CHAPTER

The Event Calculus 2

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 two axiomatizations of the event calculus: EC and 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 constants, function symbols, predicate symbols, 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, $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \Rightarrow \beta$, $\alpha \Leftrightarrow \beta$, $\exists \nu_1, \ldots, \nu_n\, \alpha$, or $\forall \nu_1, \ldots, \nu_n\, \alpha$, where α and β are formulas and $\nu_1, \ldots, \nu_n$ are variables.

The *scope* of the quantifier $\exists$ in the formula $\exists \nu\, \alpha$ is α, and the scope of $\forall$ in $\forall \nu\, \alpha$ is α. An occurrence of a variable ν in a formula Γ that is within a formula of the form $(\exists \nu\, \alpha)$ or $(\forall \nu\, \alpha)$ in Γ 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. We write this inference rule as

$$\frac{\alpha \quad \alpha \Rightarrow \beta}{\beta}$$

If a proof of a formula π given a formula ψ exists, then we write $\psi \vdash \pi$ and say that π is *provable* from ψ.

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 Section A.6.1 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 $\Gamma_1 \overset{\text{def}}{\equiv} \Gamma_2$ defines Γ_1 as an abbreviation for Γ_2. That is, $\Gamma_1 \overset{\text{def}}{\equiv} \Gamma_2$ 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 of the event calculus: 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 EC

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.

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

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.

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

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.

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

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.

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

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 $t_1 + t_2$.

$$Happens(e,t_1) \wedge Initiates(e,f_1,t_1) \wedge 0 < t_2 \wedge$$

$$\mathit{Trajectory}(f_1, t_1, f_2, t_2) \wedge \neg \mathit{StoppedIn}(t_1, f_1, t_1 + t_2) \Rightarrow \\ \mathit{HoldsAt}(f_2, t_1 + t_2)$$

Axiom EC6. If $\mathit{AntiTrajectory}(f_1, t_1, f_2, t_2)$, $0 < t_2$, 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 $t_1 + t_2$.

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

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.

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

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.

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

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.

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

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.

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

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.

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

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.

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

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.

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

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.

$$\begin{gathered}\mathit{Happens}(e,t_1) \wedge \mathit{Initiates}(e,f,t_1) \wedge t_1 < t_2 \wedge \\ \neg\mathit{StoppedIn}(t_1,f,t_2) \wedge \neg\mathit{ReleasedIn}(t_1,f,t_2) \Rightarrow \\ \mathit{HoldsAt}(f,t_2)\end{gathered}$$

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.

$$\begin{gathered}\mathit{Happens}(e,t_1) \wedge \mathit{Terminates}(e,f,t_1) \wedge t_1 < t_2 \wedge \\ \neg\mathit{StartedIn}(t_1,f,t_2) \wedge \neg\mathit{ReleasedIn}(t_1,f,t_2) \Rightarrow \\ \neg\mathit{HoldsAt}(f,t_2)\end{gathered}$$

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.

$$\begin{gathered}\mathit{Happens}(e,t_1) \wedge \mathit{Releases}(e,f,t_1) \wedge t_1 < t_2 \wedge \\ \neg\mathit{StoppedIn}(t_1,f,t_2) \wedge \neg\mathit{StartedIn}(t_1,f,t_2) \Rightarrow \\ \mathit{ReleasedAt}(f,t_2)\end{gathered}$$

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.

$$\begin{gathered}\mathit{Happens}(e,t_1) \wedge (\mathit{Initiates}(e,f,t_1) \vee \mathit{Terminates}(e,f,t_1)) \wedge \\ t_1 < t_2 \wedge \neg\mathit{ReleasedIn}(t_1,f,t_2) \Rightarrow \\ \neg\mathit{ReleasedAt}(f,t_2)\end{gathered}$$

We use EC to mean the conjunction of the 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 $\mathit{Clipped}(t_1,f,t_2)$ as an abbreviation for $\exists e,t\,(\mathit{Happens}(e,t) \wedge t_1 \leq t < t_2 \wedge \mathit{Terminates}(e,f,t))$. An axiomatization that supports event occurrences with duration is presented in Appendix C.

