Handbook of Knowledge Representation
Edited by F. van Harmelen, V. Lifschitz and B. Porter

DOI: 10.1016/S1574-6526(07)03017-9

Chapter 17

Event Calculus

Erik T. Mueller

17.1 Introduction

The event calculus [45, 66, 74, 98, 100] is a formalism for reasoning about action and change. Like the situation calculus, the event calculus has actions, which are called *events*, and time-varying properties or *fluents*. In the situation calculus, performing an action in a situation gives rise to a successor situation. Situation calculus actions are hypothetical, and time is tree-like. In the event calculus, there is a single time line on which actual events occur.

A *narrative* is a possibly incomplete specification of a set of actual event occurrences [63, 98]. The event calculus is narrative-based, unlike the standard situation calculus in which an exact sequence of hypothetical actions is represented.

Like the situation calculus, the event calculus supports context-sensitive effects of events, indirect effects, action preconditions, and the commonsense law of inertia. Certain phenomena are addressed more naturally in the event calculus, including concurrent events, continuous time, continuous change, events with duration, nondeterministic effects, partially ordered events, and triggered events.

We use a simple example to illustrate what the event calculus does. Suppose we wish to reason about turning on and off a light. We start by representing general knowledge about the effects of events:

> If a light's switch is flipped up, then the light will be on.
> If a light's switch is flipped down, then the light will be off.

We then represent a specific scenario:

> The light was off at time 0.
> Then the light's switch was flipped up at time 5.
> Then the light's switch was flipped down at time 8.

We use the event calculus to conclude the following:

> At time 3, the light was off.
> At time 7, the light was on.
> At time 10, the light was off.

Table 17.1. Original event calculus (OEC) predicates and functions (e, e_1, e_2 = event occurrences, f, f_1, f_2 = fluents, p = time period)

Predicate/function	Meaning
$Holds(p)$	p holds
$Start(p, e)$	e starts p
$End(p, e)$	e ends p
$Initiates(e, f)$	e initiates f
$Terminates(e, f)$	e terminates f
$e_1 < e_2$	e_1 precedes e_2
$Broken(e_1, f, e_2)$	f is broken between e_1 and e_2
$Incompatible(f_1, f_2)$	f_1 and f_2 are incompatible
$After(e, f)$	time period after e in which f holds
$Before(e, f)$	time period before e in which f holds

In this chapter, we discuss several versions of the event calculus, the use of circumscription in the event calculus, methods of knowledge representation using the event calculus, automated event calculus reasoning, and applications of the event calculus. We use languages of classical many-sorted predicate logic with equality.[1]

17.2 Versions of the Event Calculus

The event calculus has evolved considerably from its original version. In this section, we trace the development of the event calculus and present its important versions.

17.2.1 Original Event Calculus (OEC)

The original event calculus (OEC) was introduced in 1986 by Kowalski and Sergot [45]. OEC has sorts for event occurrences, fluents, and time periods. The predicates and functions of the original event calculus are shown in Table 17.1. The axioms of the original event calculus are as follows.

OEC1. $Initiates(e, f) \equiv Holds(After(e, f))$[2].

OEC2. $Terminates(e, f) \equiv Holds(Before(e, f))$.

OEC3. $Start(After(e, f), e)$.

OEC4. $End(Before(e, f), e)$.

OEC5. $After(e_1, f) = Before(e_2, f) \supset Start(Before(e_2, f), e_1)$.

OEC6. $After(e_1, f) = Before(e_2, f) \supset End(After(e_1, f), e_2)$.

[1] We do not treat modal logic versions of the event calculus [9].

[2] Kowalski and Sergot [45] use implication ($\supset$) in OEC1 and OEC2. Sadri [87, p. 134] points out that bi-implication ($\equiv$) was intended by Kowalski and Sergot but not used in order to prevent looping when running the axioms in Prolog.

OEC7. $Holds(After(e_1, f)) \wedge Holds(Before(e_2, f)) \wedge e_1 < e_2 \wedge \neg Broken(e_1, f, e_2) \supset After(e_1, f) = Before(e_2, f)$.

