event calculus to model the operation of a stock exchange.

CHAPTER 15

Logics for Commonsense Reasoning

Why the event calculus? How does the event calculus stack up against other logical formalisms for commonsense reasoning? There are many such formalisms. In this chapter, we review four important families—the situation calculus, the features and fluents framework, action languages, and the fluent calculus—discussing their relationship to the event calculus.

15.1 The Situation Calculus

The main difference between the situation calculus and the event calculus is that the situation calculus uses branching time, whereas the event calculus uses linear time. In the event calculus, there is a single time line on which events occur, and all events are considered to be actual events. In the situation calculus, time is represented by a tree, and each event may give rise to a different possible future; therefore, events are considered to be hypothetical events. A point in time is represented by a situation. In this section, we use a standard version of the situation calculus described by Raymond Reiter. Notational variants of the situation calculus are shown in Table 15.1.

15.1.1 Relational and Functional Fluents

The truth value of a relational fluent is represented by

$$(\neg)F(x_1, \ldots, x_n, \sigma)$$

Table 15.1 Situation calculus notational variants[a]

Source	*Fluent*	*Successor*
(McCarthy and Hayes, 1969)	$F(x_1, \ldots, x_n, \sigma)$	$result(\pi, \alpha, \sigma)$
(McCarthy, 1980)	$F(x_1, \ldots, x_n, \sigma)$	$result(\alpha, \sigma)$
(Russell and Norvig, 2003)	$F(x_1, \ldots, x_n, \sigma)$	$Result(\alpha, \sigma)$
(Hayes, 1973)	$F(x_1, \ldots, x_n, \sigma)$	$R(\alpha, \sigma)$
(Hayes, 1971; Reiter 2001; Brachman and Levesque, 2004)	$F(x_1, \ldots, x_n, \sigma)$	$do(\alpha, \sigma)$
(Green, 1969; Winston, 1984)	$F(x_1, \ldots, x_n, \sigma)$	$A(x_1, \ldots, x_n, \sigma)$
(Kowalski, 1979)	$Holds(\phi, \sigma)$	$result(\alpha, \sigma)$
(Shanahan, 1997b)	$Holds(\phi, \sigma)$	$Result(\alpha, \sigma)$
(E. Davis, 1990)	$true\_in(\sigma, \phi)$ $value\_in(\sigma, \phi)$	$result(\sigma, \alpha)$
(Charniak and McDermott, 1985)	$(T\ \sigma\ \phi)$	$(result\ \sigma\ (do\ \pi\ \alpha))$
(Genesereth and Nilsson, 1987)	$T(\phi, \sigma)$	$Do(\alpha, \sigma)$

[a] α = action; π = person; σ = situation; ϕ = fluent; A = action predicate; F = fluent predicate; $x_1, \ldots, x_n$ = action and fluent arguments.

which represents that fluent $F(x_1, \ldots, x_n)$ is true (false) in situation σ. The value of a functional fluent is represented by

$$f(x_1, \ldots, x_n, \sigma) = v$$

which represents that fluent $f(x_1, \ldots, x_n)$ has value v in situation σ.

15.1.2 Actions

The performance of an action is represented by

$$\sigma' = do(\alpha, \sigma)$$

which represents that situation σ' is the successor situation that results from performing action α in situation σ. There is an initial situation, called S_0.

15.1.3 Action Effects

An *effect axiom* of the form

$$R(x_1, \ldots, x_n, \sigma) \supset (\neg)F(x_1, \ldots, x_n, do(\alpha, \sigma))$$

represents that under the conditions given by $R(x_1, \ldots, x_n, \sigma)$, fluent $F(x_1, \ldots, x_n)$ is true (false) in the situation that results from performing action α in situation σ.

The effect axioms for each fluent $F(x_1, \ldots, x_n)$ are compiled into a single successor state axiom as follows. The conditions under which an action α initiates the fluent are collected into a single formula $\gamma_F^+(x_1, \ldots, x_n, \alpha, \sigma)$, and the conditions under which α terminates the fluent are collected into a single formula $\gamma_F^-(x_1, \ldots, x_n, \alpha, \sigma)$. The *successor state axiom* is then as follows:

$$F(x_1, \ldots, x_n, do(\alpha, \sigma)) \equiv$$
$$\gamma_F^+(x_1, \ldots, x_n, \alpha, \sigma) \vee (F(x_1, \ldots, x_n, \sigma) \wedge \neg\gamma_F^-(x_1, \ldots, x_n, \alpha, \sigma))$$

This states that a fluent is true in the situation that results from performing an action α in situation σ if and only if (1) α initiates the fluent or (2) the fluent is true in σ and α does not terminate the fluent. (Notice the resemblance of $\gamma_F^+(x_1, \ldots, x_n, \alpha, \sigma)$ to $Initiates(\alpha, F(x_1, \ldots, x_n), \sigma)$ in the event calculus and of $\gamma_F^-(x_1, \ldots, x_n, \alpha, \sigma)$ to $Terminates(\alpha, F(x_1, \ldots, x_n), \sigma)$.)

15.1.4 Action Preconditions

An *action precondition axiom* of the form

$$Poss(A(x_1, \ldots, x_n), \sigma) \equiv P(x_1, \ldots, x_n, \sigma)$$

represents that it is possible to perform the action $A(x_1, \ldots, x_n)$ in situation σ if and only if $P(x_1, \ldots, x_n, \sigma)$ is true. A situation is executable if and only if all the actions leading to that situation are possible to perform. Nonexecutable situations are those resulting from one or more impossible actions.

15.1.5 Equivalence of the Situation Calculus and the Event Calculus

Robert Kowalski and Fariba Sadri have proved the equivalence of a version of the situation calculus similar to that of Reiter and a version of the event calculus. Their version of the situation calculus assumes that no fluents are true in the initial situation and includes terms representing transitions from one situation to another. Their version of the event calculus differs from the one described in this book as follows:

- It supports branching time.
- It does not support concurrent events.
- It does not support release from the commonsense law of inertia.
- It does not support continuous change.
- It has an initial situation, and no fluents are true in the initial situation.
- It includes terms representing transitions from one situation to another.

Their version supports a distinction between actual and hypothetical situations, but this is not required for the equivalence proof. Their version of the event calculus does not support continuous time, which is also the case for the discrete event calculus. Regarding the relation of the discrete event calculus and the situation calculus, see Exercises 15.1 and 15.2.

15.1.6 Discussion

Like the event calculus, the situation calculus supports context-sensitive effects of events, indirect effects, preconditions of actions, and the commonsense law of inertia. The situation calculus can be used to represent and reason about hypothetical events and situations, which cannot be done in the event calculus unless it is extended to support branching time, as proposed by Kowalski and Sadri.

The situation calculus supports functional fluents, which are not supported by the event calculus. Nonetheless, a functional fluent may be represented in the event calculus by (1) using a relational fluent, (2) adding an extra argument to the relational fluent, and (3) adding state constraints ensuring that the relational fluent is functional. For example, if we wish to represent a functional fluent $F(x) = y$, we instead use a relational fluent $F(x, y)$ and the following state constraints:

$$HoldsAt(F(x, y_1), t) \wedge HoldsAt(F(x, y_2), t) \Rightarrow y_1 = y_2$$
$$\exists y\ HoldsAt(F(x, y), t)$$

The basic situation calculus does not support continuous change because of the discrete nature of the tree of situations. The basic situation calculus also does not support triggered events because many hypothetical events are possible in any given situation. Nondeterminism is also not supported in the basic situation calculus. A number of extensions to the situation calculus have been proposed to address these phenomena.

The basic situation calculus does not support the representation of actual situations and actual events. Two methods for extending the situation calculus to support these have been proposed:

- Situations are designated as actual using a predicate *actual*(s), and actions are designated as actually occurring in a situation using a predicate *occurs*(a, s). Time may also be added, and a function *start*(s) may be used to represent the start time of a situation.
- A time line is added to the situation calculus, times are associated with actual situations using a function *State*(t), and actions are designated as actually occurring at a time using a predicate *Happens*(a, t).

Concurrent events cannot normally be represented in the situation calculus, because each transition from situation to situation represents a single event.

Extensions have been proposed in which sets of concurrent events can be associated with transitions.

15.2 The Features and Fluents Framework

The features and fluents framework was developed by Erik Sandewall. The framework includes a specialized logic called discrete fluent logic and a methodology for assessing the range of applicability of logics. Discrete fluent logic was recast in many-sorted first-order logic with equality by Patrick Doherty and Witold Łukaszewicz. This work gave rise to a class of logics called temporal action logics (TAL) developed by Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. TAL uses the pointwise minimization of occlusion with nochange premises (PMON) entailment method, which is one of the entailment methods defined in the features and fluents framework.

15.2.1 Temporal Action Logics

TAL has many similarities with the event calculus. Like the event calculus, and unlike the situation calculus, TAL uses linear time. (In general, the features and fluents framework supports both linear and branching time.) Like the event calculus, TAL uses circumscription to enforce the commonsense law of inertia. TAL has a predicate, $Occlude(t, f)$, similar to the event calculus predicate $ReleasedAt(f, t)$, which represents that a fluent f is released from the commonsense law of inertia at timepoint t. The event calculus inherited this construct from the features and fluents framework. TAL uses discrete time, like the discrete event calculus. Whereas the discrete event calculus uses the integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ for representing time, TAL uses the nonnegative integers $\{0, 1, 2, \ldots\}$. The relation between the discrete event calculus and TAL is explored in Exercise 15.3.

TAL consists of two languages: a narrative description language, $\mathcal{L}(ND)$, and a fluent logic language, $\mathcal{L}(FL)$. Commonsense reasoning problems are described using $\mathcal{L}(ND)$, which is a specialized language. Descriptions in $\mathcal{L}(ND)$ are then translated into $\mathcal{L}(FL)$, which is a many-sorted first-order language. We now describe how things are represented in $\mathcal{L}(ND)$ and translated into $\mathcal{L}(FL)$.

Observation Statements

The $\mathcal{L}(ND)$ expression

$$[\tau]\, f(\omega_1, \ldots, \omega_n) \hat{=} v$$

represents that fluent $f(\omega_1, \ldots, \omega_n)$ has the value v at timepoint τ. f is called a feature hence the name features and fluents. (We call f a fluent symbol.)

If f has a Boolean domain, then the $\mathcal{L}(FL)$ translation of $[\tau]\, f(\omega_1, \ldots, \omega_n) \hat{=} T$ is the observation statement

$$Holds(\tau, f(\omega_1, \ldots, \omega_n))$$

and the translation of $[\tau]\, f(\omega_1, \ldots, \omega_n) \hat{=} F$ is the observation statement

$$\neg Holds(\tau, f(\omega_1, \ldots, \omega_n))$$

If f has a non-Boolean domain, then the $\mathcal{L}(FL)$ translation of $[\tau]\, f(\omega_1, \ldots, \omega_n) \hat{=} v$ is the observation statement

$$Holds(\tau, f(\omega_1, \ldots, \omega_n, v))$$

Axioms in $\mathcal{L}(FL)$ ensure that f is functional:

$$Holds(\tau, f(\omega_1, \ldots, \omega_n, v_1)) \wedge Holds(\tau, f(\omega_1, \ldots, \omega_n, v_2)) \rightarrow v_1 = v_2 \tag{15.1}$$

$$\exists v\, Holds(\tau, f(\omega_1, \ldots, \omega_n, v)) \tag{15.2}$$

Action Occurrences and Effects

Action occurrences and the effects of actions are represented as follows. The $\mathcal{L}(ND)$ action occurrence statement

$$[\tau_1, \tau_2]\, a(\omega_1, \ldots, \omega_n)$$

represents that action $a(\omega_1, \ldots, \omega_n)$ occurs between timepoints τ_1 and τ_2. The $\mathcal{L}(ND)$ action law schema

$$[\tau_1, \tau_2]\, \alpha \rightsquigarrow ([\tau_1]\, \gamma \rightarrow [\tau_1, \tau_2]\, \phi := v)$$

represents that if action α occurs between τ_1 and τ_2 and condition γ is true at τ_1, then the value of fluent ϕ will be v starting at τ_2. The value of ϕ is unchanged at τ_1 and is permitted to vary at timepoints $\tau_1 + 1$ to $\tau_2 - 1$.

If $\tau_2 = \tau_1 + 1$, then the action is called a single-step action. In this case, the meaning is the same as in the discrete event calculus:

- TAL $[\tau, \tau + 1]\, \alpha$ is the same as DEC $Happens(\alpha, \tau)$
- TAL $[\tau, \tau + 1]\, \alpha \rightsquigarrow ([\tau]\, \gamma \rightarrow [\tau, \tau + 1]\, \phi := T)$ is the same as DEC $Initiates(\alpha, \phi, \tau)$
- TAL $[\tau, \tau + 1]\, \alpha \rightsquigarrow ([\tau]\, \gamma \rightarrow [\tau, \tau + 1]\, \phi := F)$ is the same as DEC $Terminates(\alpha, \phi, \tau)$

Action occurrences statements and action law schemata are translated into $\mathcal{L}(FL)$ schedule statements as follows. For each action occurrence statement $[\tau_1, \tau_2]\ a(\omega_1, \ldots, \omega_n)$ and corresponding action law schema $[t_1, t_2]\ a(x_1, \ldots, x_n) \rightsquigarrow \Delta$, we form a $\mathcal{L}(ND)$ reassignment formula $[\tau_1, \tau_2]\ \Delta'$, where Δ' is Δ with τ_1 substituted for t_1, τ_2 substituted for t_2, ω_1 substituted for x_1, ω_2 substituted for x_2, and so on. For example, from

$$[1, 2]\ WakeUp(Nathan)$$

and

$$[t_1, t_2]\ WakeUp(a) \rightsquigarrow awake(a) := T$$

we form the reassignment formula

$$[1, 2]\ awake(Nathan) := T \tag{15.3}$$

We then translate the reassignment formula into $\mathcal{L}(FL)$. The translation of a reassignment formula $[\tau_1, \tau_2]\ f(\omega_1, \ldots, \omega_n) := v$ is the schedule statement

$$\begin{gathered} Holds(\tau_2, f(\omega_1, \ldots, \omega_n, v)) \wedge \\ \forall t, d\ (t_1 < t \leq t_2 \rightarrow Occlude(t, f(\omega_1, \ldots, \omega_n, d))) \end{gathered}$$

For example, the translation of (15.3) is

$$Holds(2, awake(Nathan)) \wedge Occlude(2, awake(Nathan))$$

The predicate $Occlude(t, f)$ represents that the truth value of fluent f may change between timepoints $t - 1$ and t, which is the same as $ReleasedAt(f, t)$ in the event calculus. This constraint is enforced by the following nochange axiom of $\mathcal{L}(FL)$, which states that if a fluent f is not occluded at timepoint $t + 1$, then the truth value of f does not change from timepoint t to $t + 1$:

$$\neg Occlude(t + 1, f) \rightarrow Holds(t, f) \equiv Holds(t + 1, f) \tag{15.4}$$

This is equivalent to the conjunction of the axioms

$$Holds(t, f) \wedge \neg Occlude(t + 1, f) \rightarrow Holds(t + 1, f) \tag{15.5}$$

and

$$\neg Holds(t, f) \wedge \neg Occlude(t + 1, f) \rightarrow \neg Holds(t + 1, f) \tag{15.6}$$

These axioms resemble axioms of the discrete event calculus. Axiom (15.5) resembles DEC5:

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

Axiom (15.6) resembles DEC6:

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

Scenario Descriptions

A TAL scenario description is given by

$$CIRC[\Sigma; Occlude] \wedge \Psi \wedge \Gamma \wedge TAL \tag{15.7}$$

where Σ is a conjunction of schedule statements derived from action occurrence statements and action law schemata; Ψ is a conjunction of domain constraints, or state constraints; Γ is a conjunction of observation statements; and TAL is a conjunction of foundational axioms and the nochange axiom (15.4). We use a notation that highlights the similarity of TAL scenario descriptions and event calculus domain descriptions (see Section 2.7). The TAL notation for Σ is Γ_{SCD}, Ψ is Γ_{DOMC}, Γ is Γ_{OBS}, and TAL $= \Gamma_{FA} \wedge \Gamma_{NCG}$, where $\Gamma_{NCG} =$ (15.4). The scenario description may entail various formulas, such as *Holds* formulas, as in the event calculus.

Discussion

Like the event calculus, TAL supports actions with duration, the commonsense law of inertia, dynamic release from the commonsense law of inertia, concurrent events, context-sensitive effects, indirect effects, and nondeterministic effects. Unlike the event calculus, TAL supports functional fluents. As described in Section 15.1, functional fluents can be represented in the event calculus using relational fluents and state constraints. These state constraints are similar to the $\mathcal{L}(FL)$ axioms (15.1) and (15.2). TAL supports a type of fluent called a *durational fluent*, which is a fluent whose value is always a default value except when the fluent is occluded.

Several features of the event calculus are not supported by TAL: continuous change, continuous time, time less than zero, and triggered events (because action occurrence statements have no provision for conditions). Event occurrences are represented explicitly in the event calculus using *Happens*; in TAL,

event occurrences are represented explicitly in $\mathcal{L}(ND)$ but not in $\mathcal{L}(FL)$. (In some versions of TAL, a predicate *Occurs* is introduced.)

15.3 Action Languages

Instead of using the general-purpose language of first-order logic for reasoning about action, specialized action languages can be developed. The original action languages were STRIPS and ADL; a number of action languages, such as $\mathcal{A}$, $\mathcal{AR}_0$, $\mathcal{A}_C$, $\mathcal{C}$, $\mathcal{C}+$, and $\mathcal{E}$, have since been developed.

The simple action language $\mathcal{A}$ was introduced by Michael Gelfond and Vladimir Lifschitz. In $\mathcal{A}$, action effects are represented using expressions of the form

$$a \textbf{ causes } f \textbf{ if } f_1, \ldots, f_n$$

where a is an action, and f, f_1, ..., f_n are (possibly negated) fluents. $\mathcal{AR}_0$ extends $\mathcal{A}$ for indirect effects and nondeterminism. $\mathcal{A}_C$ extends $\mathcal{A}$ for concurrent actions. $\mathcal{C}$ is based on a causal theory of actions. $\mathcal{C}+$ is an extension of $\mathcal{C}$. $\mathcal{E}$ is similar to the event calculus, and a mapping between the event calculus and $\mathcal{E}$ has been described. In this section, we discuss action language $\mathcal{C}+$.

15.3.1 $\mathcal{C}+$

$\mathcal{C}+$ was developed by Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Like the event calculus and TAL, $\mathcal{C}+$ uses linear time. Like the discrete event calculus, $\mathcal{C}+$ uses discrete time. Unlike the event calculus, the situation calculus, and TAL, which are based on first-order and sometimes second-order logic, $\mathcal{C}+$ is based on propositional logic.

Reasoning problems in $\mathcal{C}+$ can be transformed into SAT problems, as can reasoning problems of the discrete event calculus. The SAT-based approach used in the discrete event calculus derives from the SAT-based approach used in $\mathcal{C}+$.

Syntax of $\mathcal{C}+$

We start with a *signature*, which consists of a set of constants, and a domain $dom(c)$ associated with each constant c. The domain of a Boolean constant is $\{\mathbf{t}, \mathbf{f}\}$. If c is a constant and $v \in dom(c)$, then $c = v$ is a $\mathcal{C}+$ *atom*. A $\mathcal{C}+$ *formula* is defined as follows:

- An atom is a formula.
- The symbols $\bot$ (*false*) and $\top$ (*true*) are formulas (zero-place connectives).
- If f_1 is a formula, and f_2 is a formula, then $\neg f_1$, $f_1 \wedge f_2$, and $f_1 \vee f_2$ are formulas.

- If c is a Boolean constant, then $c = \mathbf{t}$ is abbreviated c and $c = \mathbf{f}$ is abbreviated $\neg c$.

The set of fluents and the set of actions are disjoint subsets of the set of constants. Fluents are further divided into simple fluents and statically determined fluents. A *fluent formula* is a formula whose constants are fluents, and an *action formula* is a formula whose constants are actions.

Although $\mathcal{C}+$ is based on propositional logic, a first-order notation is often used. We may define a set of constants $p(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are taken from finite sets. For example, we may represent the location of an object using the set of constants $InRoom(x)$, where $x \in \{Nathan, Book\}$.

A $\mathcal{C}+$ *causal law* is an expression of the form

$$\textbf{caused } f \textbf{ if } g \textbf{ after } h$$

or

$$\textbf{caused } f \textbf{ if } g$$

where f, g, and h are formulas. A number of abbreviations for causal laws are defined.

- The effects of actions are represented using expressions of the form

  $$a \textbf{ causes } f \textbf{ if } c$$

  where a is an action formula, and f and c are fluent formulas. This is an abbreviation for **caused** f **if** $\top$ **after** $a \wedge c$. The formula f cannot contain statically determined fluents. The condition c enables context-sensitive effects of actions to be represented.
- If f is a simple fluent, then the expression

  $$\textbf{inertial } f$$

  specifies that f is subject to the commonsense law of inertia at all times. This is an abbreviation for **caused** $f = v$ **if** $f = v$ **after** $f = v$, for each $v \in dom(f)$.
- Action preconditions are represented using expressions of the form

  $$\textbf{nonexecutable } a \textbf{ if } c$$

  where a is an action formula, and c is a fluent formula. This stands for **caused** $\bot$ **if** $\top$ **after** $a \wedge c$.

- Triggered actions are represented using causal laws of the form

  caused a **if** c

  where a is an action formula, and c is a fluent formula.
- If a is an action, then the expression

  exogenous a

  specifies that a may or may not occur at each timepoint. This stands for **caused** $a = v$ **if** $a = v$, for each $v \in dom(a)$. If **exogenous** a is not specified, then in order for a to occur it must be caused to occur.
- State constraints are represented using expressions of the form

  constraint c

  where c is a fluent formula. This stands for **caused** $\bot$ **if** $\neg c$.

$\mathcal{C}+$ treats indirect effects in an elegant way. They are represented using causal laws of the form

caused f **if** g

where f and g are fluent formulas. Such a causal law represents that f is an indirect effect of any actions that bring about g. For example, we may represent that if an agent is holding an object, and the agent is in a room, then the object will also be in that room:

$$\textbf{caused } \mathit{InRoom}(o) = r \textbf{ if } \mathit{Holding}(a) = o \wedge \mathit{InRoom}(a) = r$$

The closest analog to this in the event calculus is the pair of effect constraints:

$$\begin{gathered}\mathit{HoldsAt}(\mathit{Holding}(a, o), t) \wedge \\ \mathit{Initiates}(e, \mathit{InRoom}(a, r), t) \Rightarrow \\ \mathit{Initiates}(e, \mathit{InRoom}(o, r), t)\end{gathered}$$

$$\begin{gathered}\mathit{HoldsAt}(\mathit{InRoom}(a, r), t) \wedge \\ \mathit{Initiates}(e, \mathit{Holding}(a, o), t) \Rightarrow \\ \mathit{Initiates}(e, \mathit{InRoom}(o, r), t)\end{gathered}$$

But these formulas do not allow for the possibility that $\mathit{Holding}(a) = o$ or $\mathit{InRoom}(a) = r$ may be the result of interacting concurrent events.

We have so far described the syntax of $\mathcal{C}+$. The semantics of a $\mathcal{C}+$ theory is given by a translation into a causal theory. We first define the syntax and semantics of causal theories, and then specify the translation of a $\mathcal{C}+$ theory into a causal theory.

Syntax of Causal Theories

We have a signature, as in $\mathcal{C}+$. Causal theory atoms and formulas are as in $\mathcal{C}+$. A *causal rule* is an expression of the form

$$f_1 \Leftarrow f_2$$

where f_1 and f_2 are causal theory formulas. This represents that f_1 has a cause if f_2 is true. f_1 is called the head of the causal rule, and f_2 is called the body of the causal rule. A *causal theory* consists of a set of causal rules.

Semantics of Causal Theories

An *interpretation* I for a signature is a mapping from each constant c of the signature to an element of $dom(c)$. We define what it means for a formula ψ to be true in an interpretation I, written $I \models \psi$, as follows:

- If c is a constant and $v \in dom(c)$, then $I \models c = v$ if and only if $I(c) = v$.
- It is not the case that $I \models \bot$.
- It is the case that $I \models \top$.
- If α is a formula, then $I \models \neg\alpha$ if and only if it is not the case that $I \models \alpha$.
- If α and β are formulas, then $I \models \alpha \wedge \beta$ if and only if $I \models \alpha$ and $I \models \beta$.
- If α and β are formulas, then $I \models \alpha \vee \beta$ if and only if $I \models \alpha$ or $I \models \beta$.

An interpretation I is a *model* of a set of formulas Φ, written $I \models \Phi$, if and only if $I \models \psi$ for every $\psi \in \Phi$.

The *reduct* of a causal theory T relative to an interpretation I, written $reduct(T, I)$, is the set of every head of a causal rule $r \in T$ such that the body of r is true in I. An interpretation I is a *model* of a causal theory T, written $I \models T$, if and only if I is the unique model of the set of formulas $reduct(T, I)$. That is, there must be exactly one model of $reduct(T, I)$, and that model must equal I. A causal theory T *entails* a formula ψ, written $T \models \psi$, if and only if, for every I such that $I \models T$, we have $I \models \psi$.

Reduction to Propositional Logic

A causal theory T is *definite* if and only if (1) the head of every causal rule in T is an atom or $\bot$ and (2) no atom is the head of an infinite number of causal rules of T. If a causal theory is definite, its semantics may be restated in terms of a reduction to propositional logic.

From a causal theory, we form a conjunction of propositional formulas, called the completion of the causal theory, as follows: First, for each atom a, we add to

the conjunction

$$a \equiv g_1 \vee \cdots \vee g_m$$

where $m \geq 0$ and $g_1, \ldots, g_m$ are the bodies of the causal rules in which a appears as the head. Second, for each causal rule of the form $\bot \Leftarrow f$, we add to the conjunction

$$\neg f$$

Third, we add to the conjunction

$$\left(\bigvee_v c = v\right) \wedge \left(\bigwedge_{v_1, v_2} v_1 = v_2 \vee \neg(c = v_1 \wedge c = v_2)\right)$$

for all constants c, where v, v_1, and v_2 range over $dom(c)$, so that each constant equals exactly one value.

Translation from C+ to Causal Theories

We define the length-N translation of a $C+$ theory into a causal theory, where N is a positive integer. A $C+$ theory can be viewed as representing a *transition system*, or a directed graph whose nodes represent states, and whose links represent actions or sets of concurrently executed actions. There is a one-to-one correspondence between the models of the length-N translation of a $C+$ theory and the paths of length N through the transition system represented by the theory.

We define a set of *timepoints* $\{0, 1, 2, \ldots, N\}$. The set of constants in the causal theory is the set of all pairs $t : c$, where t is a timepoint, and c is a constant in the $C+$ theory. (For simplicity, our discussion ignores the fact that actions do not occur at timepoint N.) The domain associated with each $t : c$ in the causal theory is the domain associated with c in the $C+$ theory. The *translation* of a $C+$ formula f at timepoint t, written $t : f$, is a causal theory formula obtained by substituting $t : c$ for every constant c in f.

If f is a fluent, $v \in dom(f)$, and t is a timepoint, then $t : f = v$ represents that f has the value v at t.

If a is a Boolean action and t is a timepoint, then $t : a$ represents that a occurs at t.

In order to expose the meanings of the $C+$ expressions already mentioned, we specify here the translation of each expression separately. The translation may be defined more simply in terms of just the two causal laws **caused** f **if** g **after** h and **caused** f **if** g.

- A $\mathcal{C}+$ expression representing the effects of actions

  $$a \textbf{ causes } f \textbf{ if } c$$

  is translated into a set of causal rules

  $$t+1 : f \Leftarrow (t : a) \wedge (t : c)$$

  for each $t \in \{0, 1, 2, \ldots, N-1\}$.
- A $\mathcal{C}+$ expression of the form

  $$\textbf{inertial } f$$

  is translated into a set of causal rules

  $$t+1 : f = v \Leftarrow (t+1 : f = v) \wedge (t : f = v)$$

  for each $v \in dom(f)$ and $t \in \{0, 1, 2, \ldots, N-1\}$.
- We also add causal rules stating that the initial values of simple fluents are arbitrary. For each simple fluent f and $v \in dom(f)$ in the $\mathcal{C}+$ theory, we add

  $$0 : f = v \Leftarrow 0 : f = v \tag{15.8}$$

  For a statically determined fluent, (15.8) is not added. The value of a statically determined fluent at a timepoint t is strictly determined by the values of other fluents and actions at t. Statically determined fluent constants can be used for default reasoning.
- A $\mathcal{C}+$ precondition

  $$\textbf{nonexecutable } a \textbf{ if } c$$

  is translated into a set of causal rules

  $$\bot \Leftarrow (t : a) \wedge (t : c)$$

  for each $t \in \{0, 1, 2, \ldots, N\}$
- A $\mathcal{C}+$ expression representing a triggered action

  $$\textbf{caused } a \textbf{ if } c$$

  is translated into a set of causal rules

  $$t : a \Leftarrow t : c$$

  for each $t \in \{0, 1, 2, \ldots, N\}$.

- A $\mathcal{C}+$ expression of the form

  exogenous a

  is translated into a set of causal rules

  $$t : a = v \Leftarrow t : a = v$$

  for each $v \in dom(a)$ and $t \in \{0, 1, 2, \ldots, N\}$.
- A $\mathcal{C}+$ state constraint

  constraint c

  is translated into a set of causal rules

  $$\bot \Leftarrow \neg t : c$$

  for each $t \in \{0, 1, 2, \ldots, N\}$.
- A $\mathcal{C}+$ expression representing an indirect effect

  caused f **if** g

  is translated into a set of causal rules

  $$t : f \Leftarrow t : g$$

  for each $t \in \{0, 1, 2, \ldots, N\}$.

Discussion

$\mathcal{C}+$ supports many of the same features as the event calculus. Like the event calculus, it supports the commonsense law of inertia, concurrent events with cumulative and canceling effects, context-sensitive effects, indirect effects, nondeterministic effects (via **may cause**), preconditions, state constraints, and triggered events. Unlike the event calculus, $\mathcal{C}+$ does not support continuous change or continuous time.

15.4 The Fluent Calculus

The fluent calculus was developed by Michael Thielscher, Steffen Hölldobler, and Josef Schneeberger. The fluent calculus adds the notion of states to the notions of actions, fluents, and situations of the situation calculus.

15.4.1 States

A *state* represents a set of fluents and is defined inductively as follows:

- $\emptyset$ is a state.
- If f is a fluent, then f is a state.
- If z_1 and z_2 are states, then $z_1 \circ z_2$ is a state.
- Nothing else is a state.

The function $\circ$ is associative and commutative. The expression $Holds(f, z)$ represents that fluent f is true in state z and is defined as follows:

$$Holds(f, z) \stackrel{\text{def}}{\equiv} \exists z' \, (z = f \circ z')$$

Situations and states are related as follows. The function $State(s)$ represents the state associated with situation s. The expression $Holds(f, s)$ represents that fluent f is true in situation s and is defined as:

$$Holds(f, s) \stackrel{\text{def}}{\equiv} Holds(f, State(s))$$

As in the situation calculus, $Do(a, s)$ represents the situation that results from performing action a in situation s and the initial situation is called S_0.

15.4.2 Plus and Minus Macros

The macros + (plus sign) and − (minus sign) are used to specify the addition and removal of fluents from states:

$$z_1 - \emptyset = z_2 \stackrel{\text{def}}{\equiv} z_2 = z_1$$

$$z_1 - f = z_2 \stackrel{\text{def}}{\equiv} (z_2 = z_1 \vee z_2 \circ f = z_1) \wedge \neg Holds(f, z_2)$$

$$z_1 - (f_1 \circ f_2 \circ \ldots \circ f_n) = z_2 \stackrel{\text{def}}{\equiv} \exists z \, (z_1 - f_1 = z \wedge z - (f_2 \circ \ldots \circ f_n) = z_2)$$

$$z_2 = (z_1 - \vartheta^-) + \vartheta^+ \stackrel{\text{def}}{\equiv} \exists z \, (z_1 - \vartheta^- = z \wedge z_2 = z \circ \vartheta^+)$$

The states ϑ^- and ϑ^+ represent sets of fluents.