2.3.2 DEC

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.

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

Definition DEC2.

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

Trajectory and AntiTrajectory

The axioms for trajectories are the same as in EC.

Axiom DEC3.

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

Axiom DEC4.

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

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 $t + 1$.

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

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 $t + 1$.

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

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 $t+1$.

$$\begin{gathered} ReleasedAt(f,t) \land \\ \neg\exists e\,(Happens(e,t) \land (Initiates(e,f,t) \lor Terminates(e,f,t))) \Rightarrow \\ ReleasedAt(f,t+1) \end{gathered}$$

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 $t+1$.

$$\begin{gathered} \neg ReleasedAt(f,t) \land \neg\exists e\,(Happens(e,t) \land Releases(e,f,t)) \Rightarrow \\ \neg ReleasedAt(f,t+1) \end{gathered}$$

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 $t+1$.

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

Axiom DEC10. If a fluent is terminated by some event that occurs at timepoint t, then the fluent is false at $t+1$.

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

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 $t+1$.

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

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 $t+1$.

$$\begin{gathered} Happens(e,t) \land (Initiates(e,f,t) \lor Terminates(e,f,t)) \Rightarrow \\ \neg ReleasedAt(f,t+1) \end{gathered}$$

We use DEC to mean the conjunction of the 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 EC AND DEC

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 DEC 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 or answer set programming implementations 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:

$$\begin{gathered} \mathit{PickUp}(\mathit{Nathan}, \mathit{Glass}) \\ \mathit{SetDown}(\mathit{Ryan}, \mathit{Bowl}) \end{gathered}$$

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:

$$\begin{gathered} \mathit{PickUp}(\mathit{Nathan}, \mathit{Glass1}) \\ \mathit{PickUp}(\mathit{Nathan}, \mathit{Glass2}) \end{gathered}$$

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 $\mathit{HoldsAt}(\beta, \tau)$ is a condition.
- If γ is a condition, then $\neg\gamma$ is a condition.
- If γ_1 and γ_2 are conditions, then $\gamma_1 \wedge \gamma_2$ and $\gamma_1 \vee \gamma_2$ are conditions.
- If ν is a variable and γ is a condition, then $\exists \nu\, \gamma$ is a condition.

- If ρ is an n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ are terms, then $\rho(\tau_1, \ldots, \tau_n)$ is a condition.
- Nothing else is a condition.

(Inductive definitions are explained in Section A.2.) Atoms of the form $\rho(\tau_1, \ldots, \tau_n)$ can be used in a condition when it is necessary to incorporate general truths. Examples in this book include $Arc(s, a_1, a_2)$, which represents that the appearance of shape s can change from a_1 to a_2 by shifting the viewing angle (Section 14.3), and $ElementOf(x, l)$, which represents that x is an element of list l (Section A.6.2).

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

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

$CIRC[(2.3); Terminates]$ is

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

$CIRC[(2.4); Happens]$ is

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

Circumscription is defined using second-order logic as described in Section A.7.

Nonmonotonic reasoning

First-order logic entailment is *monotonic*: If $\psi \models \pi$, then $\psi \land \psi' \models \pi$ for every ψ'; also, if $\psi \vdash \pi$, then $\psi \land \psi' \vdash \pi$ 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 $\psi \land \psi'$ entails π.

Suppose that the only known event occurrence is given by (2.4). Then from (2.7)–(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

The algorithms for computing circumscription in general are complex. 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 2.3)

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

$$(e = E1 \land t = T1) \lor (e = E2 \land t = T2) \Leftrightarrow Happens(e, t)$$

2.6.3 EXAMPLE: CIRCUMSCRIPTION OF INITIATES

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

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

and then applying Theorem 2.1, which gives

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

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.

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***.