OEC8. $Broken(e_1, f, e_2) \equiv \exists e, f_1 ((Holds(After(e, f_1)) \vee Holds(Before(e, f_1))) \wedge Incompatible(f, f_1) \wedge e_1 < e < e_2)$.

Let OEC be the conjunction of OEC1 through OEC8.

Example 17.1. Consider the example of turning on and off a light. We have an event occurrence E_1, which precedes event occurrence E_2:

$$E_1 < E_2 \tag{17.1}$$

E_1 turns on the light and E_2 turns it off:

$$Initiates(e, f) \equiv (e = E_1 \wedge f = On) \vee (e = E_2 \wedge f = Off) \tag{17.2}$$

$$Terminates(e, f) \equiv (e = E_1 \wedge f = Off) \vee (e = E_2 \wedge f = On) \tag{17.3}$$

The light cannot be both on and off:

$$Incompatible(f_1, f_2) \equiv (f_1 = On \wedge f_2 = Off) \vee (f_1 = Off \wedge f_2 = On) \tag{17.4}$$

We also assume the following:

$$E_1 \neq E_2 \tag{17.5}$$

$$On \neq Off \tag{17.6}$$

$$\neg(e < e) \tag{17.7}$$

We can then prove that the time period after E_1 in which the light is on equals the time period before E_2 in which the light is on. Let Σ be the conjunction of (17.1) through (17.7).

Proposition 17.1. $\Sigma \wedge \text{OEC} \models After(E_1, On) = Before(E_2, On)$.

Proof. From (17.2), (17.3), OEC1, and OEC2, we have

$$Holds(After(e, f)) \equiv (e = E_1 \wedge f = On) \vee (e = E_2 \wedge f = Off) \tag{17.8}$$

$$Holds(Before(e, f)) \equiv (e = E_1 \wedge f = Off) \vee (e = E_2 \wedge f = On) \tag{17.9}$$

From (17.8), (17.9), (17.4), (17.6), (17.7), and OEC8, we get $\neg Broken(E_1, On, E_2)$. From this, $Holds(After(E_1, On)$ (which follows from (17.8)), $Holds(Before(E_2, On)$ (which follows from (17.9)), (17.1), and OEC7, we have $After(E_1, On) = Before(E_2, On)$. □

We can also prove that E_1 starts the time period before E_2 in which On holds and that E_2 ends the time period after E_1 in which On holds.

Proposition 17.2. $\Sigma \wedge \text{OEC} \models Start(Before(E_2, On), E_1) \wedge End(After(E_1, On), E_2)$.

Table 17.2. Simplified event calculus (SEC) predicates (e = event, f = fluent, t, t_1, t_2 = timepoints)

Predicate	Meaning
$Initially(f)$	f is true at timepoint 0
$HoldsAt(f, t)$	f is true at t
$Happens(e, t)$	e occurs at t
$Initiates(e, f, t)$	if e occurs at t, then f is true after t
$Terminates(e, f, t)$	if e occurs at t, then f is false after t
$StoppedIn(t_1, f, t_2)$	f is stopped between t_1 and t_2

Proof. This follows from Proposition 17.1, OEC5, and OEC6. □

Pinto and Reiter [82] argue that the *Holds* predicate of the original event calculus is problematic, because it represents that "time periods hold". They point out some undesirable consequences of axioms OEC3 and OEC4. In our light example, $Start(After(E_1, Off), E_1)$ follows from OEC3. But what is the time period $After(E_1, Off)$? Whatever it is, it does not hold. From (17.5), (17.6), and (17.8), we have $\neg Holds(After(E_1, Off))$. Similarly, from OEC3, we get $Start(After(E_2, On), E_2)$ and from OEC4, we get $End(Before(E_2, Off), E_2)$. Sadri and Kowalski [88] suggest modifying OEC3 to $Holds(After(e, f)) \supset Start(After(e, f), e)$ and OEC4 to $Holds(Before(e, f)) \supset End(Before(e, f), e)$.