15.4.3 State Update Axioms

The effects of each action a are represented using *state update axioms* for a, which are of the form:

$$Poss(a, s) \wedge \Delta(s) \supset State(Do(a, s)) = (State(s) - \vartheta^-) + \vartheta^+$$

where $Poss(a, s)$ represents that a is possible in situation s, $\Delta(s)$ is a formula containing s, the state ϑ^- represents the set of fluents terminated by a under the conditions $\Delta(s)$, and the state ϑ^+ represents the set of fluents initiated by a under the conditions $\Delta(s)$. Notice that for a given action a, if we wish to be able to determine the effects of a in all cases for which $Poss(a, s)$ is true, then the conditions $\Delta(s)$ of the state update axioms for a must exhaust all those cases. For example, the following state update axiom for the action $WakeUp(p)$ states that, if a person wakes up, then the person will no longer be asleep and the person will be awake:

$$State(Do(WakeUp(p), s)) = (State(s) - Asleep(p)) + Awake(p)$$

15.4.4 Nondeterministic Effects

The effect of flipping a coin is represented using the following state update axiom:

$$State(Do(Flip, s)) = State(s) + Heads \vee$$
$$State(Do(Flip, s)) = State(s) - Heads$$

15.4.5 Concurrent Actions

The fluent calculus deals with concurrent actions with cumulative or canceling effects by introducing concurrent actions. A *concurrent action* represents a set of actions and is defined inductively as follows:

- The empty action ϵ is a concurrent action.
- If a is an action, then a is a concurrent action.
- If c_1 and c_2 are concurrent actions, then $c_1 \circ c_2$ is a concurrent action.
- Nothing else is a concurrent action.

The function $\circ$ on concurrent actions is associative and commutative. The expression $HoldsIn(c_1, c_2)$ represents that c_1 is contained in c_2, and is defined as follows:

$$HoldsIn(c_1, c_2) \stackrel{\text{def}}{\equiv} \exists c' (c_2 = c_1 \circ c')$$

The effects of concurrent actions are then specified using state update axioms. For example, the following state update axioms represent that in order for a door

to open, it must be pushed at the same time that its latch is turned.

$$Poss(Turn \circ c, s) \wedge \neg HoldsIn(Push, c) \supset$$
$$State(Do(Turn \circ c, s)) = State(Do(c, s))$$

$$Poss(Push \circ c, s) \wedge \neg HoldsIn(Turn, c) \supset$$
$$State(Do(Push \circ c, s)) = State(Do(c, s))$$

$$Poss(Turn \circ Push \circ c, s) \supset$$
$$State(Do(Turn \circ Push \circ c, s)) = State(Do(c, s)) + Open$$

This is represented in the event calculus as follows:

$$Happens(Turn, t) \Rightarrow Initiates(Push, Open, t)$$

(The representation of cumulative and canceling effects in the event calculus is discussed in section 8.2.)

15.4.6 Discussion

Like the event calculus, the fluent calculus supports the commonsense law of inertia, concurrent events with cumulative and canceling effects, context-sensitive effects, indirect effects, nondeterministic effects, and preconditions. Methods have also been proposed for handling continuous change and triggered events in the fluent calculus. The treatment of indirect effects in the fluent calculus is described in the Bibliographic Notes of Chapter 6.

15.5 Discussion and Summary

Many of the best features of logics for commonsense reasoning have been incorporated into the classical logic event calculus. Yet the event calculus retains a certain simplicity compared to other logics. Figure 15.1 outlines the relationships between the logics and indicates some of the origins of features of the event calculus.

The situation calculus came first and has had a major influence on all of the logics. The notion of fluents, use of frame axioms, use of reification, and distinction between primitive and derived fluents are all from the situation calculus. The treatment of concurrent events with cumulative or canceling effects was first perfected in the situation calculus and then integrated into the event calculus.

The classical logic event calculus derives from the original logic programming version of the event calculus, which introduced events, event occurrences, and fluents that are initiated and terminated. The treatment of continuous change and

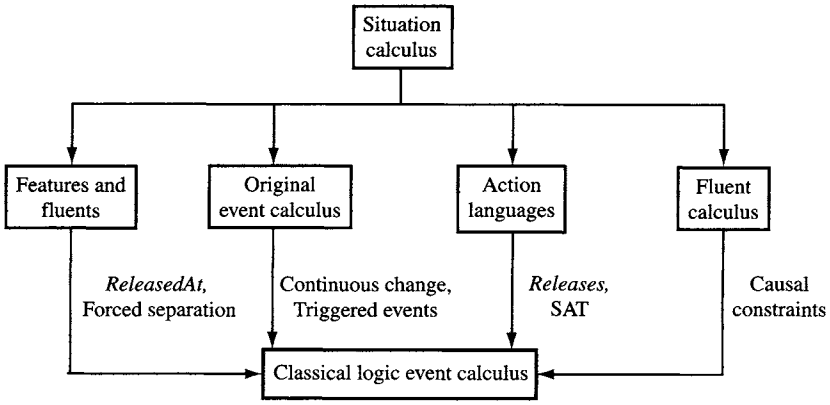


Figure 15.1 Logics for commonsense reasoning.

Table 15.2 Models of time used by logics

	Branching time	*Linear time*
Discrete time	Situation calculus Fluent calculus	TAL *C+* Discrete event calculus
Continuous time		Event calculus

triggered events in the classical logic event calculus derives from work performed in the original event calculus.

The features and fluents framework has had several influences on the classical logic event calculus. The *ReleasedAt* predicate of the classical logic event calculus derives from the occlusion construct of features and fluents. The technique of forced separation, in which event calculus axioms are outside the scope of any circumscription, derives from the filtered preferential entailment or filtering of features and fluents.

Several features of the classical logic event calculus derive from action languages. The *Releases* predicate of the event calculus is taken from action languages. The use of SAT solvers to solve event calculus reasoning problems was inspired by the use of SAT solvers in action languages.

Work on the instantaneous propagation of interacting indirect effects was first performed in the fluent calculus. This work led to the introduction of causal constraints into the classical logic event calculus.

Logics for commonsense reasoning can model time as branching or linear and as discrete or continuous. Table 15.2 shows which logics use which models of time.

Bibliographic Notes

The Situation Calculus

The version of the situation calculus we describe is that of Reiter (2001). Based on previous proposals (Reiter, 1993; Lin and Reiter, 1994; Pinto, 1998b), Pirri and Reiter (1999, p. 328) and Reiter (2001, p. 50) provide the following four foundational axioms for the situation calculus:

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2 \tag{15.9}$$

$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s) \tag{15.10}$$

$$\neg(s \sqsubset S_0) \tag{15.11}$$

$$s \sqsubset do(a, s') \equiv s = s' \vee s \sqsubset s' \tag{15.12}$$

Axiom (15.10) is a formula of second-order logic, an induction axiom, which allows us to prove general properties of situations and limits the set of situations to the smallest set S such that (1) $S_0 \in S$ and (2) $s \in S$ and a is an action implies $do(a, s) \in S$ (Reiter, 1993). Axiom (15.12) defines the ordering relation $\sqsubset$ on situations.

Axioms (15.9) and (15.11) restrict the graph of situations to a tree; that is, these axioms identify a situation with the sequence of actions that led to it. The original situation calculus of McCarthy and Hayes (1969) does not contain this restriction. They define a situation as "the complete state of the universe at an instant of time" (p. 477). Shanahan (1997b, p. 43) calls (15.9) and (15.11) the axioms of arboreality. He provides the following alternative, called the situation-state axiom, which identifies a situation with the fluents that are true in it:

$$s1 = s2 \leftrightarrow \forall f\ [Holds(f, s1) \leftrightarrow Holds(f, s2)]$$

Thielscher (1999b) takes a third approach: He identifies a situation with a sequence of actions and adds a separate notion of state. He introduces a function *State* that maps a situation to a state, as discussed in Section 15.4.

The equivalence of versions of the situation calculus and the event calculus was proved by Kowalski and Sadri (1997), based on their previous work (Kowalski and Sadri, 1994). Gelfond et al. (1991) discuss some of the apparent limitations of the situation calculus and outline ways of overcoming them. In order to deal with indirect effects in the situation calculus, they may be represented explicitly as effect axioms (Lin and Reiter, 1994, sec. 5; Reiter, 2001, pp. 401–402). Other methods are discussed in the Bibliographic Notes of Chapter 6.

Continuous Change

Gelfond et al. (1991, pp. 170–171) sketch a method for handling continuous change in the situation calculus. They introduce a function *Time* that maps situations to real numbers. Situations have the cardinality of the real numbers. Changing

quantities are represented using fluents. Building on previous work (Pinto and Reiter, 1993), Pinto (1994, chap. 6) presents a way of representing continuous change and triggered events in the situation calculus. He argues (pp. 3, 67) against the approach of Gelfond et al. involving a continuum of situations. Instead, he proposes to represent episodes of continuous change within single situations. He introduces the function *start*(s), which represents the start time of a situation s, and *end*(s, a), which represents the end time of a situation s if action a is performed in s (p. 29), where a time is a nonnegative real number.

Pinto and Reiter (1993) use a predicate *start*(s, t), which represents that t is the start time of s. Instead of representing continuous change using fluents, Pinto (1994) introduces the notion of a continuous parameter (p. 68). He then uses *holds*$(p \doteq f, s)$ to represent that in situation s, the value of parameter p is given by function of time f. Thus, starting at time *start*(s), the value of p at time t equals $f(t)$. Actions initiate and terminate episodes of continuous change. An effect axiom of the form

$$Poss(a, s) \supset holds(p \doteq f, do(a, s))$$

represents that, if a is performed in s, it initiates an episode of continuous change within the situation $do(a, s)$ in which p is described by f.

Actual Situations and Events

Pinto and Reiter (1993) and Pinto (1994, sec. 4.2) add actual situations and actual events to the situation calculus. The predicate *actual*(s) represents that s is an actual situation. The initial situation is actual. Every actual situation has at most one successor situation that is actual. The predecessor situation of an actual situation is actual. The predicate *occurs*(a, s) represents that action a actually occurs in situation s and is defined by (p. 29):

$$occurs(a, s) \equiv actual(do(a, s))$$

Thus, actions occur along a single time line, as in the event calculus. If *occurs*(a, s), then a occurs at time *start*$(do(a, s))$. That is, an action a performed in situation s is considered to occur at the start time of the successor situation $do(a, s)$.

Natural Actions

Pinto (1994, p. 68) introduces the notion of a *natural action*, or action that is not performed by an agent but rather occurs as a result of physical laws, such as a falling ball hitting the floor. The predicate $\Pi_t(a, s, t)$ represents that natural action a is possible at time t in situation s. A natural action a is possible in a situation s—that is, *Poss*(a, s)—if and only if a is possible at *end*(s, a) in s and no other action is possible at a time greater than *pstart*(s) and less than or equal to *pend*(s, a) in s. Only one natural action is possible in each situation (p. 75). Thus, a sequence of possible natural actions uses linear time rather than branching time

(p. 85). If a natural action a is possible in a situation s and a nonnatural action does not occur before $end(s, a)$, then a must occur in s (p. 75). For example, the natural action *Fall* occurs in situation S_1, which gives rise to successor situation S_2. This action initiates an episode of continuous change within S_2 in which the height of the falling object is given by an equation of free-fall motion. A natural action *HitGround* occurs in S_2, which gives rise to successor situation S_3. This is reminiscent of the operation of *Trajectory* in the event calculus.

Reiter (1996) presents some variations on Pinto's proposal. The last argument of every action predicate symbol is a time, which is a real number. The function $time(a)$ represents the time of the action a. The predicate $\Pi_t(a, s, t)$ is dispensed with because the time of a is embedded in a and whether a natural action is possible at a time is represented directly using *Poss*.

Narrative Time Line

R. Miller and Shanahan (1994) introduce a narrative time line consisting of the nonnegative real numbers into the situation calculus. The predicate $Happens(a, t)$ represents that action a actually occurs at time t. The function $State(t)$ represents the actual situation at time t. If an action a occurs at time t_1, $State(t_1) = s_1$, and no other actions occur between t_1 and $t_2 > t1$, then $State(t_2) = do(a, s_1)$. Thus, times greater than the time of occurrence of an action correspond to the successor situation given that action. All times before the first action occurs correspond to the initial situation S_0.

Continuous Change and Triggered Events

Using the approach of R. Miller and Shanahan (1994), and building on work by Baker (1991), R. Miller (1996) proposes a way of representing continuous change and triggered events in the situation calculus. He introduces valuations and continuously varying parameters. A *valuation* is a function from a parameter to a real number. The predicate $Triggers(v, s, a)$ represents that action a is triggered given valuation v and situation s. For example, we may represent that, if a sink is filling, then the water will overflow when its height reaches a certain level:

$$Triggers(v, s, a) \leftrightarrow$$
$$v(Height) = H \wedge Holds(Filling, s) \wedge a = Overflow$$

An action occurs if and only if it is performed or it is triggered.

Release from Inertia

A proposal for permanently releasing a fluent from the commonsense law of inertia in the situation calculus is provided by Lifschitz (1990b, p. 370). He introduces the predicate $Frame(f)$, which represents that fluent f is always subject to the law; $\neg Frame(f)$ represents that f is always released from the law.

Concurrent Events

Ways of dealing with concurrent events in the situation calculus are discussed in the Bibliographic Notes of Chapter 8.

Nondeterminism

Lin (1996) treats nondeterminism in the situation calculus. Mateus, Pacheco, and Pinto (2002) present a probabilistic version of the situation calculus. See also the Bibliographic Notes of Chapter 9.

Knowledge Machine

The Knowledge Machine (KM) system (P. Clark and Porter, 2004) contains a situation mechanism (P. Clark and Porter, 2000) that implements a version of the situation calculus. KM allows us to specify whether each fluent should or should not be subject to the commonsense law of inertia. A fluent subject to this law persists from a situation to a successor situation, provided that it is consistent for it to persist. The effects of actions are represented in KM using preconditions, add lists, and delete lists, along the lines of STRIPS (Fikes and Nilsson, 1971). The KM situation mechanism performs temporal projection. It does not perform abduction or postdiction, and it does not support nondeterminism, continuous change, or actions with duration.

The Features and Fluents Framework

The features and fluents framework was introduced by Sandewall (1989a, 1989b, 1991, 1994). The framework has many facets. Sandewall (1994) formalizes a semantics for worlds inhabited by agents, called inhabited dynamical systems (IDS). He defines the syntax and semantics of several varieties of a logic called discrete fluent logic (DFL). This logic is used to define *chronicles*, which specify a set of objects, the effects of events, event occurrences, and observed states. (Chronicles are similar to event calculus domain descriptions, discussed in Section 2.7.)

Assessment of Logics

Sandewall (1994) presents the following systematic methodology for assessing logics for reasoning about action and change (p. 65). First, we formally define a set of chronicle classes. For example, Sandewall defines a chronicle class $\mathcal{K}$**sp-IAd**. Chronicles of this class involve accurate knowledge ($\mathcal{K}$), complete knowledge of the initial state (**s**), pure prediction (**p**), context-free inertia and integer time (**I**), alternative effects of events (**A**), and deterministic change (**d**). Benchmark commonsense reasoning problems in $\mathcal{K}$**sp-IAd** include the Yale shooting scenario (Hanks and McDermott, 1987, pp. 387–390), the Stockholm delivery scenario (Sandewall, 1994, pp. 31–33), and the ferryboat connection scenario (Sandewall, 1994, pp. 154–155).

Second, for each chronicle class, we formally define the set of intended models for any chronicle of the class. Specifically, we define a model set function *Mod* from chronicles to sets of intended IDS models (p. 182), and a coercion function **Ci** from sets of IDS models to sets of DFL models (p. 185).

Third, we define *entailment methods*, or functions from chronicles to sets of DFL models (p. 195). Sandewall defines a number of such methods, including prototypical global minimization (PGM), original chronological minimization (OCM), prototypical chronological minimization (PCM), chronological minimization of occlusion and change (CMOC), chronological assignment and minimization of occlusion and change (CAMOC), global minimization of occlusion with nochange premises (GMON), and pointwise minimization of occlusion with nochange premises (PMON, the basis of TAL).

Finally, we prove that one or more entailment methods are correctly applicable (p. 201) to one or more chronicle classes. An entailment method is *correctly applicable* to a chronicle class if and only if, for every chronicle c of the class, the set of intended DFL models $\mathbf{Ci}(Mod(c))$ equals the set of DFL models produced by applying the entailment method to c.

Sandewall focuses on the $\mathcal{K}$-**IA** set of chronicle classes (pp. 159–168). He assesses the applicability of 12 entailment methods relative to the classes (pp. 201–211, 228–230, 244–252). The CAMOC and PMON entailment methods are found to be correctly applicable to $\mathcal{K}$-**IA**. Sandewall (1996) assesses methods for dealing with indirect effects of events. Brandano (2001) makes some progress on assessing the event calculus using Sandewall's methodology.

Temporal Action Logics

Doherty and Łukaszewicz (1994) and Doherty (1994) recast features and fluents in first-order logic. TAL was previously called PMON; the most complete description of TAL (PMON) is provided by Doherty (1996) and a later treatment of TAL is provided by Doherty et al. (1998). Our description is a composite of Doherty (1996) and Doherty et al. (1998). Indirect effects may be represented in TAL using domain constraints or using the causal constraints introduced by Gustafsson and Doherty (1996). Karlsson, Gustafsson, and Doherty (1998) treat delayed effects of events in TAL. Karlsson and Gustafsson (1997) present TAL-C, an extension of TAL that supports concurrent events with cumulative and canceling effects. Kvarnström (2001) presents the VITAL program, a tool for performing commonsense reasoning using TAL. VITAL works by using "constraint propagation to find all possible models for a given narrative" (p. 41). TAL is used by the TALplanner planning system (Kvarnström and Doherty, 2000b; Kvarnström, 2005).

Action Languages

Pednault (1989) introduced ADL, an action description language with a syntax similar to that of STRIPS (Fikes and Nilsson, 1971). Gelfond and Lifschitz (1993) introduced action language $\mathcal{A}$, Kartha and Lifschitz (1994) introduced

$\mathcal{AR}_0$, Baral and Gelfond (1997) introduced $\mathcal{A}_C$, and Kakas and Miller (1997a, 1997b) introduced action language $\mathcal{E}$. R. Miller and Shanahan (2002) describe a mapping between $\mathcal{E}$ and the classical logic event calculus. Vladimir Lifschitz and Erik Sandewall have debated the merits of action languages versus classical logic (Lifschitz 1997a, 1997b; Sandewall 1997a, 1997b). E. Giunchiglia and Lifschitz (1998) introduced action language $\mathcal{C}$, based on the language of causal theories of McCain and Turner (1997). E. Giunchiglia et al. (2004) introduced action language $\mathcal{C}+$. Geffner (1990) introduced the distinction between what is true and what has a cause. E. Giunchiglia et al. (2004, pp. 88–89, 98) show that a causal theory in which all constants are Boolean can be translated into a theory of Reiter's (1980b) default logic. The causal calculator (CCALC) (McCain, 1997; E. Giunchiglia et al. 2004) uses satisfiability to solve reasoning problems expressed in the CCALC language, which is similar to $\mathcal{C}+$.

The Fluent Calculus

The fluent calculus was introduced and developed by Hölldobler and Schneeberger (1990), Hölldobler and Thielscher (1995), Bornscheuer and Thielscher (1996), and Thielscher (1995, 1996, 1997, 1999b). A tutorial on the fluent calculus is provided by Thielscher (1999c). Thielscher (1999b, 2000) treats nondeterministic effects in the fluent calculus. The state update axiom for flipping a coin is based on Thielscher (1999b, p. 290). Thielscher (1999c, 2000) discusses concurrent events in the fluent calculus. The state update axioms for opening a door are from Thielscher (1999c). The door latch scenario is due to Allen (1991). Thielscher (1995, 1996, 1999b, 1999c, 2000) discusses indirect effects in the fluent calculus. Thielscher (1999a, 1999c, 2000) treats continuous change in the fluent calculus.

Inferential Frame Problem

Thielscher (1999b) discusses how fluent calculus state update axioms solve the inferential frame problem of the situation calculus. The problem is to reduce the computational cost of determining the truth values of fluents in the successor situation $Do(a, s)$ that results from performing action a in situation s. In the situation calculus, we must apply the successor state axiom for each fluent of interest. In the fluent calculus, we may reason about the truth values of fluents in the successor situation simply by applying a single-state update axiom for the action a. This reasoning is possible even in the presence of partial information about the truth values of fluents. Thielscher (2005) presents the FLUX program for performing commonsense reasoning using the fluent calculus.

Cyc

A system that uses logic for commonsense reasoning is Cyc (Lenat and Guha, 1990; Lenat, 1995). Under development since 1984, Cyc is a large commonsense knowledge base, inference engine, and natural language processing system.

Knowledge is represented in Cyc using CycL, whose syntax is based on the syntax of first-order logic and Lisp (Cycorp, 2002c, chap. 2). The inference engine (Cycorp, 2002a, 2002b, 2002d) uses the resolution principle (Robinson, 1965) and heuristic search (Pearl, 1984). It uses negation as failure to implement the unique names assumption (Reiter, 1980a; McCarthy, 1986) and minimization of abnormalities for default reasoning (McCarthy, 1984a). Cyc also has specialized inference procedures to deal with reflexivity, symmetry, transitivity, dates, lists, numbers, sets, strings, and inheritance (Cycorp, 2002c, chap. 11).

Cyc has two basic ways of representing time-varying properties. In the first representation, an element e of the collection `SomethingExisting` is divided into elements of `SomethingExisting` that represent e at different slices of time (Cycorp, 2002g). These objects are called *temporal subabstractions* of e (Lenat and Guha, 1990, pp. 211–213). For example, the following formula represents that `Nathan2007` is a temporal subabstraction of the person `Nathan`:

```
(subAbstrac Nathan Nathan2007)
```

Temporal subabstractions can be divided into further temporal subabstractions:

```
(subAbstrac Nathan2007 Nathan20070304)
(subAbstrac Nathan2007 Nathan20070305)
(subAbstrac Nathan2007 Nathan20070306)
                ⋮
```

Assertions are made about temporal subabstractions. The following formula states that Nathan was a scientist on March 4, 2007:

```
(occupationOf Nathan20070304 scientist)
```

The second representation uses the `TemporalThing` collection, which has `TimeInterval`, `TimePoint`, and `SomethingExisting` as specializations (Cycorp, 2002g). A formula of the form `(holdsIn` t f`)` represents that formula f is true at all moments of temporal thing t. For example, the previous assertion can be represented as follows:

```
(holdsIn
 (DayFn 4 (MonthFn March (YearFn 2007)))
 (occupationOf Nathan scientist))
```

Assertions involving temporal subabstractions and those involving `holdsIn` are related by several axioms (Guha and Lenat, 1990b).

Cyc represents event occurrences using instances of the `Event` collection, which is a specialization of the `Situation-Temporal` collection (Cycorp, 2002c, sec. 8.1).

Attributes of event instances are represented using various predicates. The time of occurrence of an event may be represented in a variety of ways:

(`dateOfEvent` *e d*)
(`startingDate` *e d*)
(`endingDate` *e d*)
(`contiguousAfter` e_1 e_2)
(`endsAfterEndingOf` e_1 e_2)
(`overlapsStart` e_1 e_2)
(`startsAfterStartingOf` e_1 e_2)
(`temporallyFinishedBy` e_1 e_2)
(`temporallyIntersects` e_1 e_2)
⋮

where e, e_1, and e_2 are event instances, and d is a date. The expression (`subEvents` e_1 e_2) represents that event e_2 is a part of event e_1. The expression (`eventOccursAt` e l) represents that l is the location of event e. The `Action` collection is a specialization of the `Event` collection. The expression (`performedBy` e a) represents that a is the agent of action e.

The representation of event occurrences in Cyc is based on the proposal of Davidson (1967/2001) to represent event occurrences as objects and to represent the meanings of action sentences using predicates that take those objects as arguments. Under this proposal, the sentence "I flew my spaceship to the Morning Star" (p. 114) is represented as (p. 119)

$$(\exists x)(\text{Flew}(\text{I, my spaceship, } x) \& \text{ To}(\text{the Morning Star, } x))$$

Cyc does not have a standard representation for the preconditions and effects of events but, rather, uses a variety of predicates (Parmar, 2001). Cyc cannot simulate the occurrence of events (Parmar, 2001), although previous versions of Cyc had the following rudimentary mechanism for temporal projection (Guha and Lenat, 1990a, pp. 46–47; Guha and Lenat, 1990b). Associated with each formula is an *interval of persistence* that specifies how long the formula is expected to remain true. A frame axiom states that if (1) a formula is proved true at timepoint t_0 without using this frame axiom, (2) the formula's interval of persistence is i, (3) $t_0 < t_1 < t_0 + i$, and (4) the formula is not abnormal with respect to this frame axiom at timepoint t_1, then the formula is true at t_1. Guha (1990) describes a way of handling the Yale shooting scenario in Cyc.

OpenCyc (Cycorp, 2002f) includes an implementation (Cycorp, 2002e) of a hierarchical task network planner (Ghallab et al. 2004, chap. 11). The planner supports preconditions, context-sensitive effects, and compound events. It does not support concurrent events with cumulative and canceling effects, continuous change, indirect effects, nondeterministic effects, release from the commonsense law of inertia, or triggered events.

Other Formalisms

Other logical formalisms for commonsense reasoning include McDermott's (1982) temporal logic, motivated action theory (Stein and Morgenstern, 1994), and $\mathcal{LAP}_{\rightsquigarrow}$ (Castilho, Gasquet, and Herzig, 1999), which is based on dynamic logic (Pratt, 1976; Harel, 1984).

Exercises

15.1 (Research Problem) Prove the equivalence of DEC without *ReleasedAt*, *Releases*, *Trajectory*, *AntiTrajectory*, and concurrent events and the situation calculus restricted to linear time.

15.2 (Research Problem) Prove the equivalence of DEC without *ReleasedAt*, *Releases*, *Trajectory*, *AntiTrajectory*, and concurrent events extended to branching time and the situation calculus.

15.3 (Research Problem) In DEC, disallow (1) times less than zero, (2) triggered events, (3) *Trajectory*, and (4) *Anti-Trajectory*. In TAL, disallow (1) actions that are not single-step actions, (2) duration constraints, (3) durational fluents, and (4) functional fluents. Prove that the resulting versions of DEC and TAL are equivalent.

15.4 (Research Problem) Write a program that translates event calculus domain descriptions into the input language of CCALC (McCain, 1997; E. Giunchiglia et al., 2004).

15.5 (Research Problem) Write a program that translates event calculus domain descriptions into the input language of VITAL (Kvarnström, 2001).

CHAPTER 16

Nonlogical Methods for Commonsense Reasoning

This book uses the event calculus for commonsense reasoning, which is based on first-order logic. Why logic? What about other approaches to commonsense reasoning? In this chapter, we review several other important approaches to commonsense reasoning: qualitative reasoning, analogical processing, probabilistic reasoning, and the society of mind. We assess their strengths and weaknesses and discuss their relationships to the event calculus.

16.1 Qualitative Reasoning

Qualitative reasoning is concerned with explaining and predicting the behavior of physical systems. The task is to take as input a description of a physical system and an initial state and produce as output the possible behaviors of the system given the initial state. Some examples of physical systems modeled using this approach include a water pressure regulator, two tanks connected by a pipe, heat flow between two objects, boiling water in a pot, and a block connected to a spring. Several schemes for qualitative reasoning have been developed. In this section we describe QSIM, which is representative.

16.1.1 QSIM

In QSIM, a physical system is described by a qualitative differential equation (QDE), which corresponds to an infinite set of ordinary differential equations. A QDE consists of a set of variables and a set of constraints on those variables.

Possible constraints are $x+y=z$, $xy=z$, $x=-y$, $y=\frac{dx}{dt}$, $M^+(x,y)$, and $M^-(x,y)$, where x, y, and z are variables. The constraint $M^+(x,y)$ means that x is monotonically increasing with y, and $M^-(x,y)$ means that x is monotonically decreasing with y. Each variable is associated with landmark values $l_1 < \cdots < l_n$. A *qualitative value* of a variable consists of a landmark value l_i or interval (l_i, l_{i+1}) and a direction, which is increasing, steady, or decreasing. A state assigns a qualitative value to each variable.

Given a QDE and an initial state, the QSIM algorithm produces a *behavior tree*, or tree of states. The QSIM algorithm operates by repeatedly taking a state and generating successor states that satisfy the constraints. QSIM also allows a physical system to be described by multiple QDEs and transitions between the QDEs. For example, a bouncing ball can be modeled using one QDE for motion through the air and another QDE for bouncing.

Qualitative reasoning can be used for commonsense reasoning about physical systems. But general commonsense reasoning requires reasoning about both continuous and discrete change. Qualitative reasoning is largely concerned with continuous change and is not concerned with discrete events and associated commonsense phenomena such as context-sensitive, indirect, and nondeterministic effects of events. The commonsense law of inertia is not represented (or even needed) in qualitative reasoning because the behaviors of the system are determined by the QDE.

It has been shown how qualitative descriptions of systems can be expressed in first-order logic and how system behaviors can be predicted using deduction. Some work has been performed on incorporating QSIM into the event calculus; see Exercises 16.2 and 16.3.

16.2 Analogical Processing

A person who encounters a novel situation may not already have the commonsense knowledge necessary to reason about that situation. In such a case, the person might be able to reason about the novel situation by analogy to a familiar situation. For example, consider the case of sand being poured into a kitchen sink with the stopper in place. By analogy to the case of water, we can predict that the sand level will increase and that the sand will eventually spill onto the floor.

Some researchers have proposed that analogical processing can be used for commonsense reasoning and, moreover, that commonsense reasoning often involves analogical processing. In this section, we examine a particularly well-developed mechanism for analogical processing, the structure-mapping engine.

16.2.1 Structure-Mapping Engine

The structure-mapping engine (SME) takes as input a base domain and a target domain and produces as output zero or more global mappings from the base

domain to the target domain. A domain consists of a set of labeled ordered trees, or rooted trees such that (1) each node has an associated label and (2) the children of each node are ordered. Leaf nodes represent objects such as *Beaker* and *Coffee*. Nonleaf nodes represent functions such as *Pressure* and *Temperature*, attributes such as *Liquid* and *Red*, and relations such as *Causes*, *Implies*, and *GreaterThan*. The children of a nonleaf node represent the arguments of the function, attribute, or relation. A function node has one or more children, an attribute node has one child, and a relation node has two or more children.

Global Mappings

A global mapping consists of (1) a one-to-one mapping M from nodes of the base domain to nodes of the target domain satisfying certain constraints, (2) candidate inferences in the target domain, and (3) a score. The mapping M satisfies the following constraints:

- M maps object nodes to object nodes, function nodes to function nodes, attribute nodes to attribute nodes, and relation nodes to relation nodes.
- If M maps a base function or attribute node b to a target node, then b is not a root node. That is, function and attribute nodes are not mapped unless they are part of a larger structure.
- If M maps a base attribute or relation node b to a target node t, then b and t have the same label. Note that this constraint does not apply to object and function nodes.
- If M maps a function, attribute, or relation node b to a target node t, then M maps each child of b to a child of t.

Generating Candidate Inferences

In order to generate candidate inferences in the target domain, we start by forming the set C_b of all trees B of the base domain such that M maps any nonleaf node of B to a target node. That is, we form the set C_b of base domain trees having one or more mapped function, attribute, or relation nodes. These are the base domain trees involved in the analogy. Then, the target domain candidate inferences C_t consist of the base domain trees C_b with their object and function nodes relabeled using the terminology of the target domain. Specifically, for each object and function node b of C_b, we do the following: If M maps b to a target node t, then we relabel b with the label of t; otherwise, if M does not map b to any target node, then we relabel b with a new unique label.