Table 2.1 Event Calculus Formulas Used for Commonsense Reasoning[a]

Formula	**Form**	**Defined in**
Unique names axiom $[\Omega]$	$U[\phi_1, \ldots, \phi_n]$	Section 2.4.1
Observation $[\Gamma]$	$(\neg)\mathit{HoldsAt}(\beta, \tau)$ or $(\neg)\mathit{ReleasedAt}(\beta, \tau)$	Section 2.7
Event occurrence formula $[\Delta_1]$	$\mathit{Happens}(\alpha, \tau)$	Section 2.7
Temporal ordering formula $[\Delta_1]$	$\tau_1 < \tau_2,\ \tau_1 \le \tau_2,\ \tau_1 = \tau_2, \ldots$	Section 2.7
Positive effect axiom $[\Sigma]$	$\gamma \Rightarrow \mathit{Initiates}(\alpha, \beta, \tau)$	Section 3.1
Negative effect axiom $[\Sigma]$	$\gamma \Rightarrow \mathit{Terminates}(\alpha, \beta, \tau)$	Section 3.1
Fluent precondition axiom $[\Sigma]$	$\gamma \Rightarrow \delta(\alpha, \beta, \tau)$	Section 3.3.1
Action precondition axiom $[\Psi]$	$\mathit{Happens}(\alpha, \tau) \Rightarrow \gamma$	Section 3.3.2
State constraint $[\Psi]$	γ_1, $\gamma_1 \Rightarrow \gamma_2$, or $\gamma_1 \Leftrightarrow \gamma_2$	Section 3.4
Trigger axiom $[\Delta_2]$	$\gamma \Rightarrow \mathit{Happens}(\alpha, \tau)$	Section 4.1
Release axiom $[\Sigma]$	$\gamma \Rightarrow \mathit{Releases}(\alpha, \beta, \tau)$	Section 5.3
Effect constraint $[\Sigma]$	$\gamma \wedge \delta_1(\alpha, \beta_1, \tau) \Rightarrow \delta_2(\alpha, \beta_2, \tau)$	Section 6.4
Causal constraint $[\Delta_2]$	$\theta(\beta, \tau) \wedge \pi_1(\beta_1, \tau) \wedge \cdots \wedge \pi_n(\beta_n, \tau) \Rightarrow \mathit{Happens}(\alpha, \tau)$	Section 6.5
Trajectory axiom $[\Pi]$	$\gamma \Rightarrow \mathit{Trajectory}(\beta_1, \tau_1, \beta_2, \tau_2)$	Section 7.1
Antitrajectory axiom $[\Pi]$	$\gamma \Rightarrow \mathit{AntiTrajectory}(\beta_1, \tau_1, \beta_2, \tau_2)$	Section 7.2
Event occurrence constraint $[\Psi]$	$\mathit{Happens}(\alpha_1, \tau) \wedge \gamma \Rightarrow (\neg)\mathit{Happens}(\alpha_2, \tau)$	Section 8.1.2
Positive cumulative effect axiom $[\Sigma]$	$\gamma \wedge (\neg)\mathit{Happens}(\alpha_1, \tau) \wedge \cdots \wedge (\neg)\mathit{Happens}(\alpha_n, \tau) \Rightarrow \mathit{Initiates}(\alpha, \beta, \tau)$	Section 8.2
Negative cumulative effect axiom $[\Sigma]$	$\gamma \wedge (\neg)\mathit{Happens}(\alpha_1, \tau) \wedge \cdots \wedge (\neg)\mathit{Happens}(\alpha_n, \tau) \Rightarrow \mathit{Terminates}(\alpha, \beta, \tau)$	Section 8.2
Disjunctive event axiom $[\Delta_2]$	$\mathit{Happens}(\alpha, \tau) \Rightarrow \mathit{Happens}(\alpha_1, \tau) \vee \cdots \vee \mathit{Happens}(\alpha_n, \tau)$	Section 9.2
Cancellation axiom $[\Theta]$	$\gamma \Rightarrow \mathit{Ab}_i(\ldots, \tau)$	Section 12.3