17.2.2 Simplified Event Calculus (SEC)

The simplified event calculus (SEC) was proposed in 1986 by Kowalski [40, p. 25] (see also [41]) and developed by Sadri [87, pp. 137–139], Eshghi [17], and Shanahan [93, 94, 98]. It differs from the original event calculus in the following ways:

- It replaces time periods with timepoints, which are either nonnegative integers or nonnegative real numbers.
- It replaces event occurrences or tokens with event types. The predicate $Happens(e, t)$ represents that event (type) e occurs at timepoint t.
- It eliminates the notion of incompatible fluents.
- It adds a predicate $Initially(f)$, which represents that fluent f is initially true [95, p. 254] (see also [12] and [98, p. 253]).

The predicates of the simplified event calculus are shown in Table 17.2. The axioms of the simplified event calculus are as follows.

SEC1. $((Initially(f) \wedge \neg StoppedIn(0, f, t)) \vee \exists e, t_1\ (Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge t_1 < t \wedge \neg StoppedIn(t_1, f, t))) \equiv HoldsAt(f, t)$[3].

SEC2. $StoppedIn(t_1, f, t_2) \equiv \exists e, t\ (Happens(e, t) \wedge t_1 < t < t_2 \wedge Terminates(e, f, t))$.

[3] Kowalski [40] uses implication, whereas Sadri [87] uses bi-implication.

Let SEC be the conjunction of SEC1 and SEC2.

SEC1 represents that (1) a fluent that is initially true remains true until it is terminated, and (2) a fluent that is initiated remains true until it is terminated. Thus fluents are subject to the *commonsense law of inertia* [48, 49, 98], which states that a fluent's truth value persists unless the fluent is affected by an event.

Example 17.2. Consider again the example of turning on and off a light. If a light is turned on, it will be on, and if a light is turned off, it will no longer be on:

$$Initiates(e, f, t) \equiv (e = TurnOn \wedge f = On) \tag{17.10}$$

$$Terminates(e, f, t) \equiv (e = TurnOff \wedge f = On) \tag{17.11}$$

Initially, the light is off:

$$\neg Initially(On) \tag{17.12}$$

The light is turned on at timepoint 2 and turned off at timepoint 4:

$$Happens(e, t) \equiv (e = TurnOn \wedge t = 2) \vee (e = TurnOff \wedge t = 4) \tag{17.13}$$

We further assume the following:

$$TurnOn \neq TurnOff \tag{17.14}$$

We can then show that the light will be off at timepoint 1, on at timepoint 3, and off again at timepoint 5. Let Σ be the conjunction of (17.10) through (17.14).

Proposition 17.3. $\Sigma \wedge \text{SEC} \models \neg HoldsAt(On, 1)$.

Proof. From (17.13), we have $\neg\exists e, t_1\,(Happens(e, t_1) \wedge Initiates(e, On, t_1) \wedge t_1 < 1 \wedge \neg StoppedIn(t_1, On, 1))$. From this, (17.12), and SEC1, we have $\neg HoldsAt(On, 1)$. □

Proposition 17.4. $\Sigma \wedge \text{SEC} \models HoldsAt(On, 3)$.

Proof. From (17.13) and SEC2, we have $\neg StoppedIn(2, On, 3)$. From this, $Happens(TurnOn, 2)$ (which follows from (17.13)), $Initiates(TurnOn, On, 2)$ (which follows from (17.10)), $2 < 3$, and SEC1, we have $HoldsAt(On, 3)$. □

Proposition 17.5. $\Sigma \wedge \text{SEC} \models \neg HoldsAt(On, 5)$.