The *score* is an estimate of the quality of the mapping M. A higher score is assigned to the extent that larger trees are mapped, base nodes are mapped to target nodes with the same label, and children of a base node are mapped to children of a target node in the right order.

Example

SME handles over 40 examples, one of which involves an analogy between water flow and heat flow. The base domain, water flow, is represented as follows:

$$Causes(GreaterThan(Pressure(Beaker), Pressure(Vial))$$
$$Flow(Beaker, Vial, Water, Pipe))$$
$$GreaterThan(Diameter(Beaker), Diameter(Vial))$$
$$Liquid(Water)$$
$$FlatTop(Water)$$

The target domain, heat flow, is represented as follows:

$$GreaterThan(Temperature(Coffee), Temperature(IceCube))$$
$$Flow(Coffee, IceCube, Heat, Bar)$$
$$Liquid(Coffee)$$
$$FlatTop(Coffee)$$

Given these base and target domains, SME produces three global mappings. The global mapping with the highest score (=5.99) includes the following mapping M:

$$Beaker \mapsto Coffee$$
$$Vial \mapsto IceCube$$
$$Water \mapsto Heat$$
$$Pipe \mapsto Bar$$
$$Pressure(Beaker) \mapsto Temperature(Coffee)$$
$$Pressure(Vial) \mapsto Temperature(IceCube)$$
$$GreaterThan(Pressure(Beaker), Pressure(Vial)) \mapsto$$
$$GreaterThan(Temperature(Coffee), Temperature(IceCube))$$
$$Flow(Beaker, Vial, Water, Pipe) \mapsto$$
$$Flow(Coffee, IceCube, Heat, Bar)$$

This global mapping includes the following candidate inference:

$$Causes(GreaterThan(Temperature(Coffee), Temperature(IceCube)),$$
$$Flow(Coffee, IceCube, Heat, Bar))$$

Discussion

SME is not intended to be a complete method for commonsense reasoning. Although SME can find potential analogies and generate candidate inferences, it does not address the crucial problem of how to evaluate and possibly repair the inferences. It is assumed that other commonsense knowledge and commonsense reasoning mechanisms will be used to perform these tasks. An interesting line of research would be to integrate SME and the event calculus; see Exercise 16.4.

16.3 Probabilistic Reasoning

Some researchers have proposed the use of probability theory for commonsense reasoning. By using probability measures or other measures of uncertainty, we can quantify not only our uncertainty about a particular scenario but also our uncertainty about our knowledge of how the world works. We can quantify the degree to which we are certain about our formalization of a domain, such as how sure we are that a given action has given preconditions or effects. Using probabilistic reasoning, we can quantify our uncertainty about various sorts of commonsense conclusions derived from information that is given.

16.3.1 Probability and Action

Let us consider how probability theory can be used to reason about a simple action, turning on a faucet. In order to reason about properties over time, we introduce a random variable for each relevant property and timepoint or time interval. We represent the fact that the faucet is running at time 0 as a Boolean random variable R_0 and the fact that the faucet is running at time 1 as a Boolean random variable R_1.

In order to reason about events that occur in time, we introduce a random variable for each relevant event and timepoint or time interval. We represent the event of turning on the faucet at time 0 as a Boolean random variable T_0.

Then we represent our knowledge of this domain. We do not specify a full joint distribution. Instead, we assume that for every time t and random variable v at time t:

- v is directly influenced by zero or more random variables V at time $t - 1$
- the random variable v and the random variables at all times less than or equal to $t - 2$ are conditionally independent given the random variables V

In order to represent the effects of events, we specify the conditional probabilities of properties at time t given the properties and events at time $t - 1$. We consider the influence of R_0 and T_0 on R_1. We specify that, if a faucet that is not running

is turned on, it will be running with high probability:

$$P(R_1|\neg R_0, T_0) = 0.98 \tag{16.1}$$

(Here we deal only with the times 0 and 1. In a formulation with n times, we would write $P(R_t|\neg R_{t-1}, T_{t-1}) = 0.98$ for each $t \in \{1, 2, \ldots, n-1\}$.) If a faucet that is already running is turned on, it will be running with high probability:

$$P(R_1|R_0, T_0) = 0.99$$

Furthermore, we represent what happens when events do not occur. That is, we represent instances of the commonsense law of inertia. A faucet that is not running and is not turned on will not suddenly start running:

$$P(R_1|\neg R_0, \neg T_0) = 0.01 \tag{16.2}$$

A faucet that is running and is not turned on will continue to run:

$$P(R_1|R_0, \neg T_0) = 0.95$$

We can then perform temporal projection, abduction, and postdiction. For temporal projection, we may ask whether the faucet is running at time 1 given that it is not running at time 0 and is turned on at time 0. We see immediately from (16.1) that this is very likely.

For abduction, we may ask whether the faucet is turned on at time 0 given that it is not running at time 0 and is running at time 1. We must determine

$$P(T_0|\neg R_0, R_1)$$

From the definition of conditional probability, we have

$$P(T_0|\neg R_0, R_1) = \frac{P(R_1, \neg R_0, T_0)}{P(\neg R_0, R_1)} \tag{16.3}$$

Using the chain rule, we have

$$P(R_1, \neg R_0, T_0) = P(R_1|\neg R_0, T_0)P(\neg R_0|T_0)P(T_0)$$

From this and (16.1), we have

$$P(R_1, \neg R_0, T_0) = 0.98 \cdot P(\neg R_0|T_0)P(T_0) \tag{16.4}$$

At this point we cannot proceed without further information. We first make an assumption regarding the independence of R_0 and T_0:

$$P(R_0|T_0) = P(R_0) \tag{16.5}$$

(In a more detailed model, we might not wish to make such an assumption. For example, an agent may be unlikely to turn on a faucet that is already running.)

We must also specify prior probabilities for the faucet running at time 0 and the faucet being turned on at time 0:

$$P(R_0) = 0.5 \tag{16.6}$$
$$P(T_0) = 0.1 \tag{16.7}$$

From (16.4), (16.5), (16.6), and (16.7), we have

$$P(R_1, \neg R_0, T_0) = 0.98 \cdot 0.5 \cdot 0.1 = 0.049 \tag{16.8}$$

From the product rule, we have

$$P(\neg R_0, R_1) = P(R_1 | \neg R_0) P(\neg R_0) \tag{16.9}$$

By expanding the possible cases for T_0, we have

$$P(R_1 | \neg R_0) = P(R_1 | \neg R_0, \neg T_0) P(\neg T_0 | \neg R_0) + P(R_1 | \neg R_0, T_0) P(T_0 | \neg R_0)$$

From this, (16.2), (16.5), (16.7), and (16.1), we have

$$P(R_1 | \neg R_0) = 0.01 \cdot 0.9 + 0.98 \cdot 0.1 = 0.107$$

From this, (16.9), and (16.6), we have

$$P(\neg R_0, R_1) = 0.107 \cdot 0.5 = 0.0535$$

From this, (16.3), and (16.8), we have

$$P(T_0 | \neg R_0, R_1) = \frac{0.049}{0.0535} = 0.9159$$

Thus, it is likely that the faucet is turned on at time 0.

For postdiction, we may calculate the probability that the faucet is not running at time 0 given that it is turned on at time 0 and is running at time 1:

$$P(\neg R_0 | T_0, R_1) = 0.4975$$

This calculation is performed in a fashion similar to the calculation for abduction.

This is a simple example involving just three random variables. It is apparent that using these techniques to perform commonsense reasoning on larger examples will be cumbersome.

16.3.2 Bayesian Networks

A more convenient way of performing these calculations is to use Bayesian networks. A *Bayesian* network is a directed acyclic graph in which each node represents a random variable, and a directed link from node n_1 to node n_2 represents that n_1 directly influences n_2. Each node n is associated with a conditional probability table that specifies the conditional probability for every value of n given every combination of values of every node that directly influences n or the prior probability of n if no nodes directly influence n.

Still, even using Bayesian networks, we must introduce a node for every property and event at every timepoint or time interval and specify the conditional probability table for every node. There are many probabilities to specify. In our faucet example, we had to specify probabilities for each instance of the commonsense law of inertia, because probability theory does not supply us with any general mechanism for representing this law.

16.4 Society of Mind

Marvin Minsky developed the *society of mind* theory of human cognition, which proposes that the flexibility of human intelligence arises from the use of many different techniques, not simply one technique such as analogy, connectionism, logic, or probability. This theory starts with the following notions.

The mind consists of a vast collection of interacting simple processes, called agents, critics, or resources.

The agents use many different types of representations and many different methods of reasoning in order to exploit the advantages of each representation and reasoning method and to avoid their disadvantages.

Representations in different agents are connected to one another. These connections facilitate the simultaneous use of representations and switching between representations whenever difficulties are encountered with one of the representations.

The agents are concerned with various realms; examples include the aesthetic, physical, possessional, psychological, spatial, temporal, and topological realms. One or more agents may be concerned with any given realm.

Some agents are concerned with *reflection*, or monitoring and influencing other agents. For example, reflective agents can notice other agents getting into loops and help them break out of those loops.

Minsky proposes various constructs that can be used to build up a society of mind.

A *frame* consists of slots and default values for the slots used to capture expectations about a frequently encountered class of situations. When a frame is applied to a situation, the default values may be used as is or they may be modified to match the situation.

A *frame-array* is a collection of frames that share one or more slots. The frames represent a class of situations from different perspectives, or they represent related classes of situations. The shared slots enable switching from one frame to another to facilitate shifting perspectives or situations.

A *transframe* is a type of frame that has slots for an origin, action, and destination and is used to represent the effects of actions. Transframes may be chained together by matching the destination of each transframe with the origin of the next transframe to predict the results of a sequence of actions.

A *K-line* is a memory structure that records what agents were used to solve a problem. The K-line is used to activate those agents when solving similar problems in the future.

A *paranome* is a mapping among alternative representations of a situation used by two or more agents. When one of the agents updates its representation, the paranome allows the other agents to update their representations appropriately.

Other constructs proposed by Minsky include censors, level-bands, micronemes, picture-frames, polynemes, pronomes, proto-specialists, and suppressors.

There are two ways of viewing the relationship between the society of mind and the event calculus:

- The event calculus could be viewed as one effective method for performing commonsense reasoning within a larger society of mind.
- Constructs from the society of mind could themselves be integrated into the event calculus; see Exercise 16.6.

In this section, we review three commonsense reasoning systems inspired by the society of mind: ThoughtTreasure, Polyscheme, and EM-ONE. All three systems involve collections of interacting agents. ThoughtTreasure and Polyscheme use multiple representation and reasoning methods. Polyscheme and EM-ONE use reflection.

16.4.1 ThoughtTreasure

The ThoughtTreasure commonsense reasoning and story understanding architecture/system uses multiple representation and reasoning mechanisms, including finite automata, grids, path planning, logical formulas, theorem proving, and scripts. Reasoning is performed by a collection of agents that communicate over a shared data structure.

Commonsense Domains

ThoughtTreasure represents and reasons about objects, time-varying properties, events, time, space, and goals. Time is represented using timepoints and time intervals. If t_1 and t_2 are timepoints, e is an event, and p is a property, then

$@t_1 : t_2 \mid e$

represents that e occurs from t_1 to t_2, and

$@t_1 : t_2 \mid p$

represents that p is true from t_1 to t_2. Space is represented by a network of *grids* (occupancy arrays). A physical object occupies one or more cells of a grid over a given time interval.

Reasoning Types

ThoughtTreasure performs two types of reasoning: simulation and understanding. In simulation, the system takes as input a state of the world in which a character has an active goal and produces as output the events and time-varying properties that follow from that state of the world. The system handles several benchmark problems involving simulation: moving a penny from one jar to another, traveling from Paris to New York, and watching television. In the television scenario, the system is told that a character Jim is located in a particular apartment grid containing various objects such as appliances and furniture. The system then produces a sequence of events in which Jim walks from his initial location in the grid to the television and turns it on.

In understanding, the system takes natural language text as input and parses the text into properties and events. Given these observations, the system produces as output zero or more models that tally with those observations. Each model consists of the input properties and events augmented with properties and events filled in by the system. Several benchmark problems involving understanding are handled: going to sleep, stepping outside a grocery store, and taking a shower. In the shower scenario, the system is told that Jim woke up and poured shampoo on his hair. The system produces a model in which Jim gets out of bed, walks to the shower, turns it on, picks up the shampoo, and pours it on his hair.

Discussion

ThoughtTreasure has no general mechanism to deal with the commonsense law of inertia. The burden falls on each agent to update representations appropriately. For example, if Jim is known to be at one location starting at time 0:

```
@0:na|[at-grid Jim AptGrid <gridsubspace 20 78>]
```

and the `grid-walk` agent simulates Jim walking to another location from time 2 to time 3:

```
@2:3|[grid-walk AptGrid 20 78 16 82]
```

then that agent updates the representations to reflect that Jim was at the first location from time 0 to time 3 and at the second location starting at time 3:

```
@0:3|[at-grid Jim AptGrid <gridsubspace 20 78>]
@3:inf|[at-grid Jim AptGrid <gridsubspace 16 82>]
```

Thus, knowledge of the effects of events is represented procedurally within agents. Similarly, knowledge about context-sensitive effects and preconditions of actions is represented procedurally within agents. For example, the agent that simulates a television represents that the television will not go on unless it is plugged in.

ThoughtTreasure does not have a general mechanism to deal with the indirect effects of events. Just as for the commonsense law of inertia, individual agents are responsible for performing the appropriate updates. For example, when the `grid-walk` agent simulates a character walking from one grid location to another, the agent moves not only the character but also all objects held by the character and all objects inside those objects.

ThoughtTreasure does not address continuous change, nondeterministic effects, or release from the commonsense law of inertia. Although ThoughtTreasure is able to represent concurrent events, it does not address concurrent events with cumulative and canceling effects.

ThoughtTreasure is effective at simulation, but less effective at understanding. Both simulation and understanding are performed by agents. Although it is relatively easy to write agents for simulation, it is more difficult to write agents for understanding. Agents must be written to coordinate with one another in order to agree on an interpretation of the input.

ThoughtTreasure does not have a general mechanism for dealing with the problem of agent coordination. This problem is taken up in Polyscheme.

Agent coordination is not an issue in the event calculus because it does not use agents. Instead of representing knowledge procedurally within agents, the approach in the event calculus is to represent knowledge declaratively as logical formulas and to use off-the-shelf procedures for reasoning. The declarative approach allows us to use the best available search techniques, as integrated into off-the-shelf solvers. A procedural knowledge representation requires the procedures to deal with knowledge representation as well as search—a tall order.

16.4.2 Polyscheme

The Polyscheme cognitive architecture/system uses multiple representation and reasoning techniques, including scripts, transframes, hash tables, constraint graph propagation, rules, and single-layer neural networks. Reasoning is performed by a collection of communicating specialists: perception, attribute, physics, causal, temporal, identity hypothesis, identity, place, path, and reflective specialists. Each specialist uses its own techniques for representing and reasoning about the effects

of events, context-sensitive effects of events, event preconditions, triggered events, and other commonsense phenomena.

Commonsense Domains

Polyscheme represents and reasons about objects, object identity, time-varying properties, events, time, and space. Although specialists may use their own representations internally, specialists communicate using logical propositions. A proposition consists of a predicate, arguments, time interval, and world. Worlds, which are similar to the situations of the situation calculus, are used for hypothetical reasoning. Space is represented as a three-dimensional grid. Thus, for example, the proposition

```
at(x, p1-1-1, t1, R)
```

says that object `x` is at grid location `p1-1-1` over time interval `t1` in world `R`.

Reasoning Types

The Polyscheme system performs both temporal projection and explanation. It handles the following temporal projection benchmark problems involving gravity:

A block starts to fall in back of a screen. Infer that the block must have fallen and hit the floor.
If you are then told that another block was in back of the screen, infer that the first block must have fallen and hit the second block.

Polyscheme handles the following explanation benchmark problems involving object identity:

A dry mouse walks behind a screen where there is known to be a puddle of water, and then a dry mouse emerges. Infer that the first and second mouse are not the same mouse.
A red ball rolls behind a screen, and then a red ball rolls out. Infer that the first and second ball are the same ball.
A ball rolls behind one screen, and then a ball rolls out of another screen that is separated from the first screen. Infer that the first and second ball are not the same ball.
A wood ball and a salt ball roll behind a screen. Then a ball rolls out, moves into a puddle of water, and does not melt. Infer that the ball that rolled out was the wooden ball.

(We show how the event calculus can be used to solve similar problems of object identity in Section 10.3.)

Focus Mechanism

The coordination of the activities of specialists is accomplished in Polyscheme via a focus mechanism. At any moment, all specialists focus on a single proposition. A *focus* consists of a proposition and a focus mode, which is PERCEIVE, ASSERT, DENY, WONDER, FIND, ELABORATE, or IMAGINE. An *attraction* consists of a focus and charge. Polyscheme conducts a search for a solution to a commonsense reasoning problem using the following algorithm:

1. The specialists issue attractions.
2. The current focus is set to be the focus of the most highly charged attraction.
3. The specialists concentrate on the current focus:

    (a) Each specialist informs the reflective specialist whether it considers the truth value of the proposition to be true, false, or unknown.

    (b) The reflective specialist determines whether the specialists agree or disagree.

    (c) If the specialists disagree, then the reflective specialist issues attractions designed to move the specialists toward agreement. For example, the reflective specialist may issue an attraction to try to perceive the truth value of the proposition in the external environment or to imagine the consequences of the proposition being false.

4. Go to step 1.

By setting the charges appropriately, various algorithms such as depth-first search, breadth-first search, stochastic simulation, and truth maintenance can be implemented or approximated. In contrast to systems that switch representation techniques only when a difficulty is encountered, Polyscheme is able to make use of all representation techniques implemented by specialists at every step of the search.

Discussion

Polyscheme does not have a general way of dealing with the commonsense law of inertia. The burden is placed on the individual specialists to make the appropriate inferences. Polyscheme does not address continuous change, indirect effects, or nondeterministic effects. Polyscheme can represent concurrent events, but does not deal with concurrent events with cumulative and canceling effects.

Polyscheme improves on ThoughtTreasure by providing a focus mechanism to coordinate the activities of the specialists. An important problem in need of further research is how to set the attractions and charges in such a way as to avoid classical problems of search such as combinatorial explosion of alternatives, getting stuck at local maxima, and oscillation.

16.4.3 EM-ONE

The EM-ONE architecture/system for reflective commonsense thinking has its roots in the multilayer cognitive architectures of Marvin Minsky and Aaron Sloman. EM-ONE consists of a set of critics and a library of narratives or cases. Critics observe, diagnose, and repair problems both in the external world and in the internal mental world of the system. *Narratives* are representations of stories in a simulated world that serve as a source of commonsense knowledge for use by the critics.

Critics

A critic consists of a condition pattern, a narrative pattern, and an action. When a critic is invoked, its patterns are evaluated. If the condition pattern matches the current conditions and the narrative pattern matches a narrative, then the critic is engaged and its action is performed. Four types of critics are implemented in EM-ONE: reactive critics, deliberative critics, reflective critics, and metacritics.

Reactive critics propose and execute actions in the world in response to world conditions. For instance, one reactive critic in EM-ONE executes an immediate turn toward a person who has just called for help.

Deliberative critics reason about a situation before taking action. The critics conduct a heuristic search through a space of hypothetical worlds. Some deliberative critics in EM-ONE evaluate hypothetical worlds, and others generate new ones. Examples of critics that evaluate worlds are those that look for unachieved goals or unexpected consequences of actions. Examples of critics that generate new worlds are those that consult narratives in order to infer the effects of an event, infer a plan that a character might select in order to achieve a goal, or infer that a character might execute the next step of a plan.

Reflective critics reason about and modify the mental processes of the system. They consult *traces* of mental activity, which are records of the critics that are invoked or engaged, and the facts that critics assert or retract. One reflective critic in EM-ONE is engaged when an action fails to produce the intended effect and the examination of the trace reveals that a narrative containing a precondition for that action was not consulted. The reflective critic then modifies the critic that failed to consult the narrative so that it will take this narrative into account in the future.

Metacritics are used to manage reactive, deliberative, and reflective critics. They implement Minsky's *critic-selector model,* in which a critic (1) notices a

particular type of problem, and then (2) initiates an appropriate way of thinking about that type of problem.

Metacritics play a key role in the top-level control loop of EM-ONE, which is as follows:

1. Observations of the world are accepted.
2. Metacritics are invoked.
3. Some metacritics become engaged, which may lead to the invocation of reactive, deliberative, and reflective critics.
4. Some invoked critics become engaged, and those critics may invoke other critics, which may become engaged, and so on.
5. Actions are executed in the world.
6. Go to step 1.

Commonsense Domains

EM-ONE represents entities such as physical objects, characters, events, properties, and relations. Its narrative library represents knowledge about the commonsense domains of space, time, and the mental and social worlds. For example, one narrative says that the character named Pink attached a stick to a board, which caused the stick to be attached to the board. Another narrative says that Pink wants the stick to be visible, which implies that Pink believes that Pink wants the stick to be visible. Facts are represented in EM-ONE using a frame-based knowledge representation language.

Reasoning Types

The deliberative critics of EM-ONE perform forms of temporal projection and planning. The reflective critics perform reflective reasoning. EM-ONE handles a benchmark problem that engages all types of critics. The benchmark problem involves two characters named Green and Pink in a simulated world. The world contains a table with only two legs. The problem is for Green to attach a third leg to the table. Green, as modeled by EM-ONE, attempts to attach a third leg to the table, but this fails because there is not enough space between the table and the ground. Green then asks Pink for help. Pink, also modeled by EM-ONE, mistakenly believes that Green wants to disassemble the table and starts removing a leg from the table. Green realizes that Pink has misunderstood its goal and informs Pink of its goal. Pink lifts up the table for Green, and Green successfully attaches the third leg to the table.

Discussion

EM-ONE represents and reasons about event effects and preconditions. It does not have general mechanisms for dealing with the commonsense law of inertia or indirect effects. EM-ONE performs a novel type of reasoning, reflection, which is not addressed in this book.

Bibliographic Notes

Qualitative Reasoning

Qualitative reasoning was introduced by de Kleer (1975). de Kleer and Brown (1984) introduce component models for qualitative reasoning, Forbus (1984) introduces qualitative process (QP) theory, and Kuipers (1986) introduces QSIM. Kuipers (1994) discusses QSIM in detail. A collection of readings on qualitative reasoning is provided by Weld and de Kleer (1990). Sandewall (1989a) shows how to reason about physical systems using differential equations and logic. E. Davis (1990, pp. 312–320) formalizes in first-order logic a component model of a scale, and E. Davis (1992) formalizes in first-order logic a QP theory of heating a can of water.

Analogical Processing

Gentner (1983) introduces the structure-mapping theory of analogy. Falkenhainer, Forbus, and Gentner (1989) describe the structure-mapping engine (SME). Holyoak and Thagard (1989) describe another algorithm, the analogical constraint mapping engine (ACME), that incorporates constraints for semantic similarity and pragmatic centrality, in addition to structural constraints similar to those used by SME. Gentner, Holyoak, and Kokinov (2001) discuss structure-mapping and a number of other proposals for analogical thinking. We give a condensed formalization of SME that does not capture all of the details of the implementation. For example, SME allows different mapping constraints to be defined by the user. The example analogy between water flow and heat flow is from Falkenhainer et al. (1989). Labeled ordered trees are discussed by Aho, Hopcroft, and Ullman (1983).

Case-Based Reasoning

Case-based reasoning (CBR) (Kolodner, 1993) is similar to analogical processing. In CBR, a problem (target) is solved using the following steps: (1) a *case*, or past experience (base), similar to the current problem is retrieved; (2) the case is used to propose a solution; (3) the solution is evaluated; and (4) problems with the solution are repaired. Because CBR relies on a single case or a small number of

cases (Kolodner, 1993, p. 14), CBR may have difficulty with problems whose solution requires the combination of many cases. The evaluation and repair steps are difficult challenges for CBR systems (Leake, 1996, pp. 11–12, 22–27), which is similar to the situation in analogical processing. Evaluation and repair are usually performed through rule-based reasoning, although some have proposed that CBR itself can be used for evaluation and repair (Sycara, 1988; Kolodner, 1993, pp. 7, 461–462; Leake, 1995). It is recognized that CBR is best used in advisory systems where human users perform evaluation and repair (Leake, 1996, pp. 11, 23).

What is missing from both analogical processing and CBR is a way of performing *reasoning from first principles*, or reasoning based on general knowledge, as described in this book. Along these lines, Forbus and Gentner (1997) and Forbus (2001, pp. 36–40) have proposed that human commonsense reasoning can be explained as a tight combination of analogical processing and first-principles qualitative reasoning. They propose that reasoning from first principles is gradually learned through generalization of many situations understood through analogical processing. They suggest that reasoning from first principles extends, rather than replaces, analogical processing.

The combination of analogical processing and qualitative reasoning has been implemented in the PHINEAS program (Falkenhainer, 1988). This program takes observations of a new domain as input and learns about the domain by using SME to find an analogy to a previously known domain, using the analogy to generate hypotheses about the new domain, using qualitative reasoning to generate predictions, comparing those predictions against the observations, and revising its hypotheses if necessary.

Probabilistic Reasoning

Books on probabilistic reasoning and reasoning about uncertainty are by Pearl (1988b) and Halpern (2003). Russell and Norvig (2003) review probability theory, reasoning about uncertainty, probabilistic reasoning, and probabilistic reasoning over time. Hanks and Madigan (2005) review methods for probabilistic temporal reasoning. Our treatment of turning on a faucet is based on the treatments of the Yale shooting scenario of Pearl (1988a; 1988b, pp. 509–516) and Hanks and Madigan (2005).

Bayesian networks are discussed by Pearl (1988b), Halpern (2003), and Russell and Norvig (2003). Kushmerick, Hanks, and Weld (1995) present a framework for probabilistic planning. Baral, Tran, and Tuan (2002) introduce PAL, a probabilistic action language based on action language $\mathcal{A}$. Tran and Baral (2004a) show how to represent the probabilistic causal models of Pearl (2000) in PAL.

Society of Mind

Minsky developed the society of mind approach (Minsky and Papert, 1972, pp. 92–99; Minsky, 1974, 1977, 1980, 1982, 1986, 1991a, 1994, in press), which uses communicating agents and societies of agents, diverse formalisms and

representations, and reflection. Singh (2004) reviews the society of mind approach and work inspired by it. McCarthy et al. (2002); Minsky, Singh, and Sloman (2004); Singh and Minsky (2004); and Singh, Minsky, and Eslick (2004) propose society-of-mind-like architectures for human-level intelligence. ThoughtTreasure is described by Mueller (1998). Polyscheme is described by Cassimatis (2002) and Cassimatis, Trafton, Bugajska, and Schultz (2004). Search is discussed by Pearl (1984), Russell and Norvig (2003), and Hoos and Stützle (2005). EM-ONE is described by Singh (2005). Sloman (2001) discusses the H-Cogaff three-layer architecture, and Minsky (in press) discusses the Model Six six-level architecture. Minsky (in press) discusses the critic-selector model. Other systems based on the society of mind approach include those of Riecken (1994), Singh (1998), and Hearn (2001).

Connectionism and Neural Networks

There are several other approaches related to commonsense reasoning. In the connectionist or neural network approach (Rumelhart, McClelland, and PDP Research Group, 1986; McClelland, Rumelhart, and PDP Research Group, 1986), a problem is represented as the activation levels of nodes in a network of nodes and directional links, and a solution is produced by the propagation of activation in the network. Representations in neural networks may be local or distributed. In a *local representation*, a concept is represented by a single node. In a *distributed representation*, a concept is represented as the pattern of activation across many nodes.

Sun (1994) presents a connectionist architecture for commonsense reasoning. The architecture consists of two connected neural networks: a network that uses local representations for rule application and a second network that uses distributed representations for similarity matching. The architecture is able to handle examples that combine rule-based and similarity-based reasoning. For example, the system is able to reason that geese may quack because ducks quack and geese and ducks are similar (pp. 23, 82–84). The architecture does not deal with events and time (Sun, 1996, p. 198).

Narayanan (1997) presents a model of representation and reasoning inspired by high-level motor control. He introduces a procedural representation called x-schemas, similar to Petri nets (Reisig, 1985), that can be used to represent and reason about motion words such as *walk*, *run*, *stumble*, and *fall*. Narayanan (1997, sec. 4.1) shows how x-schemas can be encoded as connectionist networks. He provides (sec. 3.7.1) an algorithm for temporal projection using x-schemas and suggests (sec. 3.8.2) that postdiction could be performed using the results of Portinale (1994).

Natural Language Representation and Reasoning

Some researchers have proposed to represent knowledge and reason directly in natural language or in representations close to natural language (Iwańska and Shapiro,

2000; Singh, 2002; Singh et al., 2002; Liu and Singh, 2004a; Singh, Barry, and Liu, 2004). Work has been performed on automatically extracting commonsense knowledge from natural language text (Hearst, 1998; Schubert and Tong, 2003; Liu and Singh, 2004b). The extraction of information from text corpora is discussed by Zernik (1991), Armstrong (1993), Manning and Schütze (1999), and Jurafsky and Martin (2000).

Exercises

16.1 Write event calculus formulas to formalize the definition of a table used by the EM-ONE system (Singh, 2005): a table consists of "four sticks attached to a board" p. 36).

16.2 (Research Problem) Continue the work of R. Miller and Shanahan (1996) on incorporating QSIM (Kuipers, 1994) into the event calculus.

16.3 (Research Problem) Jordan (2004) describes a system that tutors students in introductory physics. The system presents a problem to the student, who enters an answer and an explanation. The system uses abductive reasoning to build a proof for the student's answer and explanation. The proof exposes problems in the student's understanding, such as the *impetus misconception,* or the notion that a force must be continually applied to an object in order for it to move. Investigate how the event calculus, extended using the techniques of R. Miller and Shanahan (1996), could be used to diagnose common misconceptions of introductory physics students.

16.4 (Research Problem) Investigate ways of integrating SME (Falkenhainer et al. 1989) and the event calculus. One idea is as follows: Given event calculus axioms describing a base domain and a target domain, SME could be used to find mappings from the base domain axioms to the target domain axioms and suggest additional target domain axioms.

16.5 (Research Problem) Develop techniques for probabilistic reasoning further with respect to various commonsense phenomena and benchmark problems.

16.6 (Research Problem) Investigate how constructs from the society of mind (Minsky, 1986), such as frame-arrays and paranomes, could be integrated into the event calculus.

CHAPTER 17

Conclusion

This book has provided a detailed treatment of commonsense reasoning and a guide to automating commonsense reasoning using first-order logic and the event calculus. We have described general methods that can be applied to a wide range of commonsense reasoning problems. In this chapter, we summarize what has been accomplished and discuss where this work is leading.

17.1 What Was Accomplished?

We started by describing commonsense reasoning, a common activity that people perform almost without realizing it. We then considered several examples of commonsense reasoning and presented important aspects of this activity that need to be treated by any method for automating it or any formalism for describing it. We demonstrated that all of these aspects can be handled using a particular logic, the event calculus. We described the event calculus, showed how it can be used for commonsense reasoning, and presented some sample applications. We reviewed other logical and nonlogical methods for commonsense reasoning.

17.1.1 What Is the Event Calculus?

The event calculus is based on many-sorted first-order logic with equality. It is characterized by a set of axioms and definitions: 17 for the event calculus (EC) and 12 for the discrete calculus (DEC). The event calculus uses reification: It treats time-varying properties and events as objects so that statements can be made about the truth values of properties at timepoints and the occurrences of events at timepoints.

The event calculus uses circumscription, or the minimization of the extensions of predicates, to allow default reasoning. The event calculus uses the default assumptions that (1) the only events that occur are those known to occur and (2) the only effects of events are those that are known. When new information becomes available, the circumscriptions can be recomputed and reasoning again performed, which may lead to new conclusions. Circumscription is defined by a formula of second-order logic, but in most cases of interest circumscriptions compile into formulas of first-order logic.