[a] $\alpha, \alpha_1, \ldots, \alpha_n$ = *event terms;* $\beta, \beta_1, \ldots, \beta_n$ = *fluent terms;* $\gamma, \gamma_1, \gamma_2$ = *conditions;* $\delta, \delta_1, \delta_2$ = *Initiates or Terminates;* $\pi_1, \ldots, \pi_n$ = *Initiated or Terminated;* θ = *Stopped or Started;* τ, τ_1, τ_2 = *timepoint terms;* $\phi_1, \ldots, \phi_n$ = *function symbols.*

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

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

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 axioms such as EC or DEC.
- CC is the conjunction of axioms to support causal constraints defined in Section 6.5.

The axiomatization consists of Σ, Δ_2, Θ, Ω, Ψ, Π, E, and CC, 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*

$$\begin{aligned} &CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \\ &\qquad \Omega \wedge \Gamma \wedge \text{EC} \vdash HoldsAt(Awake(Nathan), 3) \end{aligned}$$

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

$$\begin{aligned} &(Initiates(e,f,t) \Leftrightarrow \exists a\,(e = WakeUp(a) \wedge f = Awake(a))) \wedge \\ &(Terminates(e,f,t) \Leftrightarrow \exists a\,(e = FallAsleep(a) \wedge f = Awake(a))) \wedge \\ &\neg Releases(e,f,t) \end{aligned} \tag{2.15}$$

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

$$Happens(e,t) \Leftrightarrow (e = WakeUp(Nathan) \wedge t = 1) \tag{2.16}$$

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*

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

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

$$\begin{aligned} &(Initiates(e,f,t) \Leftrightarrow \exists a\,(e = WakeUp(a) \wedge f = Awake(a))) \wedge \\ &(Terminates(e,f,t) \Leftrightarrow \exists a\,(e = FallAsleep(a) \wedge f = Awake(a))) \wedge \\ &\neg Releases(e,f,t) \end{aligned} \tag{2.17}$$

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

$$(e = WakeUp(Nathan) \wedge t = 1) \Leftrightarrow Happens(e, t) \tag{2.18}$$

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

$$\neg ReleasedAt(Awake(Nathan), 1) \tag{2.19}$$

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, and answer set programming discussed in Chapter 15, 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, E, and CC 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

$$\begin{aligned}&\mathit{CIRC}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathit{CIRC}[\Delta; \mathit{Happens}] \wedge\\ &\quad \mathit{CIRC}[\Theta; \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge \mathrm{E} \wedge \mathrm{CC} \vdash \Gamma'\end{aligned}$$

Propositions 2.1 and 2.2 are deduction problems. From the fact that Nathan woke up, we can deduce that afterward he was awake.

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 $\Phi =$

$$\begin{aligned}&\mathit{CIRC}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathit{CIRC}[\Delta_1 \wedge \Delta_2; \mathit{Happens}] \wedge\\ &\quad \mathit{CIRC}[\Theta; \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge \Omega \wedge \Psi \wedge \Pi \wedge \Gamma \wedge \mathrm{E} \wedge \mathrm{CC}\end{aligned}$$

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

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 answer set programming. We create a file called `sleep.pl` containing the following:

```
initiated(F,T) :- happens(E,T), initiates(E,F,T).
terminated(F,T) :- happens(E,T), terminates(E,F,T).
holdsAt(F,T+1) :- holdsAt(F,T), not terminated(F,T), time(T).
:- holdsAt(F,T+1), not holdsAt(F,T), not initiated(F,T), time(T).
holdsAt(F,T+1) :- happens(E,T), initiates(E,F,T).
:- holdsAt(F,T+1), happens(E,T), terminates(E,F,T).
{holdsAt(F,T)} :- fluent(F), time(T).
{happens(E,T)} :- event(E), time(T).

time(0).
fluent(awake(nathan)).
event(wakeUp(nathan)).
event(fallAsleep(nathan)).

initiates(wakeUp(nathan),awake(nathan),T) :- time(T).
terminates(fallAsleep(nathan),awake(nathan),T) :- time(T).

:- holdsAt(awake(nathan),0).
holdsAt(awake(nathan),1).
```