An event calculus domain description consists of an axiomatization, observations of world properties, and a narrative of known world events. Given a domain description, various types of commonsense reasoning can be performed. Temporal projection consists of predicting the results of events given an initial state. Abduction consists of determining what events might have led from an initial state to a final state. Postdiction consists of determining the initial state given events and a final state. Model finding consists of determining properties and events that agree with a given narrative and set of observations.

17.1.2 How Is the Event Calculus Used?

We have discussed a number of techniques for using the event calculus to perform commonsense reasoning. How do we go about doing this? First, we create an axiomatization that will be used to solve various problems. Second, we create a domain description for any given problem we wish to solve. Third, we solve the problem using manual proof techniques or using automated reasoning programs such as the Discrete Event Calculus Reasoner.

The axiomatization is created as follows. The kinds of problems to be handled are considered and the areas needing formalization are determined; it is useful to work with several sample problems. Sorts, subsort relationships, and variables for each sort are defined. Predicate, function, fluent, and event symbols are defined. The symbols are then used to write the necessary axioms for the areas of interest.

The domain description for a problem is created as follows. The logical constants for the particular problem are defined. These constants are used to specify observations of time-varying properties (fluents) and a narrative of event occurrences. Temporal ordering formulas, which relate the times of properties and events, may be specified. For example, it may be known that one event occurs prior to another event. Unique names axioms for events and fluents are specified.

17.2 Where Is This Leading?

What is the reach of the techniques described in this book? Where is this leading? What we have here is the following. Given an axiomatization of some commonsense domains and incomplete information about any given world scenario

involving those domains, we can run a reasoning program to fill in missing information about that scenario. This missing information can include events, initial states, intermediate states, and final states. In sum, we have a strategy for building computer systems that understand any commonsense domains of interest. The strategy is to axiomatize the domains, represent a scenario, and then run a reasoning program. We can use this strategy to enrich countless applications with commonsense reasoning capabilities.

There are several potential obstacles to the use of commonsense reasoning. The first obstacle is computing resources. Not all commonsense reasoning problems can be solved efficiently on present-day computers. There are several ways of dealing with this. We can magnify the power of current computers by using grid computing, which exploits the unused computing cycles of many machines on the Internet. We can continue to perform research on the efficiency of reasoning algorithms in order to discover and implement more efficient algorithms. We can also use the knowledge gained from this research to state problems in such a way that they can be solved efficiently. For example, experiments have shown that random SAT problems are most quickly solved either when they are underconstrained so that they have many solutions or when they are overconstrained so that they have few solutions.

Another potential obstacle is our ability to construct complete and correct axiomatizations. Although we have presented appropriate axiomatizations and techniques for a number of commonsense areas, the commonsense world is large. Considerable effort and ingenuity are required to develop good axiomatizations. To some degree the task can be eased by using machine-learning techniques such as inductive logic programming, which learns axiomatizations automatically given sets of examples.

Another potential obstacle is that of linking up the event calculus to the world. How do we transform a problem in the world into an event calculus problem? Some answers to this question have been given in Chapter 14. Certain problems such as those involving electronic payment systems use representations that are easily translated into the event calculus. Other problems require more extensive preprocessing. Natural language text or speech must first be processed by syntactic and semantic parsers, and images must first be treated by image processing software.

17.3 Closing Remarks

This book has considered commonsense reasoning and how the event calculus can capture it. This formalization of commonsense reasoning has two main benefits: It brings us a greater understanding of the human capacity for commonsense reasoning, and it allows us to build more flexible and friendly computer systems. The current situation of automated commonsense reasoning can be likened to the situation of the web in the early 1990s. Decades of research on hypertext and computer networking had finally crystallized into two simple and powerful mechanisms for

sharing information online: the hypertext transfer protocol (HTTP) and the hypertext markup language (HTML). In the years following, these mechanisms allowed the online world to take off exponentially. Similarly, decades of research have now crystallized into a simple and powerful mechanism for commonsense reasoning, the event calculus. I hope that the event calculus will lead to an explosion of applications. I invite you to build them.

Bibliographic Notes

Foster and Kesselman (2004) treat grid computing in detail. Mitchell, Selman, and Levesque (1992) study the time taken by the Davis-Putnam-Logemann-Loveland procedure (M. Davis and Putnam, 1960; M. Davis, Logemann, and Loveland, 1962) to solve randomly generated 3-SAT problems, or SAT problems whose clauses contain three literals. They vary the ratio of the number of clauses to the number of variables and find that problems with a low clause-to-variable ratio (around 2) or a high clause-to-variable ratio (around 8) are quickly solved. The problems with a low clause-to-variable ratio are underconstrained—most truth assignments are satisfying. The problems with a high clause-to-variable ratio are overconstrained—few if any truth assignments are satisfying. On the other hand, problems with a clause-to-variable ratio around 4.25, for which approximately half the truth assignments are satisfying, take much longer to solve. Baum (2004, pp. 22–23, 300–302) argues that the world is overconstrained, which allows humans to develop compact and overconstrained representations, which in turn allows reasoning to proceed efficiently. Muggleton and De Raedt (1994) review methods for inductive logic programming.

APPENDIX A

Logical Foundations

This appendix discusses the logical foundations of the event calculus: relations, inductive definitions, first-order logic, many-sorted first-order logic, second-order logic, real numbers, lists, and circumscription.

A.1 Relations

Relations are used in the definitions of the semantics of first-order logic and many-sorted first-order logic, and in the section on state constraints (Section 3.4).

DEFINITION A.1 An ***n-ary relation*** R on a set S is a subset of S^n.

DEFINITION A.2 A ***binary relation*** R on a set S is a subset of $S \times S$.

DEFINITION A.3 A binary relation $R \subseteq S \times S$ is ***reflexive*** if and only if, for every $a \in S$, $\langle a, a \rangle \in R$.

DEFINITION A.4 A binary relation $R \subseteq S \times S$ is ***irreflexive*** if and only if, for every $a \in S$, $\langle a, a \rangle \notin R$.

DEFINITION A.5 A binary relation $R \subseteq S \times S$ is ***symmetric*** if and only if, for every $a, b \in S$, whenever $\langle a, b \rangle \in R$, we have $\langle b, a \rangle \in R$.

DEFINITION A.6 A binary relation $R \subseteq S \times S$ is ***antisymmetric*** if and only if, for every $a, b \in S$, whenever $\langle a, b \rangle \in R$ and $a \neq b$, we have $\langle b, a \rangle \notin R$.

DEFINITION A.7 A binary relation $R \subseteq S \times S$ is ***transitive*** if and only if, for every $a, b, c \in S$, whenever $\langle a, b \rangle \in R$ and $\langle b, c \rangle \in R$, we have $\langle a, c \rangle \in R$.

DEFINITION A.8 A binary relation $R \subseteq S \times S$ is ***intransitive*** if and only if, for every $a, b, c \in S$, whenever $\langle a, b \rangle \in R$ and $\langle b, c \rangle \in R$, we have $\langle a, c \rangle \notin R$.

DEFINITION A.9 A binary relation $R \subseteq S \times S$ is ***trichotomous*** if and only if, for every $a, b \in S$, exactly one of the following is the case: $\langle a, b \rangle \in R$, $a = b$, or $\langle b, a \rangle \in R$.

DEFINITION A.10 A binary relation $R \subseteq S \times S$ is ***total*** if and only if, for every $a \in S$, there is a $b \in S$ such that $\langle a, b \rangle \in R$.

DEFINITION A.11 A binary relation $R \subseteq S \times S$ is ***functional*** if and only if, for every $a, b, c \in S$, whenever $\langle a, b \rangle \in R$ and $\langle a, c \rangle \in R$, we have $b = c$.

DEFINITION A.12 A binary relation $R \subseteq S \times S$ is ***surjective*** if and only if, for every $b \in S$, there is an $a \in S$ such that $\langle a, b \rangle \in R$.

DEFINITION A.13 A binary relation $R \subseteq S \times S$ is ***injective*** if and only if, for every $a, b, c \in S$, whenever $\langle a, c \rangle \in R$ and $\langle b, c \rangle \in R$, we have $a = b$.

DEFINITION A.14 A binary relation R is an ***equivalence relation*** if and only if R is reflexive, symmetric, and transitive.

DEFINITION A.15 A binary relation R is a ***partial order*** if and only if R is reflexive, antisymmetric, and transitive.

DEFINITION A.16 A partial order $R \subseteq S \times S$ is a ***total order*** if and only if, for every $a, b \in S$, either $\langle a, b \rangle \in R$ or $\langle b, a \rangle \in R$.

A.2 Inductive Definitions

An *inductive definition* of a set S consists of

- *basis clauses* specifying objects that are in S
- *inductive clauses* specifying how to create new objects in S given other objects in S

- an *extremal clause* (or *limiting clause*) specifying that the only objects in S are those that can be obtained by applying the basis and inductive clauses a finite number of times

For example, a nonnegative integer is defined inductively as follows:

- *Basis clause*: 0 is a nonnegative integer.
- *Inductive clause*: If n is a nonnegative integer, then $n + 1$ is a nonnegative integer.
- *Extremal clause*: Nothing else is a nonnegative integer.

A.3 First-Order Logic

This section discusses the syntax, semantics, and proof theory of first-order logic.

A.3.1 Syntax of First-Order Logic

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

DEFINITION A.17

A ***term*** is defined inductively as follows:

- A constant is a term.
- A variable is a term.
- If ϕ is an n-ary function symbol and $\tau_1, \ldots, \tau_n$ are terms, then $\phi(\tau_1, \ldots, \tau_n)$ is a term. The terms $\tau_1, \ldots, \tau_n$ are the ***arguments*** of ϕ.
- Nothing else is a term.

DEFINITION A.18

A ***ground term*** is a term containing no variables.

DEFINITION A.19

An ***atom*** is defined inductively as follows:

- If ρ is an n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ are terms, then $\rho(\tau_1, \ldots, \tau_n)$ is an atom. The terms $\tau_1, \ldots, \tau_n$ are the arguments of ρ.
- If τ_1 and τ_2 are terms, then $\tau_1 = \tau_2$ is an atom.
- Nothing else is an atom.

DEFINITION A.20 A ***ground atom*** is an atom containing no variables.

DEFINITION A.21 A ***formula*** is defined inductively as follows:

- An atom is a formula.
- $\bot$ (***false***) and $\top$ (***true***) are formulas.
- If α is a formula, then $\neg\alpha$ is a formula, called a ***negation***.
- If α and β are formulas, then $(\alpha \wedge \beta)$ is a formula, called a ***conjunction***. The formulas α and β are the ***conjuncts*** of $(\alpha \wedge \beta)$.
- If α and β are formulas, then $(\alpha \vee \beta)$ is a formula, called a ***disjunction***. The formulas α and β are the ***disjuncts*** of $(\alpha \vee \beta)$.
- If α and β are formulas, then $(\alpha \Rightarrow \beta)$ is a formula, called a ***conditional*** or ***implication***. The formula α is the ***antecedent*** or ***hypothesis*** of $(\alpha \Rightarrow \beta)$, whereas β is its ***consequent*** or ***conclusion***. $\Rightarrow$ is sometimes written as $\rightarrow$ or $\supset$. The implication $(\alpha \Rightarrow \beta)$ represents that α is a ***sufficient condition*** for β or that β is a ***necessary condition*** for α.
- If α and β are formulas, then $(\alpha \Leftrightarrow \beta)$ is a formula, called a ***biconditional***, ***bi-implication***, or ***equivalence***. $\Leftrightarrow$ is sometimes written as $\leftrightarrow$ or $\equiv$.
- If v is a variable and α is a formula, then $(\exists v\, \alpha)$ is a formula, called an ***existential quantification***. $\exists v$ is an ***existential quantifier***.
- If v is a variable and α is a formula, then $(\forall v\, \alpha)$ is a formula, called a ***universal quantification***. $\forall v$ is a ***universal quantifier***.
- Nothing else is a formula.

The symbols $\bot$, $\top$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ are logical connectives. The symbols $\bot$ and $\top$ are zero-place connectives. The symbol $\exists$ is the existential quantifier symbol, and the symbol $\forall$ is the universal quantifier symbol.

DEFINITION A.22 The ***scope*** of the existential quantifier $\exists v$ in the formula $(\exists v\, \alpha)$ is α.

DEFINITION A.23 The ***scope*** of the universal quantifier $\forall v$ in the formula $(\forall v\, \alpha)$ is α.

DEFINITION A.24 An occurrence of a variable v is ***bound*** in a formula if and only if (1) the occurrence is within the scope of an existential quantifier $\exists v$ or a universal quantifier $\forall v$ in the formula, or (2) the occurrence is the variable v of an existential quantifier $\exists v$ or a universal quantifier $\forall v$ in the formula.

DEFINITION A.25 An occurrence of a variable is ***free*** in a formula if and only if it is not bound in the formula.

DEFINITION A.26 A ***sentence*** is a formula containing no occurrences of variables that are free in the formula.

In order to simplify the reading and writing of first-order logic formulas and sentences, we use the following notational conventions:

- We omit unnecessary parentheses. The precedence of logical connectives and quantifier symbols, from highest to lowest, is: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$.
- We use $\tau_1 \neq \tau_2$ as an abbreviation for $\neg(\tau_1 = \tau_2)$.
- We use $\exists v_1, \ldots, v_n \alpha$ as an abbreviation for $\exists v_1 \ldots \exists v_n \alpha$, and $\forall v_1, \ldots, v_n \alpha$ as an abbreviation for $\forall v_1 \ldots \forall v_n \alpha$.

(See also Section 2.1.5 for further notational conventions.)

A.3.2 Semantics of First-Order Logic

The *semantics* of first-order logic is defined as follows. The meaning of a formula of first-order logic is the set of structures in which the formula is true or the set of models of the formula.

DEFINITION A.27 A ***structure*** (or ***interpretation***) $\mathcal{S}$ for a language $\mathcal{L}$ of first-order logic with equality consists of:

- A nonempty set of objects $dom(\mathcal{S})$ called the ***domain***.
- A function $c^{\mathcal{S}}$ from the constants of $\mathcal{L}$ to $dom(\mathcal{S})$. Given a constant a, we say that $c^{\mathcal{S}}(a)$ is the object ***named*** by a in $\mathcal{S}$.
- For each n-ary predicate symbol ρ of $\mathcal{L}$, an n-ary relation $\rho^{\mathcal{S}} \subseteq dom(\mathcal{S})^n$. The set $\rho^{\mathcal{S}}$ is called the ***extension*** of ρ in $\mathcal{S}$.
- For each n-ary function symbol ϕ of $\mathcal{L}$, an n-ary function $\phi^{\mathcal{S}}$ from $dom(\mathcal{S})^n$ to $dom(\mathcal{S})$.

DEFINITION A.28 A ***variable assignment*** on a structure $\mathcal{S}$ is a function from the variables of $\mathcal{L}$ to $dom(\mathcal{S})$.

DEFINITION A.29 If V is a variable assignment, v and v are variables, and $d \in dom(\mathcal{S})$, then $V[v/d]$ is defined as follows:

$$V[v/d](v) = \begin{cases} d & \text{if } v = v \\ V(v) & \text{otherwise} \end{cases}$$

DEFINITION A.30

A variable assignment V on $\mathcal{S}$ is extended to a function $\overline{V}$ from terms of $\mathcal{L}$ to $dom(\mathcal{S})$ as follows:

- If v is a variable, then $\overline{V}(v) = V(v)$.
- If a is a constant, then $\overline{V}(a) = c^{\mathcal{S}}(a)$.
- If ϕ is an n-ary function symbol and $\tau_1, \ldots, \tau_n$ are terms, then $\overline{V}(\phi(\tau_1, \ldots, \tau_n)) = \phi^{\mathcal{S}}(\overline{V}(\tau_1), \ldots, \overline{V}(\tau_n))$.

DEFINITION A.31

The ***truth*** of a formula ψ in a structure $\mathcal{S}$ under variable assignment V, written $(\mathcal{S}, V) \models \psi$, is defined as follows:

- If ρ is an n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ are terms, then $(\mathcal{S}, V) \models \rho(\tau_1, \ldots, \tau_n)$ if and only if $\langle \overline{V}(\tau_1), \ldots, \overline{V}(\tau_n) \rangle \in \rho^{\mathcal{S}}$.
- If τ_1 and τ_2 are terms, then $(\mathcal{S}, V) \models \tau_1 = \tau_2$ if and only if $\overline{V}(\tau_1) = \overline{V}(\tau_2)$.
- It is not the case that $(\mathcal{S}, V) \models \bot$.
- It is the case that $(\mathcal{S}, V) \models \top$.
- If α is a formula, then $(\mathcal{S}, V) \models \neg\alpha$ if and only if it is not the case that $(\mathcal{S}, V) \models \alpha$.
- If α and β are formulas, then $(\mathcal{S}, V) \models (\alpha \wedge \beta)$ if and only if $(\mathcal{S}, V) \models \alpha$ and $(\mathcal{S}, V) \models \beta$.
- If α and β are formulas, then $(\mathcal{S}, V) \models (\alpha \vee \beta)$ if and only if $(\mathcal{S}, V) \models \alpha$ or $(\mathcal{S}, V) \models \beta$.
- If α and β are formulas, then $(\mathcal{S}, V) \models (\alpha \Rightarrow \beta)$ if and only if $(\mathcal{S}, V) \models \neg\alpha$ or $(\mathcal{S}, V) \models \beta$.
- If α and β are formulas, then $(\mathcal{S}, V) \models (\alpha \Leftrightarrow \beta)$ if and only if $(\mathcal{S}, V) \models (\alpha \Rightarrow \beta)$ and $(\mathcal{S}, V) \models (\beta \Rightarrow \alpha)$.
- If v is a variable and α is a formula, then $(\mathcal{S}, V) \models (\exists v\, \alpha)$ if and only if $(\mathcal{S}, V[v/d]) \models \alpha$ for some $d \in dom(\mathcal{S})$.
- If v is a variable and α is a formula, then $(\mathcal{S}, V) \models (\forall v\, \alpha)$ if and only if $(\mathcal{S}, V[v/d]) \models \alpha$ for every $d \in dom(\mathcal{S})$.

DEFINITION A.32

A formula ψ is ***true*** in $\mathcal{S}$, written $\mathcal{S} \models \psi$, if and only if, for every variable assignment V, we have $(\mathcal{S}, V) \models \psi$. In this case, we say that $\mathcal{S}$ is a ***model*** of ψ, and that $\mathcal{S}$ ***satisfies*** ψ.

DEFINITION A.33

A formula ψ is ***satisfiable*** if and only if, for some structure $\mathcal{S}$ and variable assignment V, we have $(\mathcal{S}, V) \models \psi$.

Table A.1 Useful tautologies[a]

Name	*Tautology*
Associative laws	$((\alpha \wedge \beta) \wedge \gamma) \Leftrightarrow (\alpha \wedge (\beta \wedge \gamma))$
	$((\alpha \vee \beta) \vee \gamma) \Leftrightarrow (\alpha \vee (\beta \vee \gamma))$
Commutative laws	$(\alpha \wedge \beta) \Leftrightarrow (\beta \wedge \alpha)$
	$(\alpha \vee \beta) \Leftrightarrow (\beta \vee \alpha)$
Contraposition	$(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\beta \Rightarrow \neg\alpha)$
De Morgan's laws	$\neg(\alpha \wedge \beta) \Leftrightarrow (\neg\alpha \vee \neg\beta)$
	$\neg(\alpha \vee \beta) \Leftrightarrow (\neg\alpha \wedge \neg\beta)$
Distributive laws	$(\alpha \vee (\beta \wedge \gamma)) \Leftrightarrow ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$
	$(\alpha \wedge (\beta \vee \gamma)) \Leftrightarrow ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$
Double negation	$\neg\neg\alpha \Leftrightarrow \alpha$
Exportation, importation	$((\alpha \wedge \beta) \Rightarrow \gamma) \Leftrightarrow (\alpha \Rightarrow (\beta \Rightarrow \gamma))$
Idempotence	$(\alpha \wedge \alpha) \Leftrightarrow \alpha$
	$(\alpha \vee \alpha) \Leftrightarrow \alpha$
Law of contradiction	$\neg(\alpha \wedge \neg\alpha)$
Law of excluded middle	$\alpha \vee \neg\alpha$
Laws for bi-implication	$(\alpha \Leftrightarrow \beta) \Leftrightarrow ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$
	$(\alpha \Leftrightarrow \beta) \Leftrightarrow ((\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha))$
Law for implication	$(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\alpha \vee \beta)$
Quantifier exchange	$(\neg\exists v\, \alpha) \Leftrightarrow (\forall v\, \neg\alpha)$

[a] α, β, γ = formulas; v = variable.

DEFINITION A.34

A formula ψ is ***valid***, written $\models \psi$, if and only if, for every structure $\mathcal{S}$, we have $\mathcal{S} \models \psi$. A valid formula is called a ***tautology***. Some useful tautologies are shown in Table A.1.

DEFINITION A.35

A formula ψ ***entails*** a formula π, written $\psi \models \pi$, if and only if, for every $\mathcal{S}$ such that $\mathcal{S} \models \psi$, we have $\mathcal{S} \models \pi$.

A.3.3 Proof Theory

Proof theory is concerned with defining the notion of a proof.

DEFINITION A.36 An ***axiom system*** A consists of a set of ***logical axioms*** and a set of ***inference rules***. The logical axioms are formulas, and inference rules specify how a formula may be inferred from other formulas.

A standard inference rule is *modus ponens*, which states that, from α and $\alpha \Rightarrow \beta$, we may infer β.

DEFINITION A.37 A ***proof*** of a formula π given a formula ψ is a finite sequence of formulas such that the last formula is π and each formula is either a logical axiom, ψ, or a formula inferred from previous formulas using one of the inference rules.

DEFINITION A.38 A formula π is ***provable*** from a formula ψ, written $\psi \vdash \pi$, if and only if there is a proof of π given ψ.

DEFINITION A.39 A formula ψ is ***consistent*** if and only if there is no formula π such that both $\psi \vdash \pi$ and $\psi \vdash \neg\pi$.

DEFINITION A.40 A formula is ***inconsistent*** if and only if it is not consistent.

DEFINITION A.41 An axiom system A is ***sound*** if and only if, for all formulas ψ and π, it is the case that $\psi \vdash \pi$ implies $\psi \models \pi$.

DEFINITION A.42 An axiom system A is ***complete*** if and only if, for all formulas ψ and π, it is the case that $\psi \models \pi$ implies $\psi \vdash \pi$.

Many sound and complete axiom systems exist for first-order logic.

A.4 Many-Sorted First-Order Logic

Many-sorted first-order logic is used by the event calculus to distinguish such things as events and fluents.

A.4.1 Syntax of Many-Sorted First-Order Logic

In the event calculus we write formulas such as

$$\exists e, t\ (Happens(e, t) \wedge t_1 \leq t < t_2 \wedge Initiates(e, f, t))$$

But consider the ground atom $Happens(Lucy, Margaret)$. We would like to regard this as ill-formed. We would like the variable e to range only over events and the

variable t to range only over timepoints. Therefore, the event calculus is based on a logic with many *sorts*.

Many-sorted first-order logic with equality is an extension of first-order logic with equality, which is one-sorted. A language $\mathcal{L}$ of many-sorted first-order logic with equality is associated with a nonempty set of sorts S. The sorts are ordered by a subsort partial order $H \subseteq S \times S$. Every function symbol, constant, and variable has an associated sort. Every argument position of every predicate symbol and function symbol has an associated sort.

DEFINITION A.43

A sort s_1 is a ***subsort*** of a sort s_2 if and only if $\langle s_1, s_2 \rangle \in H$. Note that every sort is a subsort of itself.

DEFINITION A.44

The sort of a term τ is defined as follows:

- If τ is a constant or variable, the sort of the term τ is the sort of the constant or variable τ.
- If τ is $\phi(\tau_1, \ldots, \tau_n)$, where ϕ is an n-ary function symbol, $\tau_1, \ldots, \tau_n$ are terms, and for each $i \in \{1, \ldots, n\}$ the sort of τ_i is a subsort of the sort of the ith argument position of ϕ, then the sort of τ is the sort of ϕ.

DEFINITION A.45

A ***term*** is defined inductively as follows:

- A constant is a term.
- A variable is a term.
- If ϕ is an n-ary function symbol, $\tau_1, \ldots, \tau_n$ are terms, and for each $i \in \{1, \ldots, n\}$ the sort of τ_i is a subsort of the sort of the ith argument position of ϕ, then $\phi(\tau_1, \ldots, \tau_n)$ is a term.
- Nothing else is a term.

DEFINITION A.46

An ***atom*** is defined inductively as follows:

- If ρ is an n-ary predicate symbol, $\tau_1, \ldots, \tau_n$ are terms, and for each $i \in \{1, \ldots, n\}$ the sort of τ_i is a subsort of the sort of the ith argument position of ρ, then $\rho(\tau_1, \ldots, \tau_n)$ is an atom.
- If τ_1 and τ_2 are terms, then $\tau_1 = \tau_2$ is an atom.
- Nothing else is an atom.

The definition of a formula is the same as for one-sorted logic.

A.4.2 Semantics of Many-Sorted First-Order Logic

DEFINITION A.47

A ***structure*** $\mathcal{S}$ for a language $\mathcal{L}$ of many-sorted first-order logic with equality consists of:

- For each sort s of $\mathcal{L}$, a nonempty set of objects $dom_s(\mathcal{S})$ called the ***domain*** of s such that for every sort s' of $\mathcal{L}$, if s is a subsort of s' (that is, if $\langle s, s' \rangle \in H$), then $dom_s(\mathcal{S}) \subseteq dom_{s'}(\mathcal{S})$.
- For each sort s of $\mathcal{L}$, a function $c_s^{\mathcal{S}}$ from the set of constants of $\mathcal{L}$ whose sort is s to $dom_s(\mathcal{S})$.
- For each n-ary predicate symbol ρ of $\mathcal{L}$, an n-ary relation $\rho^{\mathcal{S}} \subseteq dom_{s_1}(\mathcal{S}) \times \cdots \times dom_{s_n}(\mathcal{S})$, where $s_1, \ldots, s_n$ are the sorts of the argument positions of ρ.
- For each n-ary function symbol ϕ of $\mathcal{L}$, an n-ary function $\phi^{\mathcal{S}}$ from $dom_{s_1}(\mathcal{S}) \times \cdots \times dom_{s_n}(\mathcal{S})$ to $dom_s(\mathcal{S})$, where $s_1, \ldots, s_n$ are the sorts of the argument positions of ϕ and s is the sort of ϕ.

The definition of truth is similar to that for one-sorted logic. Many-sorted first-order logic is reducible to one-sorted first-order logic. The usual method is to replace formulas of the form

$$\exists v \, \Sigma(v) \tag{A.1}$$

with

$$\exists v \, (Sort(v) = s \Rightarrow \Sigma(v)) \tag{A.2}$$

where s is the sort of the variable v. The formula (A.2) is called the relativization of (A.1). Universal quantification is treated similarly.

A.5 Second-Order Logic

Second-order logic extends first-order logic with quantification over predicates and functions. It is used in this book to define circumscription. Second-order logic introduces predicate variables and function variables. The definition of a term is modified to allow the use of a function variable in place of a function symbol. The definition of an atom is modified to allow the use of a predicate variable in place of a predicate symbol. The definition of a formula is modified to allow predicate variables and function variables to appear in quantifiers.

A.6 Datatypes

Real numbers and lists are used in the event calculus and domain formalizations.

A.6.1 Real Numbers

We have a real number sort R, with numeric constants such as -1, 3, and π. Function and predicate symbols are defined as in Table A.2, which also provides the abbreviations that are used. Table A.3 provides further abbreviations.

We do not provide an axiomatization of real numbers but, instead, assume a standard interpretation of the reals. That is, we assume that in all structures $\mathcal{S}$:

- We have $dom_R(\mathcal{S}) = \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers.

Table A.2 Functions and predicates for real numbers

Type	*Function/predicate*	*Abbreviation*
Function	$Plus(x,y)$	$x+y$
	$Minus(x,y)$	$x-y$
	$Times(x,y)$	xy, $x \cdot y$
	$DividedBy(x,y)$	x/y
	$Remainder(x,y)$	
	$Power(x,y)$	x^y
	$Floor(x)$	$\lfloor x \rfloor$
	$Sin(x)$	
	$Cos(x)$	
Predicate	$LessThan(x,y)$	$x < y$

Table A.3 Abbreviations for real number formulas

Abbreviation	*Formula*
$x \leq y$	$LessThan(x,y) \vee x = y$
$x > y$	$\neg LessThan(x,y) \wedge x \neq y$
$x \geq y$	$\neg LessThan(x,y)$
$\sqrt{x}$	$Power(x, 0.5)$
$x < y < z$	$x < y \wedge y < z$
$x < y \leq z$	$x < y \wedge y \leq z$
$x \leq y < z$	$x \leq y \wedge y < z$
$x \leq y \leq z$	$x \leq y \wedge y \leq z$

- For every numeric constant a that represents the real number x, we have $c_R^{\mathcal{S}}(a) = x$.
- For every n-ary function symbol ϕ in Table A.2, we have $\phi^{\mathcal{S}}(x_1, \ldots, x_n) = \phi'(x_1, \ldots, x_n)$, where ϕ' is the function from $\mathbb{R}^n$ to $\mathbb{R}$ corresponding to ϕ. For example, $Plus^{\mathcal{S}}(x, y) = x + y$. We assume that the functions corresponding to *DividedBy* and *Power* are extended to total functions. For example, we may define $\frac{x}{0} = 0$ for every $x \in \mathbb{R}$.
- For every n-ary predicate symbol ρ in Table A.2, we have $\rho^{\mathcal{S}} = \rho'$, where ρ' is the relation on $\mathbb{R}^n$ corresponding to ρ. For example, $\langle x, y \rangle \in LessThan^{\mathcal{S}}$ if and only if $x < y$.

A.6.2 Lists

We present an axiomatization of lists based on that of Zohar Manna and Richard Waldinger. We use lists to represent plans in Section 11.1. We have a list sort, with variables l, l_1, l_2, ... We have a sort for other entities, with variables x, x_1, x_2, ...

We have the following functions.

$Insert(x, l_1) = l_2$: The result of inserting x at the front of list l_1 is l_2.

$Head(l) = x$: The head of list l is x.

$Tail(l_1) = l_2$: The tail of list l_1 is list l_2.

$Length(l) = n$: The length of list l is n.

We also have a predicate $ElementOf(x, l)$, which represents that x is an element of the list l, and a constant *Empty*, which represents the empty list.

We use $[\tau_1, \tau_2, \ldots, \tau_n]$ as an abbreviation for $Insert(\tau_1, Insert(\tau_2, \ldots, Insert(\tau_n, Empty)\ldots))$, and we use $x \in l$ as an abbreviation for $ElementOf(x, l)$.

Intuitively, we would like the following to be true:

$$Insert(A, [B, C]) = [A, B, C]$$
$$Head([A, B]) = A$$
$$Tail([A, B, C]) = [B, C]$$
$$Length(Empty) = 0$$
$$Length([A, B, C]) = 3$$
$$B \in [A, B, C]$$
$$[A, B, C] = [A, B, C]$$
$$P = Q \Rightarrow [P, B] = [Q, B]$$

We have the following axioms.

AXIOM L1 $Head(Insert(x, l)) = x$

AXIOM L2 $Tail(Insert(x, l)) = l$

AXIOM L3 $Length(Empty) = 0$

AXIOM L4 $Length(Insert(x, l)) = 1 + Length(l)$

AXIOM L5 $\neg ElementOf(x, Empty)$

AXIOM L6 $ElementOf(x_1, Insert(x_2, l)) \Leftrightarrow x_1 = x_2 \vee ElementOf(x_1, l)$

AXIOM L7 $Empty \neq Insert(x, l)$

AXIOM L8 $Insert(x_1, l_1) = Insert(x_2, l_2) \Leftrightarrow x_1 = x_2 \wedge l_1 = l_2$

A.7 Circumscription

Circumscription is used in the event calculus to implement default reasoning.

A.7.1 Definition of Circumscription

We start by defining some abbreviations, and then we define circumscription.

DEFINITION A.48 If ρ and σ are n-ary predicate symbols and $v_1, \ldots, v_n$ are distinct variables of appropriate sort, then:

- $\rho = \sigma$ is an abbreviation for $\forall v_1, \ldots, v_n\, \rho(v_1, \ldots, v_n) \Leftrightarrow \sigma(v_1, \ldots, v_n)$.
- $\rho \leq \sigma$ is an abbreviation for $\forall v_1, \ldots, v_n\, \rho(v_1, \ldots, v_n) \Rightarrow \sigma(v_1, \ldots, v_n)$.

DEFINITION A.49 If ρ and σ are n-ary predicate symbols, then $\rho < \sigma$ is an abbreviation for $(\rho \leq \sigma) \wedge \neg(\rho = \sigma)$.

DEFINITION A.50 If Γ is a formula containing the predicate symbol ρ, then the ***(basic) circumscription*** of ρ in Γ, written $CIRC[\Gamma; \rho]$, is the formula of second-order logic

$$\Gamma \wedge \neg\exists \phi\, (\Gamma(\phi) \wedge \phi < \rho)$$

where ϕ is a predicate variable with the same arity and argument sorts as ρ, and $\Gamma(\phi)$ is the formula obtained from Γ by replacing each occurrence of ρ with ϕ.

This says that in every model of $CIRC[\Gamma; \rho]$, the extension of ρ complies with Γ and there is no proper subset of the extension of ρ that complies with Γ. That is, the extension of ρ is minimal given Γ.

A.7.2 Example: Circumscription of *P(A)*

Suppose $\Gamma = P(A)$. Then $CIRC[\Gamma; P]$ is given by the second-order formula

$$P(A) \wedge \neg\exists\phi\,(\phi(A) \wedge \phi < P)$$

This means that for every model $\mathcal{M}$ of $CIRC[\Gamma; P]$, if $\mathbf{A}$ is the object named by A in $\mathcal{M}$, then $\mathbf{A} \in P^{\mathcal{M}}$ and there is no proper subset Q of $P^{\mathcal{M}}$ such that $\mathbf{A} \in$ Q, from which we conclude that $P^{\mathcal{M}} = \{\mathbf{A}\}$. Therefore, $CIRC[\Gamma; P]$ is equivalent to the first-order formula

$$\forall x\,(P(x) \Leftrightarrow x = A)$$

A.7.3 Parallel Circumscription

The event calculus requires circumscription of several predicates, called parallel circumscription.

DEFINITION A.51

If Γ is a formula containing the predicate symbols $\rho_1, \ldots, \rho_n$, then the ***(parallel) circumscription*** of $\rho_1, \ldots, \rho_n$ in Γ, written $CIRC[\Gamma; \rho_1, \ldots, \rho_n]$, is the formula of second-order logic

$$\Gamma \wedge \neg\exists\phi_1, \ldots, \phi_n\,(\Gamma(\phi_1, \ldots, \phi_n) \wedge \bigwedge_{i=1}^{n} \phi_i < \rho_i)$$

where $\phi_1, \ldots, \phi_n$ are distinct predicate variables with the same arities and argument sorts as $\rho_1, \ldots, \rho_n$, respectively, and $\Gamma(\phi_1, \ldots, \phi_n)$ is the formula obtained from Γ by replacing each occurrence of $\rho_1, \ldots, \rho_n$ with $\phi_1, \ldots, \phi_n$, respectively.

Bibliographic Notes

Inductive definitions are discussed by Kleene (1952, pp. 258–261). First-order logic is treated by Church (1956), Curry (1963), Kleene (1967), Mates (1972), Hamilton (1988), Ebbinghaus, Flum, and Thomas (1994), Mendelson (1997), Nerode and Shore (1997), Enderton (2001), Hodges (2001), and Wolf (2005). Model theory is treated in detail by Hodges (1993). The notation $(\mathcal{S}, V) \models \psi$ is taken from Fagin et al. (1995, p. 79). Lists of tautologies are provided by Whitehead and Russell (1925), Kleene (1952, pp. 118–119, 162–163), Mates (1972, pp. 103–110, 126–130), Doherty et al. (1997, p. 301), Nerode and Shore (1997, pp. 35–36), Enderton (2001, pp. 26–27), and Wolfram (2002, p. 817). Proof theory is treated in detail by Buss (1998). Our definition of an axiom system is based on that of Fagin et al. (1995, p. 50). Proofs of soundness and completeness of axiom systems for first-order logic are given by Hamilton (1988, pp. 73–104), Ebbinghaus et al. (1994, pp. 59–85) and Enderton (2001, pp. 109–146). These proofs of completeness are patterned after the one given by Henkin (1949), which in turn derives from the original proof of Gödel (1930/1967).

Our definition of many-sorted first-order logic is based on that of Enderton (2001, pp. 295–299), extended with notions from Walther (1983, 1987) and Oberschelp (1990, pp. 12–13), who add a subsort order to many-sorted first-order logic. Many-sorted logic with a subsort order is sometimes called "order-sorted logic" (Oberschelp, 1990, p. 8). Enderton (2001, pp. 296–299) reduces many-sorted first-order logic to one-sorted first-order logic. Relativization is discussed by Walther (1983, p. 883; 1987, p. 6), Oberschelp (1990, p. 10), Ebbinghaus et al. (1994, p. 49), and Enderton (2001, p. 297).

In our version of many-sorted first-order logic, $\tau_1 = \tau_2$ is an atom regardless of the sorts of τ_1 and τ_2, as in Walther (1983, p. 885). In contrast, in the many-sorted logic of Enderton (2001, pp. 295–296), each sort i may have an equality symbol $=_i$, and $\tau_1 =_i \tau_2$ is an atom if the sort of both τ_1 and τ_2 is i. An introduction to the foundations of the real number system is provided by Eves (1990, pp. 173–211). Our axiomatization of lists is based on that of Manna and Waldinger (1985, pp. 437–469). See also Green's (1969, pp. 106–108) axiomatization of Lisp. Second-order logic is treated by Ebbinghaus et al. (1994, pp. 138–142), Mendelson (1997, pp. 368–382), Enderton (2001, pp. 282–306), and van Benthem and Doets (2001, pp. 200–216).

APPENDIX B

Equivalence of EC and DEC

In this appendix, we prove that EC and DEC are logically equivalent if the timepoint sort is restricted to the integers.[1] Our proof is structured as follows: We must show that EC implies DEC and that DEC implies EC. It is easy to show that EC implies DEC. This follows by universal instantiation, substituting $t_1 + 1$ for t_2. Showing that DEC implies EC is more difficult. We prove a number of lemmas stating that individual EC axioms follow from DEC.

LEMMA B.1 If the timepoint sort is restricted to the integers, then

$$\text{DEC} \Rightarrow \text{EC9}$$

Proof Suppose DEC. Let τ_1 and τ_2 be arbitrary integer timepoints and β be an arbitrary fluent. We must show

$$\begin{aligned}(\mathit{HoldsAt}(\beta, \tau_1) \wedge \tau_1 < \tau_2 \wedge \\ \mathit{PersistsBetween}(\tau_1, \beta, \tau_2) \wedge \neg \mathit{Clipped}(\tau_1, \beta, \tau_2)) \Rightarrow \\ \mathit{HoldsAt}(\beta, \tau_2)\end{aligned} \tag{B.1}$$

Case 1: $\tau_1 \geq \tau_2$. (B.1) is trivially satisfied.

Case 2: $\tau_1 < \tau_2$. We proceed by mathematical induction.

Base case. We show that (B.1) is true for $\tau_2 = \tau_1 + 1$. Suppose

$$\mathit{HoldsAt}(\beta, \tau_1) \wedge \mathit{PersistsBetween}(\tau_1, \beta, \tau_1 + 1) \wedge \neg \mathit{Clipped}(\tau_1, \beta, \tau_1 + 1)$$

---

1. This appendix is a slightly altered version of Mueller (2004a, sec. 3.2) in the *Journal of Logic and Computation*, published by Oxford University Press.

From $PersistsBetween(\tau_1, \beta, \tau_1 + 1)$ and definition EC7, we have $\neg ReleasedAt(\beta, \tau_1 + 1)$. From $\neg Clipped(\tau_1, \beta, \tau_1 + 1)$ and definition EC1, we have $\neg\exists a(Happens(a, \tau_1) \wedge Terminates(a, \beta, \tau_1))$. From this, $HoldsAt(\beta, \tau_1)$, $\neg ReleasedAt(\beta, \tau_1 + 1)$, and DEC5, we have $HoldsAt(\beta, \tau_1 + 1)$ as required.

Induction step. Suppose (B.1) is true for $\tau_2 = k$, $k > \tau_1$ (induction hypothesis):

$$(HoldsAt(\beta, \tau_1) \wedge \tau_1 < k \wedge PersistsBetween(\tau_1, \beta, k) \wedge \neg Clipped(\tau_1, \beta, k)) \Rightarrow HoldsAt(\beta, k) \tag{B.2}$$

We must show that (B.1) is true for $\tau_2 = k + 1$. Suppose

$$HoldsAt(\beta, \tau_1) \wedge \tau_1 < k + 1 \wedge PersistsBetween(\tau_1, \beta, k + 1) \wedge \neg Clipped(\tau_1, \beta, k + 1)$$

From $PersistsBetween(\tau_1, \beta, k + 1)$ and definition EC7, we have $PersistsBetween(\tau_1, \beta, k)$. From $\neg Clipped(\tau_1, \beta, k + 1)$ and definition EC1, we have $\neg Clipped(\tau_1, \beta, k)$. From this, $HoldsAt(\beta, \tau_1)$, $\tau_1 < k$, $PersistsBetween(\tau_1, \beta, k)$, and the induction hypothesis (B.2), we have $HoldsAt(\beta, k)$. From $PersistsBetween(\tau_1, \beta, k + 1)$ and definition EC7, we have $\neg ReleasedAt(\beta, k + 1)$. From $\neg Clipped(\tau_1, \beta, k + 1)$ and definition EC1, we have $\neg\exists a(Happens(a, k) \wedge Terminates(a, \beta, k))$. From this, $HoldsAt(\beta, k)$, $\neg ReleasedAt(\beta, k + 1)$, and DEC5, we have $HoldsAt(\beta, k + 1)$ as required. ■

LEMMA B.2 If the timepoint sort is restricted to the integers, then

$$\text{DEC} \Rightarrow \text{EC10}$$

Proof The proof is identical to that of Lemma B.1, except that $\neg HoldsAt$ is substituted for *HoldsAt*, *Inititates* is substituted for *Terminates*, DEC6 is substituted for DEC5, *Declipped* is substituted for *Clipped*, EC2 is substituted for EC1, and EC10 is substituted for EC9. ■

LEMMA B.3 If the timepoint sort is restricted to the integers, then

$$\text{DEC} \Rightarrow \text{EC11}$$

Proof Suppose DEC. Let τ_1 and τ_2 be arbitrary integer timepoints and β be an arbitrary fluent. We must show

$$(ReleasedAt(\beta, \tau_1) \wedge \tau_1 < \tau_2 \wedge \neg Clipped(\tau_1, \beta, \tau_2) \wedge \neg Declipped(\tau_1, \beta, \tau_2)) \Rightarrow ReleasedAt(\beta, \tau_2) \tag{B.3}$$

Case 1: $\tau_1 \geq \tau_2$. (B.3) is trivially satisfied.

Case 2: $\tau_1 < \tau_2$. We proceed by mathematical induction.

Base case. We show that (B.3) is true for $\tau_2 = \tau_1 + 1$. Suppose

$$\begin{aligned} &\mathit{ReleasedAt}(\beta, \tau_1) \wedge \tau_1 < \tau_1 + 1 \wedge \\ &\neg \mathit{Clipped}(\tau_1, \beta, \tau_1 + 1) \wedge \neg \mathit{Declipped}(\tau_1, \beta, \tau_1 + 1) \end{aligned}$$

From $\neg \mathit{Declipped}(\tau_1, \beta, \tau_1 + 1)$ and definition EC2, we have $\neg \exists a(\mathit{Happens}(a, \tau_1) \wedge \mathit{Initiates}(a, \beta, \tau_1))$. From $\neg \mathit{Clipped}(\tau_1, \beta, \tau_1 + 1)$ and definition EC1, we have $\neg \exists a(\mathit{Happens}(a, \tau_1) \wedge \mathit{Terminates}(a, \beta, \tau_1))$. From $\mathit{ReleasedAt}(\beta, \tau_1)$, $\neg \exists a(\mathit{Happens}(a, \tau_1) \wedge \mathit{Initiates}(a, \beta, \tau_1))$, $\neg \exists a(\mathit{Happens}(a, \tau_1) \wedge \mathit{Terminates}(a, \beta, \tau_1))$, and DEC7, we have $\mathit{ReleasedAt}(\beta, \tau_1 + 1)$ as required.

Induction step. Suppose (B.3) is true for $\tau_2 = k$, $k > \tau_1$ (induction hypothesis):

$$\begin{aligned} &(\mathit{ReleasedAt}(\beta, \tau_1) \wedge \tau_1 < k \wedge \\ &\neg \mathit{Clipped}(\tau_1, \beta, k) \wedge \neg \mathit{Declipped}(\tau_1, \beta, k)) \Rightarrow \\ &\mathit{ReleasedAt}(\beta, k) \end{aligned} \tag{B.4}$$

We must show that (B.3) is true for $\tau_2 = k + 1$. Suppose

$$\begin{aligned} &\mathit{ReleasedAt}(\beta, \tau_1) \wedge \tau_1 < k + 1 \wedge \\ &\neg \mathit{Clipped}(\tau_1, \beta, k + 1) \wedge \neg \mathit{Declipped}(\tau_1, \beta, k + 1) \end{aligned}$$

From $\neg \mathit{Clipped}(\tau_1, \beta, k + 1)$ and definition EC1, we have $\neg \mathit{Clipped}(\tau_1, \beta, k)$. From $\neg \mathit{Declipped}(\tau_1, \beta, k+1)$ and definition EC2, we have $\neg \mathit{Declipped}(\tau_1, \beta, k)$. From this, $\mathit{ReleasedAt}(\beta, \tau_1)$, $\tau_1 < k$, $\neg \mathit{Clipped}(\tau_1, \beta, k)$, and the induction hypothesis (B.4), we have $\mathit{ReleasedAt}(\beta, k)$. From $\neg \mathit{Declipped}(\tau_1, \beta, k + 1)$ and definition EC2, we have $\neg \exists a(\mathit{Happens}(a, k) \wedge \mathit{Initiates}(a, \beta, k))$. From $\neg \mathit{Clipped}(\tau_1, \beta, k + 1)$ and definition EC1, we have $\neg \exists a(\mathit{Happens}(a, k) \wedge \mathit{Terminates}(a, \beta, k))$. From $\mathit{ReleasedAt}(\beta, k)$, $\neg \exists a(\mathit{Happens}(a, k) \wedge \mathit{Initiates}(a, \beta, k))$, $\neg \exists a(\mathit{Happens}(a, k) \wedge \mathit{Terminates}(a, \beta, k))$, and DEC7, we have $\mathit{ReleasedAt}(\beta, k + 1)$ as required. ■

The following lemma is used in Theorem B.1 (p. 350) as well as Lemmas B.5 and B.6.

LEMMA B.4

If the timepoint sort is restricted to the integers, then

$$\mathrm{DEC} \Rightarrow \mathrm{EC12}$$

Proof Suppose DEC. Let τ_1 and τ_2 be arbitrary integer timepoints and β be an arbitrary fluent. We must show

$$(\neg ReleasedAt(\beta, \tau_1) \wedge \tau_1 < \tau_2 \wedge \neg ReleasedBetween(\tau_1, \beta, \tau_2)) \Rightarrow \neg ReleasedAt(\beta, \tau_2) \quad \text{(B.5)}$$

Case 1: $\tau_1 \geq \tau_2$. (B.5) is trivially satisfied.

Case 2: $\tau_1 < \tau_2$. We proceed by mathematical induction.

Base case. We show that (B.5) is true for $\tau_2 = \tau_1 + 1$. Suppose

$$\neg ReleasedAt(\beta, \tau_1) \wedge \tau_1 < \tau_1 + 1 \wedge \neg ReleasedBetween(\tau_1, \beta, \tau_1 + 1)$$

From $\neg ReleasedBetween(\tau_1, \beta, \tau_1 + 1)$ and definition EC8, we have $\neg\exists a(Happens(a, \tau_1) \wedge Releases(a, \beta, \tau_1))$. From this, $\neg ReleasedAt(\beta, \tau_1)$, and DEC8, we have $\neg ReleasedAt(\beta, \tau_1 + 1)$ as required.

Induction step. Suppose (B.5) is true for $\tau_2 = k$, $k > \tau_1$ (induction hypothesis):

$$(\neg ReleasedAt(\beta, \tau_1) \wedge \tau_1 < k \wedge \neg ReleasedBetween(\tau_1, \beta, k)) \Rightarrow \neg ReleasedAt(\beta, k) \quad \text{(B.6)}$$

We must show that (B.5) is true for $\tau_2 = k + 1$. Suppose

$$\neg ReleasedAt(\beta, \tau_1) \wedge \tau_1 < k + 1 \wedge \neg ReleasedBetween(\tau_1, \beta, k + 1)$$

From $\neg ReleasedBetween(\tau_1, \beta, k + 1)$ and definition EC8, we have $\neg ReleasedBetween(\tau_1, \beta, k)$. From this, $\neg ReleasedAt(\beta, \tau_1)$, $\tau_1 < k$, and the induction hypothesis (B.6), we have $\neg ReleasedAt(\beta, k)$. From $\neg ReleasedBetween(\tau_1, \beta, k + 1)$ and definition EC8, we have $\neg\exists a(Happens(a, k) \wedge Releases(a, \beta, k))$. From this, $\neg ReleasedAt(\beta, k)$, and DEC8, we have $\neg ReleasedAt(\beta, k + 1)$ as required. ■

LEMMA B.5

If the timepoint sort is restricted to the integers, then

$$\text{DEC} \Rightarrow \text{EC14}$$

Proof Suppose DEC. Let τ_1 and τ_2 be arbitrary integer timepoints, α be an arbitrary event, and β be an arbitrary fluent. We must show

$$(Happens(\alpha, \tau_1) \wedge Initiates(\alpha, \beta, \tau_1) \wedge \tau_1 < \tau_2 \wedge \neg StoppedIn(\tau_1, \beta, \tau_2) \wedge \neg ReleasedIn(\tau_1, \beta, \tau_2)) \Rightarrow HoldsAt(\beta, \tau_2) \quad \text{(B.7)}$$

Case 1: $\tau_1 \geq \tau_2$. (B.7) is trivially satisfied.

Case 2: $\tau_1 < \tau_2$. We proceed by mathematical induction.

Base case. We show that (B.7) is true for $\tau_2 = \tau_1 + 1$. Suppose

$$\begin{aligned} &\mathit{Happens}(\alpha, \tau_1) \wedge \mathit{Initiates}(\alpha, \beta, \tau_1) \wedge \tau_1 < \tau_1 + 1 \wedge \\ &\neg \mathit{StoppedIn}(\tau_1, \beta, \tau_1 + 1) \wedge \neg \mathit{ReleasedIn}(\tau_1, \beta, \tau_1 + 1) \end{aligned}$$

From this and DEC9, we have $\mathit{HoldsAt}(\beta, \tau_1 + 1)$, as required.

Induction step. Suppose (B.7) is true for $\tau_2 = k$, $k > \tau_1$ (induction hypothesis):

$$\begin{aligned} &(\mathit{Happens}(\alpha, \tau_1) \wedge \mathit{Initiates}(\alpha, \beta, \tau_1) \wedge \tau_1 < k \wedge \\ &\neg \mathit{StoppedIn}(\tau_1, \beta, k) \wedge \neg \mathit{ReleasedIn}(\tau_1, \beta, k)) \Rightarrow \\ &\mathit{HoldsAt}(\beta, k) \end{aligned} \tag{B.8}$$

We must show that (B.7) is true for $\tau_2 = k + 1$. Suppose

$$\begin{aligned} &\mathit{Happens}(\alpha, \tau_1) \wedge \mathit{Initiates}(\alpha, \beta, \tau_1) \wedge \tau_1 < k + 1 \wedge \\ &\neg \mathit{StoppedIn}(\tau_1, \beta, k + 1) \wedge \neg \mathit{ReleasedIn}(\tau_1, \beta, k + 1) \end{aligned}$$

From $\mathit{Happens}(\alpha, \tau_1)$, $\mathit{Initiates}(\alpha, \beta, \tau_1)$, and DEC9, we have $\mathit{HoldsAt}(\beta, \tau_1 + 1)$. From $\neg \mathit{StoppedIn}(\tau_1, \beta, k+1)$ and definition EC3, we have $\neg \mathit{StoppedIn}(\tau_1, \beta, k)$. From $\neg \mathit{ReleasedIn}(\tau_1, \beta, k + 1)$ and definition EC13, we have $\neg \mathit{ReleasedIn}(\tau_1, \beta, k)$. From $\mathit{Happens}(\alpha, \tau_1)$, $\mathit{Initiates}(\alpha, \beta, \tau_1)$, $\tau_1 < k$, $\neg \mathit{StoppedIn}(\tau_1, \beta, k)$, $\neg \mathit{ReleasedIn}(\tau_1, \beta, k)$, and the induction hypothesis (B.8), we have $\mathit{HoldsAt}(\beta, k)$. From $\mathit{Happens}(\alpha, \tau_1)$, $\mathit{Initiates}(\alpha, \beta, \tau_1)$, and DEC12, we have $\neg \mathit{ReleasedAt}(\beta, \tau_1 + 1)$. From $\neg \mathit{ReleasedIn}(\tau_1, \beta, k + 1)$, definition EC13, and definition EC8, we have $\neg \mathit{ReleasedBetween}(\tau_1 + 1, \beta, k + 1)$. From Lemma B.4 and DEC, we have EC12. From $\neg \mathit{ReleasedAt}(\beta, \tau_1 + 1)$, $\tau_1 + 1 < k + 1$, $\neg \mathit{ReleasedBetween}(\tau_1 + 1, \beta, k + 1)$, and EC12, we have $\neg \mathit{ReleasedAt}(\beta, k + 1)$. From $\neg \mathit{StoppedIn}(\tau_1, \beta, k + 1)$ and definition EC3, we have $\neg \exists a(\mathit{Happens}(a, k) \wedge \mathit{Terminates}(a, \beta, k))$. From this, $\mathit{HoldsAt}(\beta, k)$, $\neg \mathit{ReleasedAt}(\beta, k + 1)$, and DEC5, we have $\mathit{HoldsAt}(\beta, k + 1)$ as required. ■

LEMMA B.6 If the timepoint sort is restricted to the integers, then

$$\text{DEC} \Rightarrow \text{EC15}$$

Proof The proof is identical to that of Lemma B.5, except that $\neg \mathit{HoldsAt}$ is substituted for *HoldsAt*, *Terminates* is substituted for *Inititates*, DEC10 is substituted for DEC9, *StartedIn* is substituted for *StoppedIn*, EC4 is substituted for EC3, and DEC6 is substituted for DEC5. ■

LEMMA B.7 If the timepoint sort is restricted to the integers, then

$$\text{DEC} \Rightarrow \text{EC16}$$

Proof Suppose DEC. Let τ_1 and τ_2 be arbitrary integer timepoints, α be an arbitrary event, and β be an arbitrary fluent. We must show

$$\begin{aligned}(&\mathit{Happens}(\alpha, \tau_1) \wedge \mathit{Releases}(\alpha, \beta, \tau_1) \wedge \tau_1 < \tau_2 \wedge \\ &\neg\mathit{StoppedIn}(\tau_1, \beta, \tau_2) \wedge \neg\mathit{StartedIn}(\tau_1, \beta, \tau_2)) \Rightarrow \\ &\qquad\qquad \mathit{ReleasedAt}(\beta, \tau_2)\end{aligned} \tag{B.9}$$

Case 1: $\tau_1 \geq \tau_2$. (B.9) is trivially satisfied.

Case 2: $\tau_1 < \tau_2$. We proceed by mathematical induction.

Base case. We show that (B.9) is true for $\tau_2 = \tau_1 + 1$. Suppose

$$\begin{aligned}&\mathit{Happens}(\alpha, \tau_1) \wedge \mathit{Releases}(\alpha, \beta, \tau_1) \wedge \tau_1 < \tau_1 + 1 \wedge \\ &\neg\mathit{StoppedIn}(\tau_1, \beta, \tau_1 + 1) \wedge \neg\mathit{StartedIn}(\tau_1, \beta, \tau_1 + 1)\end{aligned}$$

From $\mathit{Happens}(\alpha, \tau_1)$, $\mathit{Releases}(\alpha, \beta, \tau_1)$, and DEC11, we have $\mathit{ReleasedAt}(\beta, \tau_1 + 1)$, as required.

Induction step. Suppose (B.9) is true for $\tau_2 = k$, $k > \tau_1$ (induction hypothesis):

$$\begin{aligned}(&\mathit{Happens}(\alpha, \tau_1) \wedge \mathit{Releases}(\alpha, \beta, \tau_1) \wedge \tau_1 < k \wedge \\ &\neg\mathit{StoppedIn}(\tau_1, \beta, k) \wedge \neg\mathit{StartedIn}(\tau_1, \beta, k)) \Rightarrow \\ &\qquad\qquad \mathit{ReleasedAt}(\beta, k)\end{aligned} \tag{B.10}$$

We must show that (B.9) is true for $\tau_2 = k + 1$. Suppose

$$\begin{aligned}&\mathit{Happens}(\alpha, \tau_1) \wedge \mathit{Releases}(\alpha, \beta, \tau_1) \wedge \tau_1 < k + 1 \wedge \\ &\neg\mathit{StoppedIn}(\tau_1, \beta, k + 1) \wedge \neg\mathit{StartedIn}(\tau_1, \beta, k + 1)\end{aligned}$$

From $\mathit{Happens}(\alpha, \tau_1)$, $\mathit{Releases}(\alpha, \beta, \tau_1)$, and DEC11, we have $\mathit{ReleasedAt}(\beta, \tau_1 + 1)$. From $\neg\mathit{StoppedIn}(\tau_1, \beta, k + 1)$ and definition EC3, we have $\neg\mathit{StoppedIn}(\tau_1, \beta, k)$. From $\neg\mathit{StartedIn}(\tau_1, \beta, k + 1)$ and definition EC4, we have $\neg\mathit{StartedIn}(\tau_1, \beta, k)$. From $\mathit{Happens}(\alpha, \tau_1)$, $\mathit{Releases}(\alpha, \beta, \tau_1)$, $\tau_1 < k$, $\neg\mathit{StoppedIn}(\tau_1, \beta, k)$, $\neg\mathit{StartedIn}(\tau_1, \beta, k)$, and the induction hypothesis (B.10), we have $\mathit{ReleasedAt}(\beta, k)$. From $\neg\mathit{StoppedIn}(\tau_1, \beta, k+1)$ and definition EC3, we have $\neg\exists a(\mathit{Happens}(a, k) \wedge \mathit{Terminates}(a, \beta, k))$. From $\neg\mathit{StartedIn}(\tau_1, \beta, k + 1)$ and definition EC4, we have $\neg\exists a(\mathit{Happens}(a, k) \wedge \mathit{Initiates}(a, \beta, k))$. From this, $\mathit{ReleasedAt}(\beta, k)$, $\neg\exists a(\mathit{Happens}(a, k) \wedge \mathit{Terminates}(a, \beta, k))$, and DEC7, we have $\mathit{ReleasedAt}(\beta, k + 1)$ as required. ■

LEMMA B.8 If the timepoint sort is restricted to the integers, then

$$\text{DEC} \Rightarrow \text{EC17}$$

Proof Suppose DEC. Let τ_1 and τ_2 be arbitrary integer timepoints, α be an arbitrary event, and β be an arbitrary fluent. We must show

$$\begin{aligned}(Happens(\alpha, \tau_1) \wedge (Initiates(\alpha, \beta, \tau_1) \vee Terminates(\alpha, \beta, \tau_1)) \wedge \quad \text{(B.11)}\\ \tau_1 < \tau_2 \wedge \neg ReleasedIn(\tau_1, \beta, \tau_2)) \Rightarrow\\ \neg ReleasedAt(\beta, \tau_2)\end{aligned}$$

Case 1: $\tau_1 \geq \tau_2$. (B.11) is trivially satisfied.

Case 2: $\tau_1 < \tau_2$. We proceed by mathematical induction.

Base case. We show that (B.11) is true for $\tau_2 = \tau_1 + 1$. Suppose

$$\begin{aligned}Happens(\alpha, \tau_1) \wedge (Initiates(\alpha, \beta, \tau_1) \vee Terminates(\alpha, \beta, \tau_1)) \wedge\\ \tau_1 < \tau_1 + 1 \wedge \neg ReleasedIn(\tau_1, \beta, \tau_1 + 1)\end{aligned}$$

From $Happens(\alpha, \tau_1)$, $(Initiates(\alpha, \beta, \tau_1) \vee Terminates(\alpha, \beta, \tau_1))$, and DEC12, we have $\neg ReleasedAt(\beta, \tau_1 + 1)$, as required.

Induction step. Suppose (B.11) is true for $\tau_2 = k$, $k > \tau_1$ (induction hypothesis):

$$\begin{aligned}(Happens(\alpha, \tau_1) \wedge (Initiates(\alpha, \beta, \tau_1) \vee Terminates(\alpha, \beta, \tau_1)) \wedge \quad \text{(B.12)}\\ \tau_1 < k \wedge \neg ReleasedIn(\tau_1, \beta, k)) \Rightarrow\\ \neg ReleasedAt(\beta, k)\end{aligned}$$

We must show that (B.11) is true for $\tau_2 = k + 1$. Suppose

$$\begin{aligned}Happens(\alpha, \tau_1) \wedge (Initiates(\alpha, \beta, \tau_1) \vee Terminates(\alpha, \beta, \tau_1)) \wedge\\ \tau_1 < k + 1 \wedge \neg ReleasedIn(\tau_1, \beta, k + 1)\end{aligned}$$

From $Happens(\alpha, \tau_1)$, $Initiates(\alpha, \beta, \tau_1) \vee Terminates(\alpha, \beta, \tau_1)$, and DEC12, we have $\neg ReleasedAt(\beta, \tau_1 + 1)$. From $\neg ReleasedIn(\tau_1, \beta, k + 1)$ and definition EC13, we have $\neg ReleasedIn(\tau_1, \beta, k)$. From this, $Happens(\alpha, \tau_1)$, $Initiates(\alpha, \beta, \tau_1) \vee Terminates(\alpha, \beta, \tau_1)$, $\tau_1 < k$, and the induction hypothesis (B.12), we have $\neg ReleasedAt(\beta, k)$. From $\neg ReleasedIn(\tau_1, \beta, k + 1)$ and definition EC13, we have $\neg \exists a(Happens(a, k) \wedge Releases(a, \beta, k))$. From this, $\neg ReleasedAt(\beta, k)$, and DEC8, we have $\neg ReleasedAt(\beta, k + 1)$ as required. ■

Now we proceed to the equivalence theorem.

THEOREM B.1 If the timepoint sort is restricted to the integers, then

$$\text{EC} \Leftrightarrow \text{DEC}$$

Proof We prove the two directions separately.

(EC $\Rightarrow$ DEC). Suppose EC. DEC3 is identical to EC5 and DEC4 is identical to EC6. DEC5 follows from EC9 by universal instantiation, substituting $t_1 + 1$ for t_2. Similarly, DEC6, DEC7, DEC8, DEC9, DEC10, DEC11, and DEC12 follow from EC10, EC11, EC12, EC14, EC15, EC16, and EC17, respectively, by universal instantiation, substituting $t_1 + 1$ for t_2.

(EC $\Leftarrow$ DEC). Suppose DEC. EC5 is identical to DEC3 and EC6 is identical to DEC4. EC9, EC10, EC11, EC12, EC14, EC15, EC16, and EC17 follow from Lemmas B.1, B.2, B.3, B.4, B.5, B.6, B.7, and B.8, respectively. ■

APPENDIX C

Events with Duration

In most versions of the event calculus, event occurrences are instantaneous. This appendix presents a revised version of EC that allows event occurrences with duration. A three-argument *Happens* predicate $Happens3(e, t_1, t_2)$ is introduced, representing that event e occurs between t_1 and t_2. EC is revised as follows.