We then run `clingo -n 0 sleep.lp`, which produces the following output:

```
clingo version 4.2.1
Reading from sleep.lp
Solving...
Answer: 1
time(0) fluent(awake(nathan)) event(wakeUp(nathan))
event(fallAsleep(nathan)) holdsAt(awake(nathan),1)
initiates(wakeUp(nathan),awake(nathan),0)
initiated(awake(nathan),0) happens(wakeUp(nathan),0)
terminates(fallAsleep(nathan),awake(nathan),0)
SATISFIABLE

Models   : 1
Calls    : 1
Time     : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
```

The `clingo` option `-n 0` is used to request all stable models. The result is one stable model in which `happens(wakeUp(nathan),0)` is abduced.

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 $\Phi =$

$$CIRC[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \land CIRC[\Delta; \mathit{Happens}] \land \\ CIRC[\Theta; \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \land \Omega \land \Psi \land \Pi \land \Gamma \land \mathrm{E} \land \mathrm{CC}$$

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

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:

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

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, E, and CC as input, and producing as output zero or more structures $\mathcal{M}$ such that

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

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 and 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). Running XOR is a special case of the AND-OR connective of Shapiro (1979, pp. 190 and 191). 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 & 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

1. Original logic programming version of the event calculus (Kowalski & 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 & 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).

1. Event calculus (Shanahan, 1998, 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 & 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 & 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)
6. Discrete event calculus with branching time (Mueller, 2007a)
7. Epistemic functional event calculus (R. Miller, Morgenstern, & Patkos, 2013)

FIGURE 2.2

Evolution of the classical logic event calculus (1998-2013).

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*

Table 2.2 Evolution of the *HoldsAt* axiom

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

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, and 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 names assumption (Genesereth & 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).

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; 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 and 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:

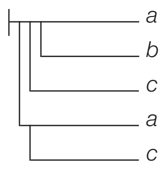
$$Before(i_1, i_2) \Leftrightarrow End(i_1) < Start(i_2)$$
$$Equal(i_1, i_2) \Leftrightarrow (Start(i_1) = Start(i_2) \wedge End(i_1) = End(i_2))$$
$$Meets(i_1, i_2) \Leftrightarrow End(i_1) = Start(i_2)$$

Nonmonotonic reasoning

Minsky (1974, p. 75) criticized the monotonicity of logic, which led (Israel, 1985; Minsky, 1991b, p. 378) to the development of nonmonotonic reasoning methods (Bobrow, 1980). Minsky (1974) used the word *monotonicity* after a suggestion from Vaughan R. Pratt (McDermott & 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 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

Table 2.3 Monotonicity of Logic

(Frege, 1879/1967, p. 43)	
(Whitehead & Russell, 1925, p. 113)	$\vdash :. p \supset r . \supset : p . q . \supset . r$
(Gentzen, 1935/1969, p. 83)	$\dfrac{\Gamma \rightarrow \Theta}{D, \Gamma \rightarrow \Theta}$
(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 $B \vdash \alpha$

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, 1985b). 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 and 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 & Ullman, 1979, pp. 119 and 120; Nerode & Shore, 1997, p. 126). See also the discussions of M. Davis (1980), McCarthy (1980), and Genesereth and Nilsson (1987). Predicate completion was introduced by K. L. Clark (1978, pp. 303-305). Genesereth and Nilsson (1987) and Russell and Norvig (2009) 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; 2000), 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).

EXERCISES

2.1 List 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:

$$\begin{gathered} P(x) \wedge Q(x) \Rightarrow R(x) \\ R(A) \\ R(B) \end{gathered}$$

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.