DEFINITION EC1′

$$Clipped(t_1, f, t_4) \stackrel{\text{def}}{\equiv}$$
$$\exists e, t_2, t_3\ (Happens3(e, t_2, t_3) \wedge t_1 \leq t_3 \wedge t_2 < t_4 \wedge Terminates(e, f, t_2))$$

DEFINITION EC2′

$$Declipped(t_1, f, t_4) \stackrel{\text{def}}{\equiv}$$
$$\exists e, t_2, t_3\ (Happens3(e, t_2, t_3) \wedge t_1 \leq t_3 \wedge t_2 < t_4 \wedge Initiates(e, f, t_2))$$

DEFINITION EC3′

$$StoppedIn(t_1, f, t_4) \stackrel{\text{def}}{\equiv}$$
$$\exists e, t_2, t_3\ (Happens3(e, t_2, t_3) \wedge t_1 < t_3 \wedge t_2 < t_4 \wedge Terminates(e, f, t_2))$$

DEFINITION EC4′

$$StartedIn(t_1, f, t_4) \stackrel{\text{def}}{\equiv}$$
$$\exists e, t_2, t_3\ (Happens3(e, t_2, t_3) \wedge t_1 < t_3 \wedge t_2 < t_4 \wedge Initiates(e, f, t_2))$$

AXIOM EC5′

$$(Happens3(e, t_1, t_2) \wedge Initiates(e, f_1, t_1) \wedge 0 < t_3 \wedge$$
$$Trajectory(f_1, t_1, f_2, t_3) \wedge \neg StoppedIn(t_1, f_1, t_2 + t_3)) \Rightarrow$$
$$HoldsAt(f_2, t_2 + t_3)$$

AXIOM EC6′

$$(Happens3(e, t_1, t_2) \wedge Terminates(e, f_1, t_1) \wedge 0 < t_3 \wedge$$
$$AntiTrajectory(f_1, t_1, f_2, t_3) \wedge \neg StartedIn(t_1, f_1, t_2 + t_3)) \Rightarrow$$
$$HoldsAt(f_2, t_2 + t_3)$$

DEFINITION EC8′

$$ReleasedBetween(t_1, f, t_4) \stackrel{\text{def}}{\equiv}$$
$$\exists e, t_2, t_3\ (Happens3(e, t_2, t_3) \wedge t_1 \leq t_3 \wedge t_2 < t_4 \wedge Releases(e, f, t_2))$$

DEFINITION EC13′
$$ReleasedIn(t_1, f, t_4) \stackrel{\text{def}}{\equiv} \exists e, t_2, t_3\,(Happens3(e, t_2, t_3) \wedge t_1 < t_3 \wedge t_2 < t_4 \wedge Releases(e, f, t_2))$$

AXIOM EC14′
$$(Happens3(e, t_1, t_2) \wedge Initiates(e, f, t_1) \wedge t_2 < t_3 \wedge \neg StoppedIn(t_1, f, t_3) \wedge \neg ReleasedIn(t_1, f, t_3)) \Rightarrow HoldsAt(f, t_3)$$

AXIOM EC15′
$$(Happens3(e, t_1, t_2) \wedge Terminates(e, f, t_1) \wedge t_2 < t_3 \wedge \neg StartedIn(t_1, f, t_3) \wedge \neg ReleasedIn(t_1, f, t_3)) \Rightarrow \neg HoldsAt(f, t_3)$$

AXIOM EC16′
$$(Happens3(e, t_1, t_2) \wedge Releases(e, f, t_1) \wedge t_2 < t_3 \wedge \neg StoppedIn(t_1, f, t_3) \wedge \neg StartedIn(t_1, f, t_3)) \Rightarrow ReleasedAt(f, t_3)$$

AXIOM EC17′
$$(Happens3(e, t_1, t_2) \wedge (Initiates(e, f, t_1) \vee Terminates(e, f, t_1)) \wedge t_2 < t_3 \wedge \neg ReleasedIn(t_1, f, t_3)) \Rightarrow \neg ReleasedAt(f, t_3)$$

AXIOM EC18
$$Happens3(e, t_1, t_2) \Rightarrow t_1 \leq t_2$$

Definition EC7 and axioms EC9, EC10, EC11, and EC12 remain unchanged. Two-argument *Happens* is defined in terms of three-argument *Happens* as follows.

AXIOM EC19
$$Happens(e, t) \Leftarrow Happens3(e, t, t)$$

Instead of representing events with duration as events, they may be represented as fluents, as discussed in Section 7.1.2.

Once events have duration, it becomes possible to represent compound events. For example, we may represent that event *E1* is a compound event that consists of event *E2* immediately followed by *E3* as follows:

$$Happens3(E1, t_1, t_3) \Leftrightarrow \exists t_2\,(t_1 < t_2 < t_3 \wedge Happens3(E2, t_1, t_2) \wedge Happens3(E3, t_2, t_3))$$

Bibliographic Notes

Three-argument *Happens* for representing event occurrences with duration and event calculus axioms for event occurrences with duration were introduced by Shanahan (1997a, 1999a) based on previous work (R. Miller and Shanahan, 1994). Shanahan (1997a, 1999a) presents examples of how to use three-argument *Happens* to represent compound events.

APPENDIX D

Answers to Selected Exercises

Chapter 2

2.2. The expression $U[Awake, Asleep]$ is an abbreviation for the conjunction of the following formulas:

$$Awake(a_1) \neq Asleep(a_2)$$
$$Awake(a_1) = Awake(a_2) \Rightarrow a_1 = a_2$$
$$Asleep(a_1) = Asleep(a_2) \Rightarrow a_1 = a_2$$

2.3. The following formulas are positive relative to Q:

$$Q(A)$$
$$P(x) \Rightarrow Q(x)$$

The following formulas are not positive relative to Q:

$$\neg Q(A)$$
$$P(x) \Rightarrow \neg Q(x)$$
$$Q(x) \Rightarrow P(x)$$
$$Q(x) \Leftrightarrow P(x)$$
$$Q(A) \Rightarrow Q(B)$$

Chapter 3

3.2. We use a positive effect axiom. The axiom states that if there exists a room such that agent 1 and agent 2 are in that room, agent 2 is listening to agent 1, and

agent 1 tells agent 2 a fact, then agent 2 will know that fact:

$$(\exists r\, HoldsAt(InRoom(a_1, r), t) \wedge HoldsAt(InRoom(a_2, r), t)) \wedge HoldsAt(ListeningTo(a_2, a_1), t) \Rightarrow Initiates(Tell(a_1, a_2, f), Know(a_2, f), t)$$

We might also wish to add a state constraint that says that an agent is in one room at a time:

$$HoldsAt(InRoom(a, r_1), t) \wedge HoldsAt(InRoom(a, r_2), t) \Rightarrow r_1 = r_2$$

3.11. The scenario in which two agents dial one another simultaneously does not expose a bug. If the two agents pick up their respective phones and then dial one another simultaneously, they will both get busy signals. If the two agents dial one another simultaneously without first picking up their phones, then nothing happens. The scenario in which one agent dials another agent just as the other agent picks up the phone exposes a bug in the formalization. We can show $HoldsAt(Ringing(Phone1, Phone2), T)$ and $HoldsAt(DialTone(Phone2), T)$, which contradict the state constraint (3.49).

Chapter 4

4.1. First we add an effect axiom that states that if a clock is beeping and an agent presses the clock's snooze button, then the clock's alarm time will be the current time plus 9:

$$HoldsAt(Beeping(c), t) \Rightarrow Initiates(PressSnooze(a, c), AlarmTime(c, t + 9), t)$$

Then we add an effect axiom that states that if a clock is beeping, the clock's alarm time is t_1, and t_1 is not already the current time plus 9, then the clock's alarm time will no longer be t_1:

$$HoldsAt(Beeping(c), t) \wedge HoldsAt(AlarmTime(c, t_1), t) \wedge t_1 \neq t + 9 \Rightarrow Terminates(PressSnooze(a, c), AlarmTime(c, t_1), t)$$

Finally, we add an effect axiom that states that if an agent presses a clock's snooze button, then the clock will not be beeping:

$$Terminates(PressSnooze(a, c), Beeping(c), t)$$

Pressing the snooze button when the clock is not beeping has no effect.

Chapter 5

5.3. From CIRC[Σ; *Initiates, Terminates, Releases*], Theorem 2.2, and Theorem 2.1, we have

$$\begin{aligned} &Initiates(e,f,t) \Leftrightarrow \\ &\exists a,g\,(e = Load(a,g) \wedge f = Loaded(g)) \end{aligned} \tag{D.1}$$

$$\begin{aligned} &Terminates(e,f,t) \Leftrightarrow \\ &\exists a_1,a_2,g\,(e = Shoot(a_1,a_2,g) \wedge \\ &f = Alive(a_2) \wedge \\ &HoldsAt(Loaded(g),t)) \vee \\ &\exists a_1,a_2,g\,(e = Shoot(a_1,a_2,g) \wedge \\ &f = Loaded(g) \wedge \\ &HoldsAt(Loaded(g),t)) \end{aligned} \tag{D.2}$$

$$\neg Releases(e,f,t) \tag{D.3}$$

From CIRC[Δ; *Happens*] and Theorem 2.1, we have

$$\begin{aligned} &Happens(e,t) \Leftrightarrow \\ &(e = Load(Nathan, Gun) \wedge t = 0) \vee \\ &(e = Wait(Nathan) \wedge t = 1) \vee \\ &(e = Shoot(Nathan, Turkey, Gun) \wedge t = 2) \end{aligned} \tag{D.4}$$

From (5.21) (which follows from (D.4)), (5.16) (which follows from (D.1)), and (5.10), we have

$$HoldsAt(Loaded(Gun), 1) \tag{D.5}$$

From (D.4) and (D.2), we have $\neg\exists e\,(Happens(e,1) \wedge Terminates(e, Loaded(Gun), 1))$. From this, (D.5), and (5.14), we have

$$HoldsAt(Loaded(Gun), 2)$$

From this, (5.23) (which follows from (D.4)), (5.17) (which follows from (D.2)), and (5.15), we have $\neg HoldsAt(Alive(Turkey), 3)$, as required.

5.4. From *CIRC*[Σ; *Initiates, Terminates, Releases*], Theorem 2.2, and Theorem 2.1, we have

$$\begin{aligned} &Initiates(e,f,t) \Leftrightarrow \\ &\exists a,g\,(e = Load(a,g) \wedge f = Loaded(g)) \end{aligned} \tag{D.6}$$

$$\begin{aligned} &Terminates(e,f,t) \Leftrightarrow \\ &\exists a_1,a_2,g\,(e = Shoot(a_1,a_2,g) \wedge \\ &f = Alive(a_2) \wedge \\ &HoldsAt(Loaded(g),t)) \vee \\ &\exists a_1,a_2,g\,(e = Shoot(a_1,a_2,g) \wedge \\ &f = Loaded(g) \wedge \\ &HoldsAt(Loaded(g),t)) \end{aligned} \tag{D.7}$$

$$\begin{aligned} &Releases(e,f,t) \Leftrightarrow \\ &\exists a,g\,(e = Spin(a,g) \wedge \\ &f = Loaded(g) \wedge \\ &HoldsAt(Loaded(g),t)) \end{aligned} \tag{D.8}$$

From *CIRC*[Δ; *Happens*] and Theorem 2.1, we have

$$\begin{aligned} &Happens(e,t) \Leftrightarrow \\ &(e = Load(Nathan, Gun) \wedge t = 0) \vee \\ &(e = Spin(Nathan, Gun) \wedge t = 1) \vee \\ &(e = Shoot(Nathan, Turkey, Gun) \wedge t = 2) \end{aligned} \tag{D.9}$$

We first show (5.34). From $HoldsAt(Loaded(Gun), 2)$, (5.23) (which follows from (D.9)), (5.17) (which follows from (D.7)), and (5.15), we have $\neg HoldsAt(Alive(Turkey), 3)$, as required.

Now we show (5.35). From (D.9) and (D.7), we have $\neg\exists e\,(Happens(e, 0) \wedge Terminates(e, Alive(Turkey), 0))$. From this, (5.19), (5.32), and (5.25), we have

$$HoldsAt(Alive(Turkey), 1) \tag{D.10}$$

From (D.9) and (D.7), we have $\neg\exists e\,(Happens(e, 1) \wedge Terminates(e, Alive(Turkey), 1))$. From this, (D.10), (5.32), and (5.25), we have

$$HoldsAt(Alive(Turkey), 2) \tag{D.11}$$

From (D.7) and $\neg HoldsAt(Loaded(Gun), 2)$, we have $\neg\exists e\,(Happens(e, 2) \wedge Terminates(e, Alive(Turkey), 2))$. From this, (D.9), (5.32), and (5.25), we have $HoldsAt(Alive(Turkey), 3)$, as required.

Chapter 6

6.4. We use a counter sort with variable c and an integer sort with variables n, n_1, and n_2. We use the following fluents.

Count(c, n) represents that the count of counter c is n.

InputLine(c) represents that the value of the input line of c is true *InputLine* is always released from the commonsense law of inertia.

True(c) represents that the current value of the input line of c is true. *True* is subject to the commonsense law of inertia. It is set to true whenever the input line changes to true and to false whenever the input line changes to false.

We use the following events.

FalseToTrue(c) represents that the input line of counter c changes from false to true.

TrueToFalse(c) represents that the input line of counter c changes from true to false.

We have two trigger axioms. The first states that, if the current value of the input line of a counter is false and the value of the input line of the counter is true, then the input line of the counter changes from false to true:

$$\begin{gathered}\neg HoldsAt(True(c), t) \wedge \\ HoldsAt(InputLine(c), t) \Rightarrow \\ Happens(FalseToTrue(c), t)\end{gathered}$$

The second states that, if the current value of the input line of a counter is true and the value of the input line of the counter is false, then the input line of the counter changes from true to false:

$$\begin{gathered}HoldsAt(True(c), t) \wedge \\ \neg HoldsAt(InputLine(c), t) \Rightarrow \\ Happens(TrueToFalse(c), t)\end{gathered}$$

We then have several effect axioms. If the value of the input line of a counter changes from false to true, then the current value of the input line of the counter is true:

$$Initiates(FalseToTrue(c), True(c), t)$$

If the value of the input line of a counter changes from true to false, then the current value of the input line of the counter is false:

$$Terminates(TrueToFalse(c), True(c), t)$$

If the count of a counter is n and the value of the input line of the counter changes from false to true, then the count of the counter will be $n + 1$ and will no longer be n:

$$HoldsAt(Count(c, n), t) \Rightarrow$$
$$Initiates(FalseToTrue(c), Count(c, n + 1), t)$$

$$HoldsAt(Count(c, n), t) \Rightarrow$$
$$Terminates(FalseToTrue(c), Count(c, n), t)$$

Last, we have a state constraint to ensure that a counter has a unique count:

$$HoldsAt(Count(c, n_1), t) \wedge$$
$$HoldsAt(Count(c, n_2), t) \Rightarrow$$
$$n_1 = n_2$$

Chapter 8

8.3. We use a sort for colors, with variables c, c_1, and c_2, and a sort for palettes with variable p. We use four effect axioms. If yellow and blue are placed on the palette simultaneously then green will be on the palette:

$$Happens(PlaceOnPalette(p, Yellow), t) \Rightarrow$$
$$Initiates(PlaceOnPalette(p, Blue), OnPalette(p, Green), t)$$

If yellow and blue are placed on the palette simultaneously and a color other than green is on the palette, then that color will no longer be on the palette:

$$Happens(PlaceOnPalette(p, Yellow), t) \wedge$$
$$HoldsAt(OnPalette(p, c), t) \wedge$$
$$c \neq Green \Rightarrow$$
$$Terminates(PlaceOnPalette(p, Blue), OnPalette(p, c), t)$$

If yellow and blue are not placed on the palette simultaneously and a color is placed on the palette, then that color will be on the palette:

$$\neg Happens(PlaceOnPalette(p, Yellow), t) \vee$$
$$\neg Happens(PlaceOnPalette(p, Blue), t) \Rightarrow$$
$$Initiates(PlaceOnPalette(p, c), OnPalette(p, c), t)$$

If yellow and blue are not placed on the palette simultaneously, color 1 is on the palette, and a color 2 that is different from color 1 is placed on the palette, then color 1 will no longer be on the palette:

$$\neg(Happens(PlaceOnPalette(p, Yellow), t) \wedge$$
$$Happens(PlaceOnPalette(p, Blue), t)) \wedge$$
$$HoldsAt(OnPalette(p, c_1), t) \wedge$$
$$c_1 \neq c_2 \Rightarrow$$
$$Terminates(PlaceOnPalette(p, c_2), OnPalette(p, c_1), t)$$

We use the following state constraint, which says that a unique color is on the palette:

$$HoldsAt(OnPalette(p, c_1), t) \wedge$$
$$HoldsAt(OnPalette(p, c_2), t) \Rightarrow$$
$$c_1 = c_2$$

References

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1983). *Data structures and algorithms*. Reading, MA: Addison-Wesley.

Aho, A. V., and Ullman, J. D. (1979). Universality of data retrieval languages. In *Conference record of the sixth annual ACM Symposium on Principles of Programming Languages* (pp. 110–120). New York: ACM.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM, 26*(11), 832–843.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence, 23*, 123–154.

Allen, J. F. (1991). Planning as temporal reasoning. In J. F. Allen, R. E. Fikes, and E. Sandewall (Eds.), *Proceedings of the second International Conference on Principles of Knowledge Representation and Reasoning* (pp. 3–14). San Mateo, CA: Morgan Kaufmann.

Allen, J. F. (1995). *Natural language understanding* (2nd ed.). Redwood City, CA: Benjamin/Cummings.

Allen, J. F., Hendler, J., and Tate, A. (Eds.). (1990). *Readings in planning*. San Mateo, CA: Morgan Kaufmann.

Alshawi, H. (Ed.). (1992). *The Core Language Engine*. Cambridge, MA: MIT Press.

Antoniou, G. (1997). *Nonmonotonic reasoning*. Cambridge, MA: MIT Press.

Armstrong, S. (Ed.). (1993). *Using large corpora*. Cambridge, MA: MIT Press.

Bacchus, F., Tenenberg, J., and Koomen, J. A. (1991). A non-reified temporal logic. *Artificial Intelligence, 52*, 87–108.

Baillargeon, R. (1995). Physical reasoning in infancy. In M. S. Gazzaniga (Ed.), *The cognitive neurosciences* (pp. 181–204). Cambridge, MA: MIT Press.

Baker, A. B. (1991). Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence, 49*(1–3), 5–23.

Baral, C. (1995). Reasoning about actions: Non-deterministic effects, constraints, and qualification. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 2017–2026). San Mateo, CA: Morgan Kaufmann.

Baral, C. (2003). *Knowledge representation, reasoning and declarative problem solving.* Cambridge, UK: Cambridge University Press.

Baral, C., and Gelfond, M. (1997). Reasoning about effects of concurrent actions. *Journal of Logic Programming, 31*(1–3), 85–117.

Baral, C., Tran, N., and Tuan, L.-C. (2002). Reasoning about actions in a probabilistic setting. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence* (pp. 507–512). Menlo Park, CA: AAAI Press.

Baum, E. B. (2004). *What is thought?* Cambridge, MA: MIT Press.

Bayardo R. J., Jr., and Schrag, R. C. (1997). Using CSP look-back techniques to solve real world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference* (pp. 203–208). Menlo Park, CA: AAAI Press.

Beale, S., Nirenburg, S., and Mahesh, K. (1995). Semantic analysis in the Mikrokosmos machine translation project. In *Proceedings of the Second Symposium on Natural Language Processing* (pp. 297–307). Bangkok, Thailand: Kasetsart University.

Beaudoin, L. (1994). *Goal processing in autonomous agents.* Unpublished doctoral dissertation, University of Birmingham, Birmingham, UK.

Bibel, W., and Nicolas, J.-M. (1989). The role of logic for data and knowledge bases: A brief survey. In J. W. Schmidt and C. Thanos (Eds.), *Foundations of knowledge base management* (pp. 3–22). Berlin: Springer.

Birnbaum, L. (1991). Rigor mortis: A response to Nilsson's "Logic and artificial intelligence." *Artificial Intelligence, 47*(1–3), 57–77.

Blum, A. L., and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence, 90*(1–2), 281–300.

Bobrow, D. G. (1980). Editor's preface [Special issue on non-monotonic reasoning]. *Artificial Intelligence, 13*(1–2).

Bobrow, D. G., and Winograd, T. (1977). An overview of KRL, a knowledge representation language. *Cognitive Science, 1*(1), 3–46.

Bornscheuer, S.-E., and Thielscher, M. (1996). *Explicit and implicit indeterminism: Reasoning about uncertain and contradictory specifications of dynamic systems* (Tech. Rep. No. TR-96-009). Berkeley, CA: International Computer Science Institute.

Brachman, R. J., and Levesque, H. J. (2004). *Knowledge representation and reasoning.* San Francisco: Morgan Kaufmann.

Braine, M. D. S. (1978). On the relation between the natural logic of reasoning and standard logic. *Psychological Review, 85*(1), 1–21.

Braine, M. D. S., and O'Brien, D. P. (Eds.). (1998). *Mental logic.* Mahwah, NJ: Lawrence Erlbaum.

Brandano, S. (2001). The event calculus assessed. In *Proceedings of the Eighth International Symposium on Temporal Representation and Reasoning (TIME-01)* (pp. 7–12). Los Alamitos, CA: IEEE Computer Society.

Bratman, M. E. (1987). *Intention, plans, and practical reason.* Cambridge, MA: Harvard University Press.

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence, 4*, 349–355.

Brewka, G., Dix, J., and Konolige, K. (1997). *Nonmonotonic reasoning: An overview*. Stanford, CA: CSLI.

Briggs, R. (1999). *The snowman*. New York: Random House.

Brooks, R. A. (1985). *A robust layered control system for a mobile robot* (A. I. Memo 864). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Brooks, R. A. (1990). *The behavior language; User's guide*. A. I. Memo 1227, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Brown, F. M. (Ed.). (1987). *The frame problem in artificial intelligence: proceedings of the 1987 workshop*. Los Altos, CA: Morgan Kaufmann.

Buss, S. R. (Ed.). (1998). *Handbook of Proof Theory*. Amsterdam: Elsevier.

Cassimatis, N. L. (2002). *Polyscheme: A cognitive architecture for integrating multiple representation and inference schemes*. Unpublished doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Cassimatis, N. L., Trafton, J. G., Bugajska, M. D., and Schultz, A. C. (2004). Integrating cognition, perception and action through mental simulation in robots. *Robotics and Autonomous Systems, 49*(1–2), 13–23.

Castilho, M. A., Gasquet, O., and Herzig, A. (1999). Formalizing action and change in modal logic I: The frame problem. *Journal of Logic and Computation, 9*(5), 701–735.

Charniak, E. (1972). *Toward a model of children's story comprehension* (Tech. Rep. No. AITR-266). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Charniak, E., and McDermott, D. V. (1985). *Introduction to artificial intelligence*. Reading, MA: Addison-Wesley.

Charniak, E., Riesbeck, C. K., and McDermott, D. V. (1980). *Artificial intelligence programming*. Hillsdale, NJ: Lawrence Erlbaum.

Church, A. (1956). *Introduction to mathematical logic* (Vol. I). Princeton, NJ: Princeton University Press.

Church, A. (2004). A note on the Entscheidungsproblem. In M. Davis (Ed.), *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions* (Corrected ed., pp. 108–115). Mineola, NY: Dover. (Original work published 1936.)

Cicekli, N. K., and Yildirim, Y. (2000). Formalizing workflows using the event calculus. In M. T. Ibrahim, J. Küng, and N. Revell (Eds.), *Database and expert systems applications* (Lecture Notes in Computer Science, Vol. 1873, pp. 222–231). Berlin: Springer.

Clark, K. L. (1978). Negation as failure. In H. Gallaire and J. Minker (Eds.), *Logic and data bases* (pp. 293–322). New York: Plenum.

Clark, P., and Porter, B. (2000). *KM—The Knowledge Machine 1.4.0: KM's situation mechanism*. Austin, TX: Department of Computer Science, University of Texas at Austin.

Clark, P., and Porter, B. (2004). *KM—The Knowledge Machine 2.0: Users [sic] manual.* Austin, TX: Department of Computer Science, University of Texas at Austin.

Cohen, P. R., and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence, 42,* 213–261.

Cohn, A. G., Bennett, B., Gooday, J., and Gotts, N. M. (1997). Representing and reasoning with qualitative spatial relations about regions. In O. Stock (Ed.), *Spatial and temporal reasoning* (pp. 97–134). Dordrecht: Kluwer.

Collins, A. M., and Quillian, M. R. (1969). Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior, 8,* 240–247.

Collins, M. (1999). *Head-driven statistical models for natural language parsing.* Unpublished doctoral dissertation, University of Pennsylvania, Philadelphia.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics, 29*(4), 589–637.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (pp. 151–158). New York: ACM.

Crawford, J. M., and Etherington, D. W. (1992). Formalizing reasoning about change: A qualitative reasoning approach. In W. R. Swartout (Ed.), *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 577–583). Menlo Park, CA: AAAI Press.

Crevier, D. (1993). *AI: The tumultuous history of the search for artificial intelligence.* New York: BasicBooks.

Curry, H. B. (1963). *Foundations of mathematical logic.* New York: McGraw-Hill.

Cycorp. (2002a). *Introduction to Cyc inferencing.* Austin, TX: Cycorp.

Cycorp. (2002b). *Negation in Cyc.* Austin, TX: Cycorp.

Cycorp. (2002c). *Ontological engineer's handbook version 0.7.* Austin, TX: Cycorp.

Cycorp. (2002d). *Resolution in Cyc.* Austin, TX: Cycorp.

Cycorp. (2002e). *The OpenCyc planner version 0.7.* Austin, TX: Cycorp.

Cycorp. (2002f). *OpenCyc release 0.7.0 [Computer software].* Austin, TX: Cycorp, `http://www.opencyc.org/`

Cycorp. (2002g). *OpenCyc selected vocabulary and upper ontology.* Austin, TX: Cycorp.

Dahlgren, K., McDowell, J., and Stabler, E. P., Jr. (1989). Knowledge representation for commonsense reasoning with text. *Computational Linguistics, 15*(3), 149–170.

Davidson, D. (2001). The logical form of action sentences. In D. Davidson (Ed.), *Essays on actions and events* (2nd ed., pp. 105–148). Oxford: Oxford University Press. (Original work published 1967.)

Davis, E. (1990). *Representations of commonsense knowledge.* San Mateo, CA: Morgan Kaufmann.

Davis, E. (1992). Axiomatizing qualitative process theory. In B. Nebel, C. Rich, and W. R. Swartout (Eds.), *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (pp. 177–188). San Mateo, CA: Morgan Kaufmann.

Davis, E. (1995). *A highly expressive language of spatial constraints* (Tech. Rep. No. TR1995-714). New York: Computer Science Department, New York University.

Davis, E. (1999, March). Guide to axiomatizing domains in first-order logic. *Electronic News Journal on Reasoning about Actions and Change, 3*(1), 25–28.

Davis, M. (1959). Lecture notes on mathematical logic. Unpublished student notes by Allen Weingarten, fall 1959, New York University.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM, 5*(7), 394–397.

Davis, M., and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM, 7*(3), 201–215.

de Kleer, J. (1975). *Qualitative and quantitative knowledge in classical mechanics* (Tech. Rep. No. AITR-352). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

de Kleer, J., and Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence, 24*(1–3), 7–83.

de Nivelle, H. (1999). *Bliksem 1.10 user manual.* Saarbrücken, Germany: Max-Planck Institut für Informatik.

Dean, T., and Boddy, M. (1988). Reasoning about partially ordered events. *Artificial Intelligence, 36*(3), 375–399.

Dechter, R. (2003). *Constraint processing.* San Francisco: Morgan Kaufmann.

Denecker, M., Dupré, D. T., and Belleghem, K. V. (1998). An inductive definition approach to ramifications. *Linköping Electronic Articles in Computer and Information Science, 3*(007).

Denecker, M., Missiaen, L., and Bruynooghe, M. (1992). Temporal reasoning with abductive event calculus. In B. Neumann (Ed.), *Proceedings of the Tenth European Conference on Artificial Intelligence* (pp. 384–388). Chichester, UK: John Wiley.

DIMACS. (1993). *Satisfiability suggested format* (Tech. Rep.). Piscataway, NJ: Center for Discrete Mathematics and Theoretical Computer Science.

Doherty, P. (1994). Reasoning about action and change using occlusion. In A. G. Cohn (Ed.), *Proceedings of the Eleventh European Conference on Artificial Intelligence* (pp. 401–405). Chichester, UK: John Wiley.

Doherty, P. (1996). *PMON$^+$: A fluent logic for action and change. Formal specification, version 1.0* (Tech. Rep. No. LiTH-IDA-R-96-33). Linköping, Sweden: Department of Computer and Information Science, Linköping University.

Doherty, P., Gustafsson, J., Karlsson, L., and Kvarnström, J. (1998). TAL: Temporal Action Logics language specification and tutorial. *Linköping Electronic Articles in Computer and Information Science, 3*(015).

Doherty, P., and Kvarnström, J. (1997). Tackling the qualification problem using fluent dependency constraints: Preliminary report. *Linköping Electronic Articles in Computer and Information Science, 2*(016).

Doherty, P., and Łukaszewicz, W. (1994). Circumscribing features and fluents. In D. M. Gabbay and H. J. Ohlbach (Eds.), *Temporal logic* (Lecture Notes in Computer Science, Vol. 827, pp. 82–100). Berlin: Springer.

Doherty, P., Łukaszewicz, W., and Szałas, A. (1997). Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning, 18*(3), 297–336.

Domeshek, E., Jones, E., and Ram, A. (1999). Capturing the contents of complex narratives. In A. Ram and K. Moorman (Eds.), *Understanding language understanding: Computational models of reading* (pp. 73–105). Cambridge, MA: MIT Press.

Du, D., Gu, J., and Pardalos, P. M. (Eds.). (1997). *Satisfiability problem: Theory and applications* (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 35). Boston, MA: American Mathematical Society.

Dubois, D., Welty, C. A., and Williams, M.-A. (Eds.). (2004). *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning.* Menlo Park, CA: AAAI Press.

Dyer, M. G. (1983). *In-depth understanding: A computer model of integrated processing for narrative comprehension.* Cambridge, MA: MIT Press.

Ebbinghaus, H.-D., Flum, J., and Thomas, W. (1994). *Mathematical logic* (2nd ed.). New York: Springer.

Elgot-Drapkin, J. J., Kraus, S., Miller, M., Nirkhe, M., and Perlis, D. (1999). *Active logics: A unified formal approach to episodic reasoning* (Tech. Rep. No. CS-TR-4072). College Park, MD: Department of Computer Science, University of Maryland.

Elgot-Drapkin, J. J., and Perlis, D. (1990). Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence, 2*(1), 75–98.

Elkan, C. (1996). Reasoning about unknown, counterfactual, and nondeterministic actions in first-order logic. In G. McCalla (Ed.), *Advances in artificial intelligence* (Lecture Notes in Computer Science, Vol. 1081, pp. 54–68). Berlin: Springer.

Enderton, H. B. (2001). *A mathematical introduction to logic* (2nd ed.). San Diego, CA: Academic Press.

Ernst, M., Millstein, T. D., and Weld, D. S. (1997). Automatic SAT-compilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (pp. 1169–1176). San Mateo, CA: Morgan Kaufmann.

Eshghi, K. (1988). Abductive planning with event calculus. In R. A. Kowalski and K. A. Bowen (Eds.), *Logic programming: Proceedings of the Fifth International Conference and Symposium* (Vol. 1, pp. 562–579). Cambridge, MA: MIT Press.

Etherington, D. W. (1988). *Reasoning with incomplete information.* Los Altos, CA: Morgan Kaufmann.

Eves, H. (1990). *Foundations and fundamental concepts of mathematics* (3rd ed.). Mineola, NY: Dover.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about knowledge.* Cambridge, MA: MIT Press.

Fahlman, S. E. (1974). A planning system for robot construction tasks. *Artificial Intelligence, 5*(1), 1–49.

Fahlman, S. E. (1979). *NETL: A system for representing and using real-world knowledge.* Cambridge, MA: MIT Press.

Falkenhainer, B. C. (1988). *Learning from physical analogies: A study in analogy and the explanation process* (Tech. Rep. No. UIUCDCS-R-88-1479). Urbana, IL: University of Illinois at Urbana-Champaign.

Falkenhainer, B. C., Forbus, K. D., and Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence, 41*(1), 1–63.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972a). Learning and executing generalized robot plans. *Artificial Intelligence, 3,* 251–288.

Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972b). Some new directions in robot problem solving. In B. Meltzer and D. Michie (Eds.), *Machine intelligence 7* (pp. 405–430). Edinburgh, UK: Edinburgh University Press.

Fikes, R. E., and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2,* 189–208.

Finger, J. J. (1987). *Exploiting constraints in design synthesis.* Unpublished doctoral dissertation, Stanford University, Stanford, CA.

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence, 24*(1–3), 85–168.

Forbus, K. D. (2001). Exploring analogy in the large. In D. Gentner, K. J. Holyoak, and B. N. Kokinov (Eds.), *The analogical mind: Perspectives from cognitive science* (pp. 23–58). Cambridge, MA: MIT Press.

Forbus, K. D., and Gentner, D. (1997). Qualitative mental models: Simulations or memories? In *Proceedings of the Eleventh International Workshop on Qualitative Reasoning* (pp. 1–8). Cortona, Italy.

Ford, K. M., and Pylyshyn, Z. W. (Eds.). (1996). *The robot's dilemma revisited: The frame problem in artificial intelligence.* Norwood, NJ: Ablex.

Foster, I., and Kesselman, C. (Eds.). (2004). *The grid: Blueprint for a new computing infrastructure* (2nd ed.). San Francisco: Morgan Kaufmann.

Frege, G. (1967). *Begriffsschrift,* a formula language, modeled upon that of arithmetic, for pure thought. In J. van Heijenoort (Ed.), *From Frege to Gödel: A source book in mathematical logic, 1879–1931* (pp. 1–82). Cambridge, MA: Harvard University Press. (Original work published 1879.)

Freud, A. (1946). *The ego and the mechanisms of defence.* New York: International Universities. (Original work published 1936.)

Gabbay, D. M., and Guenthner, F. (Eds.). (2001). *Handbook of philosophical logic* (2nd ed.). Dordrecht: Kluwer.

Gabbay, D. M., Hogger, C. J., and Robinson, J. A. (Eds.). (1994). *Handbook of logic in artificial intelligence and logic programming: Vol. 3. Nonmonotonic reasoning and uncertain reasoning.* Oxford: Oxford University Press.

Gabbay, D. M., and Ohlbach, H. J. (1992). Quantifier elimination in second-order predicate logic. In B. Nebel, C. Rich, and W. R. Swartout (Eds.), *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (pp. 425–435). San Mateo, CA: Morgan Kaufmann.

Garey, M. R., and Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness.* San Francisco: W. H. Freeman.

Garnier, R., and Taylor, J. (1996). *100% mathematical proof.* Chichester, UK: John Wiley.

Geffner, H. (1990). Causal theories for nonmonotonic reasoning. In T. Dietterich and W. R. Swartout (Eds.), *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 524–530). Menlo Park, CA: AAAI Press.

Gelfond, M., and Lifschitz, V. (1993). Representing action and change by logic programs. *Journal of Logic Programming, 17*(2–4), 301–321.

Gelfond, M., Lifschitz, V., and Rabinov, A. (1991). What are the limitations of the situation calculus? In R. S. Boyer (Ed.), *Automated reasoning: Essays in honor of Woody Bledsoe* (pp. 167–179). Dordrecht: Kluwer.

Genesereth, M. R., and Nilsson, N. J. (1987). *Logical foundations of artificial intelligence.* Palo Alto, CA: Morgan Kaufmann.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science, 7*(2), 155–170.

Gentner, D., Holyoak, K. J., and Kokinov, B. N. (Eds.). (2001). *The analogical mind: Perspectives from cognitive science.* Cambridge, MA: MIT Press.

Gentzen, G. (1969). Investigations into logical deduction. In M. E. Szabo (Ed.), *The collected papers of Gerhard Gentzen* (pp. 68–131). Amsterdam: North-Holland. (Original work published 1935.)

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated planning: Theory and practice.* San Francisco: Morgan Kaufmann.

Gildea, D., and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics, 28*(3), 245–288.

Ginsberg, M. L. (Ed.). (1987). *Readings in nonmonotonic reasoning.* Los Altos, CA: Morgan Kaufmann.

Ginsberg, M. L. (1991). [Review of the book by Marvin Minsky, *The Society of Mind*]. *Artificial Intelligence, 48,* 335–339.

Ginsberg, M. L., and Smith, D. E. (1987a). Reasoning about action I: A possible worlds approach. In F. M. Brown (Ed.), *The frame problem in artificial intelligence: Proceedings of the 1987 workshop* (pp. 233–258). Los Altos, CA: Morgan Kaufmann.

Ginsberg, M. L., and Smith, D. E. (1987b). Reasoning about action II: The qualification problem. In F. M. Brown (Ed.), *The frame problem in artificial intelligence: Proceedings of the 1987 workshop* (pp. 259–287). Los Altos, CA: Morgan Kaufmann.

Ginsberg, M. L., and Smith, D. E. (1988a). Reasoning about action I: A possible worlds approach. *Artificial Intelligence, 35*(2), 165–195.

Ginsberg, M. L., and Smith, D. E. (1988b). Reasoning about action II: The qualification problem. *Artificial Intelligence, 35*(3), 311–342.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N. C., and Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence, 153,* 49–104.

Giunchiglia, E., and Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Conference on Innovative Applications of Artificial Intelligence* (pp. 623–630). Menlo Park, CA: AAAI Press.

Giunchiglia, E., and Sebastiani, R. (1999). Applying the Davis-Putnam procedure to non-clausal formulas. In E. Lamma and P. Mello (Eds.), AI*IA 99: Advances in artificial intelligence (Lecture Notes in Computer Science, Vol. 1792, pp. 84–94). Berlin: Springer.

Giunchiglia, F., and Ghidini, C. (1998). Local models semantics, or contextual reasoning = locality + compatibility. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro (Eds.), *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 282–289). San Francisco: Morgan Kaufmann.

Gödel, K. (1967). The completeness of the axioms of the functional calculus of logic. In J. van Heijenoort (Ed.), *From Frege to Gödel: A source book in mathematical logic, 1879–1931* (pp. 582–591). Cambridge, MA: Harvard University Press. (Original work published 1930.)

Gordon, A. S. (2004). *Strategy representation: An analysis of planning knowledge.* Mahwah, NJ: Lawrence Erlbaum.

Green, C. C. (1969). *The application of theorem proving to question-answering systems.* Unpublished doctoral dissertation, Stanford University, Stanford, CA.

Grosof, B. (1984). Default reasoning as circumscription: A translation of default logic into circumscription or maximizing defaults is minimizing predicates. In *Proceedings of the Non-Monotonic Reasoning Workshop* (pp. 115–124). Menlo Park, CA: AAAI Press.

Guha, R. V. (1990). *The representation of defaults in Cyc* (Tech. Rep. No. ACT-CYC-083-90). Austin, TX: Microelectronics and Computer Technology Corporation.

Guha, R. V. (1992). *Contexts: A formalization and some applications.* Unpublished doctoral dissertation, Stanford University, Stanford, CA.

Guha, R. V., and Lenat, D. B. (1990a). CYC: A midterm report. *AI Magazine, 11*(3), 32–59.

Guha, R. V., and Lenat, D. B. (1990b). *CycL: The Cyc representation language, Part 4* (Tech. Rep. No. ACT-CYC-154-90). Austin, TX: Microelectronics and Computer Technology Corporation.

Guicciardini, N. (1999). *Reading the Principia: The debate on Newton's mathematical methods for natural philosophy from 1687 to 1736.* Cambridge, UK: Cambridge University Press.

Gustafsson, J., and Doherty, P. (1996). Embracing occlusion in specifying the indirect effects of actions. In L. C. Aiello, J. Doyle, and S. C. Shapiro (Eds.), *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 87–98). San Francisco: Morgan Kaufmann.

Haas, A. R. (1987). The case for domain-specific frame axioms. In F. M. Brown (Ed.), *The frame problem in artificial intelligence: Proceedings of the 1987 workshop* (pp. 343–348). Los Altos, CA: Morgan Kaufmann.

Halpern, J. Y. (1999, July). On the adequacy of modal logic. *Electronic News Journal on Reasoning about Actions and Change, 3*(3).

Halpern, J. Y. (2000, May). On the adequacy of modal logic, II: A response to McCarthy. *Electronic News Journal on Reasoning about Actions and Change, 4*(1).

Halpern, J. Y. (2003). *Reasoning about uncertainty*. Cambridge, MA: MIT Press.

Halpin, H., Moore, J. D., and Robertson, J. (2004). Automatic analysis of plot for story rewriting. In D. Lin and D. Wu (Eds.), *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing* (pp. 127–133). Barcelona, Spain.

Hamilton, A. G. (1988). *Logic for mathematicians* (Revised ed.). Cambridge, UK: Cambridge University Press.

Hanks, S., and Madigan, D. (2005). Probabilistic temporal reasoning. In M. Fisher, D. Gabbay, and L. Vila (Eds.), *Handbook of temporal reasoning in artificial intelligence* (Vol. 1, pp. 315–342). Amsterdam: Elsevier.

Hanks, S., and McDermott, D. V. (1985). *Temporal reasoning and default logics* (Tech. Rep. No. YALE/DCS/tr430). New Haven, CT: Computer Science Department, Yale University.

Hanks, S., and McDermott, D. V. (1986). Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 328–333). Los Altos, CA: Morgan Kaufmann.

Hanks, S., and McDermott, D. V. (1987). Nonmonotonic logic and temporal projection. *Artificial Intelligence, 33*(3), 379–412.

Harel, D. (1984). Dynamic logic. In D. M. Gabbay and F. Guenthner (Eds.), *Handbook of philosophical logic* (1st ed., Vol. 2, pp. 497–604). Dordrecht: Reidel.

Hayes, P. J. (1971). A logic of actions. In B. Meltzer and D. Michie (Eds.), *Machine intelligence 6* (pp. 495–520). New York: Halsted.

Hayes, P. J. (1973). The frame problem and related problems in artificial intelligence. In A. Elithorn and D. Jones (Eds.), *Artificial and human thinking* (pp. 45–59). Amsterdam: Elsevier.

Hayes, P. J. (1977). In defence of logic. In R. Reddy (Ed.), *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 559–565). Los Altos, CA: William Kaufmann.

Hayes, P. J. (1979). The logic of frames. In D. Metzing (Ed.), *Frame conceptions and text understanding* (pp. 46–61). Berlin: Walter de Gruyter.

Hayes, P. J. (1985). Naive physics I: Ontology for liquids. In J. R. Hobbs and R. C. Moore (Eds.), *Formal theories of the commonsense world* (pp. 71–107). Norwood, NJ: Ablex.

Hearn, R. A. (2001). *Building grounded abstractions for artificial intelligence programming.* Unpublished master's thesis, Massachusetts Institute of Technology, Cambridge, MA.

Hearst, M. A. (1998). Automated discovery of WordNet relations. In C. Fellbaum (Ed.), *WordNet: An electronic lexical database* (pp. 131–151). Cambridge, MA: MIT Press.

Heider, F. (1958). *The psychology of interpersonal relations.* Hillsdale, NJ: Lawrence Erlbaum.

Hendrix, G. G. (1973). Modeling simultaneous actions and continuous processes. *Artificial Intelligence, 4,* 145–180.

Henkin, L. (1949). The completeness of the first-order functional calculus. *Journal of Symbolic Logic, 14*(3), 159–166.

Henle, M. (1962). On the relation between logic and thinking. *Psychological Review, 69*(4), 366–378.

Herrmann, C. S., and Thielscher, M. (1996). Reasoning about continuous processes. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Annual Conference on Innovative Applications of Artificial Intelligence* (pp. 639–644). Menlo Park, CA: AAAI Press.

Hewitt, C. (1972). *Description and theoretical analysis (using schemata) of planner: A language for proving theorems and manipulating models in a robot* (Tech. Rep. No. AITR-258). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Hewitt, C. (1987). Metacritique of McDermott and the logicist approach. *Computational Intelligence, 3,* 185–189.

Hobbs, J. R., and Moore, R. C. (Eds.). (1985). *Formal theories of the commonsense world.* Norwood, NJ: Ablex.

Hobbs, J. R., Stickel, M. E., Appelt, D. E., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence, 63,* 69–142.

Hodges, W. (1993). *Model theory.* Cambridge, UK: Cambridge University Press.

Hodges, W. (2001). Elementary predicate logic. In D. M. Gabbay and F. Guenthner (Eds.), *Handbook of philosophical logic* (2nd ed., Vol. 1, pp. 1–129). Dordrecht: Kluwer.

Hölldobler, S., and Schneeberger, J. (1990). A new deductive approach to planning. *New Generation Computing, 8*(3), 225–244.

Hölldobler, S., and Thielscher, M. (1995). Computing change and specificity with equational logic programs. *Annals of Mathematics and Artificial Intelligence, 14*(1), 99–133.

Holyoak, K. J., and Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science, 13*(3), 295–355.

Hoos, H. H., and Stützle, T. (2005). *Stochastic local search: Foundations and applications.* San Francisco: Morgan Kaufmann.

Horty, J. F., Thomason, R. H., and Touretzky, D. S. (1990). A skeptical theory of inheritance in nonmonotonic semantic networks. *Artificial Intelligence, 42*(2–3), 311–348.

Israel, D. J. (1980). What's wrong with non-monotonic logic? In R. M. Balzer (Ed.), *Proceedings of the First National Conference on Artificial Intelligence* (pp. 99–101). Menlo Park, CA: AAAI Press.

Israel, D. J. (1985). A short companion to the naive physics manifesto. In J. R. Hobbs and R. C. Moore (Eds.), *Formal theories of the commonsense world* (pp. 427–447). Norwood, NJ: Ablex.

Iwańska, L. M., and Shapiro, S. C. (Eds.). (2000). *Natural language processing and knowledge representation.* Cambridge, MA: MIT Press.

Jackson, D. (2000). Automating first-order relational logic. In *Proceedings of the Eighth International Symposium on the Foundations of Software Engineering* (pp. 130–139). New York: ACM.

Johnson-Laird, P. N. (1983). *Mental models: Toward a cognitive science of language, inference, and consciousness.* Cambridge, MA: Harvard University Press.

Johnson-Laird, P. N. (1993). *Human and machine thinking.* Hillsdale, NJ: Lawrence Erlbaum.

Jordan, P. W. (2004). Using student explanations as models for adapting tutorial dialogue. In V. Barr and Z. Markov (Eds.), *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference* (pp. 905–910). Menlo Park, CA: AAAI Press.

Jurafsky, D., and Martin, J. H. (2000). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.* Upper Saddle River, NJ: Prentice Hall.

Kakas, A. C., and Miller, R. (1997a). Reasoning about actions, narratives and ramifications. *Linköping Electronic Articles in Computer and Information Science, 2*(012).

Kakas, A. C., and Miller, R. (1997b). A simple declarative language for describing narratives with actions. *Journal of Logic Programming, 31*(1–3), 157–200.

Karlsson, L., and Gustafsson, J. (1997). Reasoning about actions in a multi-agent environment. *Linköping Electronic Articles in Computer and Information Science, 2*(014).

Karlsson, L., Gustafsson, J., and Doherty, P. (1998). Delayed effects of actions. In H. Prade (Ed.), *Proceedings of the Thirteenth European Conference on Artificial Intelligence* (pp. 542–546). Chichester, UK: John Wiley.

Kartha, G. N. (1994). Two counterexamples related to Baker's approach to the frame problem. *Artificial Intelligence, 69*(1–2), 379–391.

Kartha, G. N., and Lifschitz, V. (1994). Actions with indirect effects (preliminary report). In J. Doyle, E. Sandewall, and P. Torasso (Eds.), *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 341–350). San Francisco: Morgan Kaufmann.

Kartha, G. N., and Lifschitz, V. (1995). A simple formalization of actions using circumscription. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1970–1977). San Mateo, CA: Morgan Kaufmann.

Kautz, H. A. (1986). The logic of persistence. In *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 401–405). Los Altos, CA: Morgan Kaufmann.

Kautz, H. A., and Selman, B. (1992). Planning as satisfiability. In B. Neumann (Ed.), *Proceedings of the Tenth European Conference on Artificial Intelligence* (pp. 359–363). Chichester, UK: John Wiley.

Kautz, H. A., and Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Annual Conference*

on Innovative Applications of Artificial Intelligence (pp. 1194–1201). Menlo Park, CA: AAAI Press.

Kautz, H. A., and Selman, B. (2003). Ten challenges redux: Recent progress in propositional reasoning and search. In F. Rossi (Ed.), *Principles and practice of constraint programming* (Lecture Notes in Computer Science, Vol. 2833, pp. 1–18). Berlin: Springer.

Kleene, S. C. (1952). *Introduction to metamathematics.* New York: D. Van Nostrand.

Kleene, S. C. (1967). *Mathematical logic.* New York: John Wiley.

Kneale, W., and Kneale, M. (1962). *The development of logic.* Oxford: Oxford University Press.

Koenderink, J. J., and Doorn, A. J. van. (1979). The internal representation of solid shape with respect to vision. *Biological Cybernetics, 32,* 211–216.

Kolaitis, P. G., and Papadimitriou, C. H. (1990). Some computational aspects of circumscription. *Journal of the ACM, 37*(1), 1–14.

Kolata, G. (1982). How can computers get common sense? *Science, 217*(4566), 1237–1238.

Kolodner, J. (1993). *Case-based reasoning.* San Mateo, CA: Morgan Kaufmann.

Kowalski, R. A. (1974). *Logic for problem solving* (Memo No. 75). Edinburgh, Scotland: Department of Computational Logic, University of Edinburgh.

Kowalski, R. A. (1979). *Logic for problem solving.* New York: North-Holland.

Kowalski, R. A. (1986). *Database updates in the event calculus* (Tech. Rep. No. DOC 86/12). London: Imperial College of Science, Technology, and Medicine.

Kowalski, R. A. (1992). Database updates in the event calculus. *Journal of Logic Programming, 12,* 121–146.

Kowalski, R. A., and Sadri, F. (1994). The situation calculus and event calculus compared. In M. Bruynooghe (Ed.), *Logic programming: The 1994 international symposium* (pp. 539–553). Cambridge, MA: MIT Press.

Kowalski, R. A., and Sadri, F. (1997). Reconciling the event calculus with the situation calculus. *Journal of Logic Programming, 31*(1–3), 39–58.

Kowalski, R. A., and Sergot, M. J. (1986). A logic-based calculus of events. *New Generation Computing, 4*(1), 67–95.

Kuipers, B. (1986). Qualitative simulation. *Artificial Intelligence, 29*(3), 289–338.

Kuipers, B. (1994). *Qualitative reasoning: Modeling and simulation with incomplete knowledge.* Cambridge, MA: MIT Press.

Kuipers, B. (2000). The Spatial Semantic Hierarchy. *Artificial Intelligence, 119,* 191–233.

Kushmerick, N., Hanks, S., and Weld, D. S. (1995). An algorithm for probabilistic planning. *Artificial Intelligence, 76,* 239–286.

Kvarnström, J. (2001). *VITAL: Visualization and implementation of temporal action logic.* Linköping, Sweden: Department of Computer and Information Science, Linköping University.

Kvarnström, J. (2005). *TALplanner and other extensions to temporal action logic.* Unpublished doctoral dissertation, Linköping University, Linköping, Sweden.

Kvarnström, J., and Doherty, P. (2000a). Tackling the qualification problem using fluent dependency constraints. *Computational Intelligence, 16*(2), 169–209.

Kvarnström, J., and Doherty, P. (2000b). TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence, 30*(1–4), 119–169.

Le Berre, D., and Simon, L. (2004). The essentials of the SAT 2003 competition. In E. Giunchiglia and A. Tacchella (Eds.), *Theory and applications of satisfiability testing* (Lecture Notes in Computer Science, Vol. 2919, pp. 452–467). Berlin: Springer.

Leake, D. B. (1995). Combining rules and cases to learn case adaptation. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society* (pp. 84–89). Hillsdale, NJ: Lawrence Erlbaum.

Leake, D. B. (1996). CBR in context: The present and future. In D. B. Leake (Ed.), *Case-based reasoning: Experiences, lessons, & future directions* (pp. 3–30). Cambridge, MA: MIT Press.

Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM, 38*(11), 33–48.

Lenat, D. B., and Guha, R. V. (1990). *Building large knowledge-based systems: Representation and inference in the Cyc project.* Reading, MA: Addison-Wesley.

Lenat, D. B., and Guha, R. V. (1991). *Ideas for applying Cyc* (Tech. Rep. No. ACT-CYC-407-91). Austin, TX: Microelectronics and Computer Technology Corporation.

Levesque, H. J. (1981). *A formal treatment of incomplete knowledge bases.* Unpublished doctoral dissertation, University of Toronto, Toronto.

Levin, L. A. (1973). Universal search problems [in Russian]. *Problemy Peredachi Informatsii [Problems of Information Transmission], 9*(3), 265–266.

Lévy, F., and Quantz, J. J. (1998). Representing beliefs in a situated event calculus. In H. Prade (Ed.), *Proceedings of the Thirteenth European Conference on Artificial Intelligence* (pp. 547–551). Chichester, UK: John Wiley.

Lewin, K. (1951). Intention, will and need. In D. Rapaport (Ed.), *Organization and pathology of thought* (pp. 95–153). New York: Columbia University Press. (Original work published 1926.)

Lieberman, H., Liu, H., Singh, P., and Barry, B. A. (2004). Beating common sense into interactive applications. *AI Magazine, 25*(4), 63–76.

Lifschitz, V. (1985). Computing circumscription. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 121–127). Los Altos, CA: Morgan Kaufmann.

Lifschitz, V. (1987a). Formal theories of action. In F. M. Brown (Ed.), *The frame problem in artificial intelligence: Proceedings of the 1987 workshop* (pp. 35–57). Los Altos, CA: Morgan Kaufmann.

Lifschitz, V. (1987b). On the semantics of STRIPS. In M. P. Georgeff and A. L. Lansky (Eds.), *Reasoning about actions and plans: Proceedings of the 1986 workshop* (pp. 1–9). San Mateo, CA: Morgan Kaufmann.

Lifschitz, V. (1987c). Pointwise circumscription. In M. L. Ginsberg (Ed.), *Readings in nonmonotonic reasoning* (pp. 179–193). Los Altos, CA: Morgan Kaufmann.

Lifschitz, V. (1989). Benchmark problems for formal nonmonotonic reasoning: Version 2.00. In M. Reinfrank, J. de Kleer, M. L. Ginsberg, and E. Sandewall (Eds.), *Non-Monotonic reasoning: Proceedings of the Second international workshop* (Lecture Notes in Computer Science, Vol. 346, pp. 202–219). Berlin: Springer.

Lifschitz, V. (Ed.). (1990a). *Formalizing common sense: Papers by John McCarthy*. Norwood, NJ: Ablex.

Lifschitz, V. (1990b). Frames in the space of situations. *Artificial Intelligence, 46*(3), 365–376.

Lifschitz, V. (1994). Circumscription. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson (Eds.), *Handbook of logic in artificial intelligence and logic programming: Vol. 3. Nonmonotonic reasoning and uncertain reasoning* (pp. 298–352). Oxford: Oxford University Press.

Lifschitz, V. (1997a, November 19). [Responses to Costello and Doherty]. *Electronic Newsletter on Reasoning about Actions and Change*.

Lifschitz, V. (1997b, November 27). [Responses to Sandewall, Doherty, Costello, and Hayes]. *Electronic Newsletter on Reasoning about Actions and Change*.

Lifschitz, V. (1999). Success of default logic. In H. J. Levesque and F. Pirri (Eds.), *Logical foundations for cognitive agents: Contributions in honor of Ray Reiter* (pp. 208–212). Berlin: Springer.

Lifschitz, V. (2003, May). *Choosing a place for dinner.* Message posted to Texas Action Group electronic mailing list, archived at `http://www.cs.utexas.edu/users/vl/tag/discussions.html`

Lin, F. (1995). Embracing causality in specifying the indirect effects of actions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1985–1993). San Mateo, CA: Morgan Kaufmann.

Lin, F. (1996). Embracing causality in specifying the indeterminate effects of actions. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Annual Conference on Innovative Applications of Artificial Intelligence* (pp. 670–676). Menlo Park, CA: AAAI Press.

Lin, F., and Reiter, R. (1994). State constraints revisited. *Journal of Logic and Computation, 4*(5), 655–678.

Lin, F., and Shoham, Y. (1992). Concurrent actions in the situation calculus. In W. R. Swartout (Ed.), *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 590–595). Menlo Park, CA: AAAI Press.

Lin, F., and Shoham, Y. (1995). Provably correct theories of action. *Journal of the ACM, 42*(2), 293–320.

Liu, H., and Singh, P. (2004a). Commonsense reasoning in and over natural language. In M. G. Negoita, R. J. Howlett, and L. C. Jain (Eds.), *Knowledge-based intelligent information and engineering systems* (Lecture Notes in Computer Science, Vol. 3215, pp. 293–306). Berlin: Springer.

Liu, H., and Singh, P. (2004b). ConceptNet—A practical commonsense reasoning tool-kit. *BT Technology Journal, 22*(4), 211–226.

Ma, J., and Knight, B. (2001). Reified temporal logics: An overview. *Artificial Intelligence Review, 15*(3), 189–217.

Manna, Z., and Waldinger, R. (1985). *The logical basis for computer programming: Vol. 1. Deductive reasoning.* Reading, MA: Addison-Wesley.

Manning, C. D., and Schütze, H. (1999). *Foundations of statistical natural language processing.* Cambridge, MA: MIT Press.

Marsden, J. E. (1974). *Elementary classical analysis.* San Francisco: W. H. Freeman.

Mates, B. (1972). *Elementary logic* (2nd ed.). New York: Oxford University Press.

Mateus, P., Pacheco, A., and Pinto, J. A. (2002). Observations and the probabilistic situation calculus. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams (Eds.), *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 327–338). San Francisco: Morgan Kaufmann.

McCain, N. C. (1997). *Causality in commonsense reasoning about actions* (Tech. Rep. No. CS-TR-97-25). Austin, TX: Department of Computer Sciences, University of Texas at Austin.

McCain, N. C., and Turner, H. (1995). A causal theory of ramifications and qualifications. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1978–1984). San Mateo, CA: Morgan Kaufmann.

McCain, N. C., and Turner, H. (1997). Causal theories of action and change. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference* (pp. 460–465). Menlo Park, CA: AAAI Press.

McCarthy, J. (1959). Programs with common sense. In *Mechanisation of Thought Processes: Proceedings of a symposium held at the National Physical Laboratory on 24th, 25th, 26th and 27th November 1958* (Vol. 1, pp. 75–91). London: Her Majesty's Stationery Office.

McCarthy, J. (1963). *Situations, actions and causal laws* (Memo 2). Stanford, CA: Stanford Artificial Intelligence Project, Stanford University.

McCarthy, J. (1968). Programs with common sense. In M. Minsky (Ed.), *Semantic information processing* (pp. 403–418). Cambridge, MA: MIT Press.

McCarthy, J. (1977). Epistemological problems of artificial intelligence. In R. Reddy (Ed.), *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 1038–1044). Los Altos, CA: William Kaufmann.

McCarthy, J. (1979). First order theories of individual concepts and propositions. In J. E. Hayes, D. Michie, and L. I. Mikulich (Eds.), *Machine intelligence 9* (pp. 129–148). Chichester, UK: Ellis Horwood.

McCarthy, J. (1980). Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence, 13*(1–2), 27–39.

McCarthy, J. (1984a). Applications of circumscription to formalizing common sense knowledge. In *Proceedings of the Non-Monotonic Reasoning Workshop* (pp. 295–324). Menlo Park, CA: AAAI Press.

McCarthy, J. (1984b). Some expert systems need common sense. In H. R. Pagels (Ed.), *Computer culture: The scientific, intellectual, and social impact of the computer* (pp. 129–137). New York: New York Academy of Sciences.

McCarthy, J. (1986). Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence, 28*, 89–116.

McCarthy, J. (1987). Generality in artificial intelligence. *Communications of the ACM, 30*(12), 1030–1035.

McCarthy, J. (1988). Mathematical logic in artificial intelligence. *Dædalus, 117*(1), 297–311.

McCarthy, J. (1990). An example for natural language understanding and the AI problems it raises. In V. Lifschitz (Ed.), *Formalizing common sense: Papers by John McCarthy* (pp. 70–76). Norwood, NJ: Ablex.

McCarthy, J. (1993). Notes on formalizing context. In R. Bajcsy (Ed.), *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 555–562). San Mateo, CA: Morgan Kaufmann.

McCarthy, J. (1997). Modality, si! Modal logic, no! *Studia Logica, 59*(1), 29–32.

McCarthy, J. (1998a). Elaboration tolerance. In R. Miller and M. Shanahan (Eds.), *Proceedings of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning.* Queen Mary and Westfield College: London.

McCarthy, J. (1998b). Making robots conscious of their mental states. In K. Furukawa, D. Michie, and S. Muggleton (Eds.), *Machine intelligence 15* (pp. 3–17). Oxford: Oxford University Press.

McCarthy, J. (1999, September). Modality for robots—Responses to Halpern and Wansing. *Electronic News Journal on Reasoning about Actions and Change, 3*(5).

McCarthy, J. (2000, July 22). Modality for robots II—Response to Halpern. *Electronic Newsletter on Reasoning about Actions and Change.*

McCarthy, J., and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie (Eds.), *Machine intelligence 4* (pp. 463–502). Edinburgh, Scotland: Edinburgh University Press.

McCarthy, J., and Lifschitz, V. (1987). Commentary on McDermott. *Computational Intelligence, 3*, 196–197.

McCarthy, J., Minsky, M., Sloman, A., Gong, L., Lau, T., Morgenstern, L., Mueller, E. T., Riecken, D., Singh, M., and Singh, P. (2002). An architecture of diversity for commonsense reasoning. *IBM Systems Journal, 41*(3), 530–539.

McClelland, J. L., Rumelhart, D. E., and PDP Research Group (Eds.). (1986). *Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 2. Psychological and biological models.* Cambridge, MA: MIT Press.

McCorduck, P. (2004). *Machines who think: A personal inquiry into the history and prospects of artificial intelligence.* Natick, MA: A K Peters.

McDermott, D. V. (1982). A temporal logic for reasoning about processes and plans. *Cognitive Science, 6*, 101–155.

McDermott, D. V. (1987). A critique of pure reason. *Computational Intelligence, 3*, 151–160.

McDermott, D. V., and Doyle, J. (1980). Non-monotonic logic I. *Artificial Intelligence, 13*(1–2), 41–72.

McIlraith, S., Peppas, P., and Thielscher, M. (Eds.). (2005). *Proceedings of the Seventh International Symposium on Logical Formalizations of Commonsense Reasoning.* Dresden, Germany: Technischen Universität Dresden.

McMillan, K. L., and Amla, N. (2003). Automatic abstraction without counterexamples. In H. Garavel and J. Hatcliff (Eds.), *Tools and algorithms for the construction and analysis of systems: Tacas 2003* (Lecture Notes in Computer Science, Vol. 2619, pp. 2–17). Berlin: Springer.

Mendelson, E. (1997). *Introduction to mathematical logic* (4th ed.). Boca Raton, FL: Chapman and Hall.

Miller, G. A., Galanter, E., and Pribram, K. H. (1960). *Plans and the structure of behavior.* New York: Holt, Rinehart and Winston.

Miller, R. (1996). A case study in reasoning about actions and continuous change. In W. Wahlster (Ed.), *Proceedings of the Twelfth European Conference on Artificial Intelligence* (pp. 624–628). Chichester, UK: John Wiley.

Miller, R., and Shanahan, M. (1994). Narratives in the situation calculus. *Journal of Logic and Computation, 4*(5), 513–530.

Miller, R., and Shanahan, M. (1996). Reasoning about discontinuities in the event calculus. In L. C. Aiello, J. Doyle, and S. C. Shapiro (Eds.), *Proceedings of the fifth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 63–74). San Francisco: Morgan Kaufmann.

Miller, R., and Shanahan, M. (1999). The event calculus in classical logic—alternative axiomatisations. *Linköping Electronic Articles in Computer and Information Science, 4*(016).

Miller, R., and Shanahan, M. (2002). Some alternative formulations of the event calculus. In A. C. Kakas and F. Sadri (Eds.), *Computational logic: Logic programming and beyond: Essays in honour of Robert A. Kowalski, part II* (Lecture Notes in Computer Science, Vol. 2408, pp. 452–490). Berlin: Springer.

Minker, J. (1993). An overview of nonmonotonic reasoning and logic programming. *Journal of Logic Programming, 17*(2–4), 95–126.

Minsky, M. (1961). Descriptive languages and problem solving. In *Proceedings of the Western Joint Computer Conference* (pp. 215–218). New York: National Joint Computer Committee.

Minsky, M. (1974). *A framework for representing knowledge* (A. I. Memo 306). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Minsky, M. (1977). Plain talk about neurodevelopmental epistemology. In R. Reddy (Ed.), *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 1083–1092). Los Altos, CA: William Kaufmann.

Minsky, M. (1980). K-lines: A theory of memory. *Cognitive Science, 4,* 117–133.

Minsky, M. (1982). Learning meaning (draft). Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

Minsky, M. (1986). *The society of mind.* New York: Simon & Schuster.

Minsky, M. (1991a). Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine, 12*(2), 34–51.

Minsky, M. (1991b). Society of Mind: A response to four reviews. *Artificial Intelligence, 48,* 371–396.

Minsky, M. (1994). Negative expertise. *International Journal of Expert Systems, 7*(1), 13–19.

Minsky, M. (in press). *The emotion machine.* New York: Pantheon.

Minsky, M., and Papert, S. (1972). *Artificial intelligence progress report* (A. I. Memo 252). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Minsky, M., Singh, P., and Sloman, A. (2004). The St. Thomas common sense symposium: Designing architectures for human-level intelligence. *AI Magazine, 25*(2), 113–124.

Mitchell, D. G., Selman, B., and Levesque, H. J. (1992). Hard and easy distributions of SAT problems. In W. R. Swartout (Ed.), *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 459–465). Menlo Park, CA: AAAI Press.

Moore, J. D. (1995). *Participating in explanatory dialogues: Interpreting and responding to questions in context.* Cambridge, MA: MIT Press.

Moore, R. C. (1980). *Reasoning about knowledge and action* (Tech. Note 191). Menlo Park, CA: SRI International.

Moore, R. C. (1982). The role of logic in knowledge representation and commonsense reasoning. In D. L. Waltz (Ed.), *Proceedings of the National Conference on Artificial Intelligence* (pp. 428–433). Menlo Park, CA: AAAI Press.

Moore, R. C. (1985). Semantical considerations on nonmonotonic logic. *Artificial Intelligence, 25*(1), 75–94.

Moore, R. C. (1995). *Logic and representation.* Stanford, CA: CSLI.

Morgenstern, L. (1988). *Foundations of a logic of knowledge, action, and communication.* Unpublished doctoral dissertation, New York University, New York.

Morgenstern, L. (2001). Mid-sized axiomatizations of commonsense problems: A case study in egg cracking. *Studia Logica, 67,* 333–384.

Morgenstern, L., and Miller, R. (2004). *Common sense problem page.* From Stanford University Web site: `http://www-formal.stanford.edu/leora/commonsense/`

Mueller, E. T. (1990). *Daydreaming in humans and machines: A computer model of the stream of thought.* Norwood, NJ: Ablex.

Mueller, E. T. (1998). *Natural language processing with ThoughtTreasure.* New York: Signiform.

Mueller, E. T. (2002). Story understanding. In L. Nadel (Ed.), *Encyclopedia of cognitive science* (Vol. 4, pp. 238–246). London: Nature Publishing Group.

Mueller, E. T. (2003). Story understanding through multi-representation model construction. In G. Hirst and S. Nirenburg (Eds.), *Text Meaning: Proceedings of the HLT-NAACL 2003 workshop* (pp. 46–53). East Stroudsburg, PA: Association for Computational Linguistics.

Mueller, E. T. (2004a). Event calculus reasoning through satisfiability. *Journal of Logic and Computation, 14*(5), 703–730.

Mueller, E. T. (2004b). A tool for satisfiability-based commonsense reasoning in the event calculus. In V. Barr and Z. Markov (Eds.), *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference* (pp. 147–152). Menlo Park, CA: AAAI Press.

Mueller, E. T. (2004c). Understanding script-based stories using commonsense reasoning. *Cognitive Systems Research, 5*(4), 307–340.

Mueller, E. T., and Sutcliffe, G. (2005). Discrete event calculus deduction using first-order automated theorem proving. In B. Konev and S. Schulz (Eds.), *Proceedings of the Fifth International Workshop on the Implementation of Logics* (pp. 43–56). Liverpool, UK: Department of Computer Science, University of Liverpool.

Mueller, E. T., and Sutcliffe, G. (2005). Reasoning in the event calculus using first-order automated theorem proving. In I. Russel and Z. Markov (Eds.), *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference* (pp. 840–841). Menlo Park, CA: AAAI Press.

Muggleton, S., and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19–20*, 629–679.

Narayanan, S. S. (1997). *Knowledge-based action representations for metaphor and aspect (KARMA)*. Unpublished doctoral dissertation, University of California, Berkeley, CA.

Nerode, A., and Shore, R. A. (1997). *Logic for applications* (2nd ed.). New York: Springer.

Newell, A., and Simon, H. A. (1961). GPS, a program that simulates human thought. In H. Billing (Ed.), *Lernende automaten* (pp. 109–124). Munich: Oldenbourgh.

Newell, A., and Simon, H. A. (1972). *Human problem-solving*. Englewood Cliffs, NJ: Prentice-Hall.

Newton, I. (1969). De methodis serierum et fluxionum. In D. T. Whiteside (Ed.), *The mathematical papers of Isaac Newton* (Vol. III: 1670–1673, pp. 32–353). Cambridge, UK: Cambridge University Press. (Original work published 1670.)

Nilsson, N. J. (1980). *Principles of artificial intelligence*. Los Altos, CA: Morgan Kaufmann.

Nilsson, N. J. (1991). Logic and artificial intelligence. *Artificial Intelligence, 47*(1–3), 31–56.

Nilsson, N. J. (1998). *Artificial intelligence: A new synthesis*. San Francisco: Morgan Kaufmann.

Norvig, P. (1989). Marker passing as a weak method for text inferencing. *Cognitive Science, 13*(4), 569–620.

Oberschelp, A. (1990). Order sorted predicate logic. In K. H. Bläsius, U. Hedtstück, and C.-R. Rollinger (Eds.), *Sorts and types in artificial intelligence* (Lecture Notes in Computer Science, Vol. 418, pp. 8–17). Berlin: Springer.

O'Rorke, P., and Ortony, A. (1994). Explaining emotions. *Cognitive Science, 18*, 283–323.

Ortony, A., Clore, G. L., and Collins, A. M. (1988). *The cognitive structure of emotions*. Cambridge, UK: Cambridge University Press.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics, 31*(1), 71–105.

Parmar, A. (2001). *The representation of actions in KM and Cyc* (Tech. Rep. No. FRG-1). Stanford, CA: Department of Computer Science, Stanford University.

Parmar, A. (2003). *Formalizing elaboration tolerance*. Unpublished doctoral dissertation, Stanford University, Stanford, CA.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. Reading, MA: Addison-Wesley.

Pearl, J. (1988a). On logic and probability. *Computational Intelligence, 4*, 99–103.

Pearl, J. (1988b). *Probabilistic reasoning in intelligent systems: Networks of plausible inference* (Revised ed.). San Francisco: Morgan Kaufmann.

Pearl, J. (2000). *Causality: Models, reasoning, and inference*. Cambridge, UK: Cambridge University Press.

Pednault, E. P. D. (1987). Formulating multiagent, dynamic-world problems in the classical planning framework. In M. P. Georgeff and A. L. Lansky (Eds.), *Reasoning about actions and plans: Proceedings of the 1986 workshop* (pp. 47–82). San Mateo, CA: Morgan Kaufmann.

Pednault, E. P. D. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. J. Brachman, H. J. Levesque, and R. Reiter (Eds.), *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (pp. 324–332). San Mateo, CA: Morgan Kaufmann.

Pinto, J. A. (1994). *Temporal reasoning in the situation calculus*. Unpublished doctoral dissertation, University of Toronto, Toronto.

Pinto, J. A. (1998a). Causality, indirect effects and triggers. In H. Geffner and V. Lifschitz (Eds.), *Proceedings of the Workshop on Action and Causality*. Trento, Italy.

Pinto, J. A. (1998b). Occurrences and narratives as constraints in the branching structure of the situation calculus. *Journal of Logic and Computation, 8*, 777–808.

Pinto, J. A., and Reiter, R. (1993). Temporal reasoning in logic programming: A case for the situation calculus. In D. S. Warren (Ed.), *Logic programming: Proceedings of the Tenth International Conference* (pp. 203–221). Cambridge, MA: MIT Press.

Pirri, F., and Reiter, R. (1999). Some contributions to the metatheory of the situation calculus. *Journal of the ACM, 46*(3), 325–361.

Plaisted, D. A., and Greenbaum, S. (1986). A structure-preserving clause form translation. *Journal of Symbolic Computation, 2*, 293–304.

Portinale, L. (1994). *Petri net models for diagnostic knowledge representation and reasoning*. Unpublished doctoral dissertation, Università di Torino, Turin, Italy.

Pratt, V. R. (1976). Semantical considerations on Floyd-Hoare logic. In *Proceedings of the Seventeenth Symposium on the Foundations of Computer Science* (pp. 109–121). Los Alamitos, CA: IEEE Computer Society.

Priest, G. (2002). Paraconsistent logic. In D. M. Gabbay and F. Guenthner (Eds.), *Handbook of philosophical logic* (2nd ed., Vol. 6, pp. 287–393). Dordrecht: Kluwer.

Pylyshyn, Z. W. (Ed.). (1987). *The robot's dilemma: The frame problem in artificial intelligence*. Norwood, NJ: Ablex.

Quillian, M. R. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic information processing* (pp. 227–270). Cambridge, MA: MIT Press.

Ram, A., and Moorman, K. (Eds.). (1999). *Understanding language understanding: Computational models of reading*. Cambridge, MA: MIT Press.

Randell, D. A., Cui, Z., and Cohn, A. G. (1992). A spatial logic based on regions and connection. In B. Nebel, C. Rich, and W. R. Swartout (Eds.), *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (pp. 165–176). San Mateo, CA: Morgan Kaufmann.

Rao, A. S., and Georgeff, M. P. (1992). An abstract architecture for rational agents. In B. Nebel, C. Rich, and W. R. Swartout (Eds.), *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (pp. 439–449). San Mateo, CA: Morgan Kaufmann.

Reichgelt, H. (1991). *Knowledge representation: An AI perspective*. Norwood, NJ: Ablex.

Reisig, W. (1985). *Petri nets: An introduction*. Berlin: Springer.

Reiter, R. (1978). On reasoning by default. In *Proceedings of the 1978 workshop on Theoretical Issues in Natural Language Processing* (pp. 210–218). Vancouver, Canada.

Reiter, R. (1980a). Equality and domain closure in first-order databases. *Journal of the ACM, 27*(2), 235–249.

Reiter, R. (1980b). A logic for default reasoning. *Artificial Intelligence, 13*(1–2), 81–132.

Reiter, R. (1982). Circumscription implies predicate completion (sometimes). In D. L. Waltz (Ed.), *Proceedings of the National Conference on Artificial Intelligence* (pp. 418–420). Menlo Park, CA: AAAI Press.

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz (Ed.), *Artificial intelligence and mathematical theory of computation: Papers in honor of John McCarthy* (pp. 359–380). San Diego, CA: Academic Press.

Reiter, R. (1993). Proving properties of states in the situation calculus. *Artificial Intelligence, 64*(2), 337–351.

Reiter, R. (1996). Natural actions, concurrency and continuous time in the situation calculus. In L. C. Aiello, J. Doyle, and S. C. Shapiro (Eds.), *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 2–13). San Francisco: Morgan Kaufmann.

Reiter, R. (2001). *Knowledge in action: Logical foundations for specifying and implementing dynamical systems*. Cambridge, MA: MIT Press.

Reiter, R., and Criscuolo, G. (1981). On interacting defaults. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (Vol. 1, pp. 270–276). Los Altos, CA: William Kaufmann.

Riecken, D. (1994). M: An architecture of integrated agents. *Communications of the ACM, 37*(7), 107–116, 146.

Rips, L. J. (1983). Cognitive processes in propositional reasoning. *Psychological Review, 90*(1), 38–71.

Rips, L. J. (1994). *The psychology of proof: Deductive reasoning in human thinking.* Cambridge, MA: MIT Press.

Roberts, R. B., and Goldstein, I. P. (1977). *The FRL manual* (A. I. Memo 409). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM, 12*(1), 23–41.

Robinson, J. A., and Voronkov, A. (2001a). *Handbook of Automated Reasoning* (Vol. 1). Amsterdam and Cambridge, MA: Elsevier and MIT Press.

Robinson, J. A., and Voronkov, A. (2001b). *Handbook of Automated Reasoning* (Vol. 2). Amsterdam and Cambridge, MA: Elsevier and MIT Press.

Rosenbloom, P. S., Laird, J. E., and Newell, A. (Eds.). (1993). *The Soar papers: Research on integrated intelligence* (Vols. 1–2). Cambridge, MA: MIT Press.

Rumelhart, D. E., McClelland, J. L., and PDP Research Group (Eds.). (1986). *Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 1. Foundations.* Cambridge, MA: MIT Press.

Russell, S. J., and Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5*, 115–135.

Sacerdoti, E. D. (1977). *A structure for plans and behavior.* New York: Elsevier.

Sanders, K. E. (1989). *A logic for emotions: A basis for reasoning about commonsense psychological knowledge* (Tech. Rep. No. CS-89-23). Providence, RI: Department of Computer Science, Brown University.

Sandewall, E. (1972). An approach to the frame problem, and its implementation. In B. Meltzer and D. Michie (Eds.), *Machine intelligence 7* (pp. 195–204). Edinburgh, UK: Edinburgh University Press.

Sandewall, E. (1989a). Combining logic and differential equations for describing real-world systems. In R. J. Brachman, H. J. Levesque, and R. Reiter (Eds.), *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (pp. 412–420). San Mateo, CA: Morgan Kaufmann.

Sandewall, E. (1989b). Filter [*sic*] preferential entailment for the logic of action in almost continuous worlds. In N. S. Sridharan (Ed.), *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 894–899). San Mateo, CA: Morgan Kaufmann.

Sandewall, E. (1991). *Features and fluents: An agenda and logic-based framework for the representation of knowledge about dynamical systems.* Review version

of chapters 1 and 3–7 (Tech. Rep. No. LiTH-IDA-R-91-29). Linköping, Sweden: Department of Computer and Information Science, Linköping University.

Sandewall, E. (1994). *Features and fluents: The representation of knowledge about dynamical systems* (Vol. I). Oxford: Oxford University Press.

Sandewall, E. (1996). Assessments of ramification methods that use static domain constraints. In L. C. Aiello, J. Doyle, and S. C. Shapiro (Eds.), *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 99–110). San Francisco: Morgan Kaufmann.

Sandewall, E. (1997a, November 21). [Response to Lifschitz]. *Electronic Newsletter on Reasoning about Actions and Change.*

Sandewall, E. (1997b, December 9). [Response to Lifschitz]. *Electronic Newsletter on Reasoning about Actions and Change.*

Schank, R. C., and Abelson, R. P. (1977). *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures.* Hillsdale, NJ: Lawrence Erlbaum.

Schank, R. C., Kass, A., and Riesbeck, C. K. (Eds.). (1994). *Inside case-based explanation.* Hillsdale, NJ: Lawrence Erlbaum.

Schank, R. C., and Riesbeck, C. K. (Eds.). (1981). *Inside computer understanding: Five programs plus miniatures.* Hillsdale, NJ: Lawrence Erlbaum.

Schubert, L. K. (1990). Monotonic solution of the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In H. E. Kyburg, Jr., R. P. Loui, and G. N. Carlson (Eds.), *Knowledge representation and defeasible reasoning* (pp. 23–67). Dordrecht: Kluwer.

Schubert, L. K., and Tong, M. (2003). Extracting and evaluating general world knowledge from the Brown corpus. In G. Hirst and S. Nirenburg (Eds.), *Text Meaning: Proceedings of the HLT-NAACL 2003 workshop* (pp. 7–13). East Stroudsburg, PA: Association for Computational Linguistics.

Selman, B., Kautz, H. A., and Cohen, B. (1993). Local search strategies for satisfiability testing. In D. S. Johnson and M. A. Trick (Eds.), *Cliques, coloring, and satisfiability: Second DIMACS implementation challenge* (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26). Boston, MA: American Mathematical Society.

Selman, B., Kautz, H. A., and McAllester, D. A. (1997). Ten challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (pp. 50–54). San Mateo, CA: Morgan Kaufmann.

Shanahan, M. (1989). Prediction is deduction but explanation is abduction. In N. S. Sridharan (Ed.), *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 1055–1060). San Mateo, CA: Morgan Kaufmann.

Shanahan, M. (1990). Representing continuous change in the event calculus. In L. C. Aiello (Ed.), *Proceedings of the Ninth European Conference on Artificial Intelligence* (pp. 598–603). London: Pitman.

Shanahan, M. (1995a). A circumscriptive calculus of events. *Artificial Intelligence, 77*, 249–284.

Shanahan, M. (1995b). Default reasoning about spatial occupancy. *Artificial Intelligence, 74*, 147–163.

Shanahan, M. (1996). Robotics and the common sense informatic situation. In W. Wahlster (Ed.), *Proceedings of the Twelfth European Conference on Artificial Intelligence* (pp. 684–688). Chichester, UK: John Wiley.

Shanahan, M. (1997a). Event calculus planning revisited. In S. Steel and R. Alami (Eds.), *Recent advances in AI planning* (Lecture Notes in Computer Science, Vol. 1348, pp. 390–402). Berlin: Springer.

Shanahan, M. (1997b). *Solving the frame problem.* Cambridge, MA: MIT Press.

Shanahan, M. (1998a). A logical formalisation of Ernie Davis's egg cracking problem. In R. Miller and M. Shanahan (Eds.), *Proceedings of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning.* Queen Mary and Westifield College: London.

Shanahan, M. (1998b, October). [Protocol of online discussion about the article A logical account of the common sense informatic situation for a mobile robot]. *Electronic News Journal on Reasoning about Actions and Change, 2*(10), 328–329.

Shanahan, M. (1999a). The event calculus explained. In M. J. Wooldridge and M. M. Veloso (Eds.), *Artificial intelligence today: Recent trends and developments* (Lecture Notes in Computer Science, Vol. 1600, pp. 409–430). Berlin: Springer.

Shanahan, M. (1999b). The ramification problem in the event calculus. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 140–146). San Mateo, CA: Morgan Kaufmann.

Shanahan, M. (1999c). What sort of computation mediates best between perception and action? In H. J. Levesque and F. Pirri (Eds.), *Logical foundations for cognitive agents: Contributions in honor of Ray Reiter* (pp. 352–369). Berlin: Springer.

Shanahan, M. (2000a). An abductive event calculus planner. *Journal of Logic Programming, 44*(1–3), 207–240.

Shanahan, M. (2000b). *Abductive event calculus planners [Computer software].* From Department of Electrical and Electronic Engineering, Imperial College London Web site: `http://www.iis.ee.ic.ac.uk/~mpsha/planners.html`

Shanahan, M. (2002). The frame problem. In L. Nadel (Ed.), *Encyclopedia of Cognitive Science* (Vol. 2, pp. 144–150). London: Nature Publishing Group.

Shanahan, M. (2004). An attempt to formalise a non-trivial benchmark problem in common sense reasoning. *Artificial Intelligence, 153*, 141–165.

Shanahan, M. (2005). Perception as abduction: Turning sensor data into meaningful representation. *Cognitive Science, 29*, 103–134.

Shanahan, M., and Randell, D. A. (2004). A logic-based formulation of active visual perception. In D. Dubois, C. A. Welty, and M.-A. Williams (Eds.), *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 64–72). Menlo Park, CA: AAAI Press.

Shanahan, M., and Witkowski, M. (2004). Event calculus planning through satisfiability. *Journal of Logic and Computation, 14*(5), 731–745.

Shoham, Y. (1987). Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence, 33*, 89–104.

Shoham, Y. (1988). *Reasoning about change: Time and causation from the standpoint of artificial intelligence*. Cambridge, MA: MIT Press.

Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence, 60*, 51–92.

Siklóssy, L., and Dreussi, J. (1973). An efficient robot planner which generates its own procedures. In *Advance Papers of the Third International Joint Conference on Artificial Intelligence* (pp. 423–430). Los Altos, CA: William Kaufmann.

Simon, H. A. (1967). Motivational and emotional controls of cognition. *Psychological Review, 74*(1), 29–39.

Singh, P. (1998). *Failure-directed reformulation*. Unpublished master's thesis, Massachusetts Institute of Technology, Cambridge, MA.

Singh, P. (2002). The public acquisition of commonsense knowledge. In J. Karlgren (Ed.), *Acquiring (and using) linguistic (and world) knowledge for information access: Papers from 2002 AAAI spring symposium* (pp. 47–52). Menlo Park, CA: AAAI Press.

Singh, P. (2004). Examining the Society of Mind. *Computing and Informatics, 22*(6), 521–543.

Singh, P. (2005). *EM-ONE: An architecture for reflective commonsense thinking*. Unpublished doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Singh, P., Barry, B. A., and Liu, H. (2004). Teaching machines about everyday life. *BT Technology Journal, 22*(4), 227–240.

Singh, P., Lin, T., Mueller, E. T., Lim, G., Perkins, T., and Zhu, W. L. (2002). Open Mind Common Sense: Knowledge acquisition from the general public. In R. Meersman and Z. Tari (Eds.), *On the move to meaningful internet systems 2002: DOA/COOPIS/ODBASE 2002* (Lecture Notes in Computer Science, Vol. 2519, pp. 1223–1237). Berlin: Springer.

Singh, P., and Minsky, M. (2004). An architecture for cognitive diversity. In D. N. Davis (Ed.), *Visions of mind: Architectures for cognition and affect* (pp. 312–331). Hershey, PA: Idea Group.

Singh, P., Minsky, M., and Eslick, I. (2004). Computing commonsense. *BT Technology Journal, 22*(4), 201–210.

Sirbu, M. A., and Tygar, J. D. (1995). NetBill: An Internet commerce system optimized for network delivered services. In *Proceeding of the Fortieth IEEE Computer Society International Conference* (pp. 20–25). Los Alamitos, CA: IEEE Computer Society.

Sloman, A. (2001). Beyond shallow models of emotion. *Cognitive Processing, 2*(1), 177–198.

Smedslund, J. (1997). *The structure of psychological common sense*. Mahwah, NJ: Lawrence Erlbaum.

Solow, D. (2002). *How to read and do proofs: An introduction to mathematical thought processes* (3rd ed.). New York: John Wiley.

Spelke, E. S., Vishton, P. M., and von Hofsten, C. (1995). Object perception, object-directed action, and physical knowledge in infancy. In M. S. Gazzaniga (Ed.), *The cognitive neurosciences* (pp. 165–179). Cambridge, MA: MIT Press.

Stein, L. A. (1992). Resolving ambiguity in nonmonotonic inheritance hierarchies. *Artificial Intelligence, 55*(2–3), 259–310.

Stein, L. A., and Morgenstern, L. (1994). Motivated action theory: A formal theory of causal reasoning. *Artificial Intelligence, 71*(1), 1–42.

Sun, R. (1994). *Integrating rules and connectionism for robust commonsense reasoning.* New York: John Wiley.

Sun, R. (1995). A new approach toward modeling causality in commonsense reasoning. *International Journal of Intelligent Systems, 10,* 581–616.

Sun, R. (1996). Commonsense reasoning with rules, cases, and connectionist models: A paradigmatic comparison. *Fuzzy Sets and Systems, 82*(2), 187–200.

Suppes, P., and Warren, H. (1975). On the generation and classification of defence mechanisms. *International Journal of Psycho-Analysis, 56,* 405–414.

Sutcliffe, G., and Suttner, C. B. (2003). The CADE-18 ATP system competition. *Journal of Automated Reasoning, 31*(1), 23–32.

Sutcliffe, G., and Suttner, C. B. (2005). *The TPTP problem library for automated theorem proving.* From Department of Computer Science, University of Miami Web site: `http://www.cs.miami.edu/~tptp/`

Sycara, K. (1988). Patching up old plans. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society* (pp. 405–411). Hillsdale, NJ: Lawrence Erlbaum.

Synge, J. L. (1960). Classical dynamics. In S. Flügge (Ed.), *Encyclopedia of physics: Vol. III/1. Principles of classical mechanics and field theory* (pp. 1–225). Berlin: Springer.

Thagard, P. (1996). *Mind: Introduction to cognitive science.* Cambridge, MA: MIT Press.

Thanassas, D. (1992). *The phenomenon of commonsense reasoning: Nonmonotonicity, action and information.* Chichester, UK: Ellis Horwood.

Thielscher, M. (1995). Computing ramifications by postprocessing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1994–2000). San Mateo, CA: Morgan Kaufmann.

Thielscher, M. (1996). *Ramification and causality* (Tech. Rep. No. TR-96-003). Berkeley, CA: International Computer Science Institute.

Thielscher, M. (1997). Ramification and causality. *Artificial Intelligence, 89,* 317–364.

Thielscher, M. (1999a). Fluent calculus planning with continuous change. *Linköping Electronic Articles in Computer and Information Science, 4*(011).

Thielscher, M. (1999b). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence, 111,* 277–299.

Thielscher, M. (1999c). *On-line tutorial on the fluent calculus.* From Technischen Universität Dresden Web site: http://www.ki.inf.tu-dresden.de/~mit/FC/Tutorial/index.htm

Thielscher, M. (2000). Modeling actions with ramifications in nondeterministic, concurrent, and continuous domains—and a case study. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence* (pp. 497–502). Menlo Park, CA: AAAI Press.

Thielscher, M. (2001). The qualification problem: A solution to the problem of anomalous models. *Artificial Intelligence, 131,* 1–37.

Thielscher, M. (2005). FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming, 5*(4–5), 533–565.

Tran, N., and Baral, C. (2004a). Encoding probabilistic causal model in probabilistic action language. In D. L. McGuinness and G. Ferguson (Eds.), *Proceedings of the Nineteenth National Conference on Artificial Intelligence and Sixteenth Conference on Innovative Applications of Artificial Intelligence* (pp. 305–310). Menlo Park, CA: AAAI Press.

Tran, N., and Baral, C. (2004b). Reasoning about triggered actions in AnsProlog and its application to molecular interactions in cells. In D. Dubois, C. A. Welty, and M.-A. Williams (Eds.), *Proceedings of the ninth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 554–563). Menlo Park, CA: AAAI Press.

Turing, A. M. (2004). On computable numbers, with an application to the Entscheidungsproblem. In M. Davis (Ed.), *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions* (Corrected ed., pp. 115–154). Mineola, NY: Dover. (Original work published 1936.)

Van Belleghem, K., Denecker, M., and De Schreye, D. (1994). Representing continuous change in the abductive event calculus. In P. V. Hentenryck (Ed.), *Logic programming: Proceedings of the Eleventh international conference* (pp. 225–239). Cambridge, MA: MIT Press.

van Benthem, J., and Doets, K. (2001). Higher-order logic. In D. M. Gabbay and F. Guenthner (Eds.), *Handbook of philosophical logic* (2nd ed., Vol. 1, pp. 189–243). Dordrecht: Kluwer.

van Lambalgen, M., and Hamm, F. (2005). *The proper treatment of events.* Malden, MA: Blackwell.

Velleman, D. J. (1994). *How to prove it: A structured approach.* Cambridge, UK: Cambridge University Press.

Veloso, M. M. (1989). *Nonlinear problem solving using intelligent casual-commitment* (Tech. Rep. No. CMU-CS-89-210). Pittsburgh, PA: Computer Science Department, Carnegie Mellon University.

Waldinger, R. (1977). Achieving several goals simultaneously. In E. Elcock and D. Michie (Eds.), *Machine intelligence 8* (pp. 94–136). Chichester, UK: Ellis Horwood.

Walther, C. (1983). A many-sorted calculus based on resolution and paramodulation. In A. Bundy (Ed.), *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (Vol. 2, pp. 882–891). Los Altos, CA: William Kaufmann.

Walther, C. (1987). *A many-sorted calculus based on resolution and paramodulation.* London: Pitman.

Wansing, H. (1998). Editorial. *Journal of Logic, Language and Information, 7*(3), iii–vii.

Watson, J. B. (1924). *Behaviorism.* New York: W. W. Norton.

Weld, D. S., and de Kleer, J. (Eds.). (1990). *Readings in qualitative reasoning about physical systems.* San Mateo, CA: Morgan Kaufmann.

Whitehead, A. N., and Russell, B. (1925). *Principia mathematica* (2nd ed., Vol. 1). Cambridge, UK: Cambridge University Press.

Wilensky, R. (1983). *Planning and understanding: A computational approach to human reasoning.* Reading, MA: Addison-Wesley.

Wilk, J. (2004). *Dynamic workflow pulling the strings.* Distinguished Project (MEng), London: Department of Computing, Imperial College London.

Williams, M.-A. (1997). Belief revision, implementing belief revision, and interconnections. In G. Antoniou (Ed.), *Nonmonotonic reasoning* (pp. 183–225). Cambridge, MA: MIT Press.

Winikoff, M., Padgham, L., Harland, J., and Thangarajah, J. (2002). Declarative and procedural goals in intelligent agent systems. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams (Eds.), *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 470–481). San Francisco: Morgan Kaufmann.

Winograd, T. (1972). *Understanding natural language.* New York: Academic Press.

Winograd, T. (1975). Frame representations and the declarative/procedural controversy. In D. G. Bobrow and A. M. Collins (Eds.), *Representation and understanding: Studies in cognitive science* (pp. 185–210). New York: Academic Press.

Winston, P. H. (1970). *Learning structural descriptions from examples* (Tech. Rep. No. AITR-231). Cambridge, MA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Winston, P. H. (1977). *Artificial intelligence.* Reading, MA: Addison-Wesley.

Winston, P. H. (1984). *Artificial intelligence* (2nd ed.). Reading, MA: Addison-Wesley.

Wolf, R. S. (2005). *A tour through mathematical logic.* Washington, DC: Mathematical Association of America.

Wolfram, S. (2002). *A new kind of science.* Champaign, IL: Wolfram Media.

Wooldridge, M. (2000). *Reasoning about rational agents.* Cambridge, MA: MIT Press.

Workflow Management Coalition. (1999). *Workflow Management Coalition terminology & glossary.* Document Number WFMC-TC-1011, Document Status—Issue 3.0, Workflow Management Coalition, Winchester, UK.

Yolum, P., and Singh, M. P. (2004). Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence, 42*(1–3), 227–253.

Zernik, U. (Ed.). (1991). *Lexical acquisition: Exploiting on-line resources to build a lexicon.* Hillsdale, NJ: Lawrence Erlbaum.

Index

B

C

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

V

W

X

Y

Z