Proof. From $Happens(TurnOff, 4)$ (which follows from (17.13)), $2 < 4 < 5$, $Terminates(TurnOff, On, 4)$ (which follows from (17.11)), and SEC2, we have $StoppedIn(2, On, 5)$). From this, (17.13), and (17.10), we have $\neg\exists e, t_1\,(Happens(e, t_1) \wedge Initiates(e, On, t_1) \wedge t_1 < 5 \wedge \neg StoppedIn(t_1, On, 5))$. From this, (17.12), and SEC1, we have $\neg HoldsAt(On, 5)$. □

Table 17.3. Basic event calculus (BEC) predicates (e = event, f, f_1, f_2 = fluents, t, t_1, t_2 = timepoints)

Predicate	Meaning
InitiallyN(f)	f is false at timepoint 0
InitiallyP(f)	f is true at timepoint 0
HoldsAt(f, t)	f is true at t
Happens(e, t)	e occurs at t
Initiates(e, f, t)	if e occurs at t, then f is true and not released from the commonsense law of inertia after t
Terminates(e, f, t)	if e occurs at t, then f is false and not released from the commonsense law of inertia after t
Releases(e, f, t)	if e occurs at t, then f is released from the commonsense law of inertia after t
StoppedIn(t_1, f, t_2)	f is stopped between t_1 and t_2
StartedIn(t_1, f, t_2)	f is started between t_1 and t_2
Trajectory(f_1, t_1, f_2, t_2)	if f_1 is initiated by an event that occurs at t_1, then f_2 is true at $t_1 + t_2$

17.2.3 Basic Event Calculus (BEC)

Shanahan [94–96, 98] extended the simplified event calculus by allowing fluents to be released from the commonsense law of inertia via the *Releases* predicate, and adding the ability to represent continuous change via the *Trajectory* predicate. The *Initially* predicate is broken into two predicates *InitiallyP* and *InitiallyN*. We call this version of the event calculus the basic event calculus (BEC).

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 after t. In SEC, a fluent that is initiated remains true until it is terminated, and a fluent that is terminated remains false until it is initiated. In BEC, a fluent that is initiated remains true until it is terminated or released, and a fluent that is terminated remains false until it is initiated or released. After a fluent is released, its truth value is not determined by BEC and is permitted to vary. Thus there are models in which the fluent is true, and models in which the fluent is false.

This opens up several possibilities. First, releasing a fluent frees it up so that other axioms in the domain description can be used to determine its truth value. This allows us to represent continuous change using *Trajectory*, as discussed in Section 17.5.7, and indirect effects, as discussed in Section 17.5.9. Second, released fluents can be used to represent nondeterministic effects of events, as discussed in Section 17.5.8.

Trajectory(f_1, t_1, f_2, t_2) represents that, if fluent f_1 is initiated by an event that occurs at timepoint t_1, then fluent f_2 will be true at timepoint $t_1 + t_2$. This can be used to represent fluents that change as a function of time. The domain description is usually written so that the fluent f_2 is released by the events that initiate f_1.

The predicates of the basic event calculus are shown in Table 17.3. The axioms of the basic event calculus are as follows.

BEC1. $\mathit{StoppedIn}(t_1, f, t_2) \equiv \exists e, t\ (\mathit{Happens}(e, t) \wedge t_1 < t < t_2 \wedge (\mathit{Terminates}(e, f, t) \vee \mathit{Releases}(e, f, t)))$.

BEC2. $StartedIn(t_1, f, t_2) \equiv \exists e, t\,(Happens(e, t) \wedge t_1 < t < t_2 \wedge (Initiates(e, f, t) \vee Releases(e, f, t)))$.

BEC3. $Happens(e, t_1) \wedge Initiates(e, f_1, t_1) \wedge 0 < t_2 \wedge Trajectory(f_1, t_1, f_2, t_2) \wedge \neg StoppedIn(t_1, f_1, t_1 + t_2) \supset HoldsAt(f_2, t_1 + t_2)$.

BEC4. $InitiallyP(f) \wedge \neg StoppedIn(0, f, t) \supset HoldsAt(f, t)$.

BEC5. $InitiallyN(f) \wedge \neg StartedIn(0, f, t) \supset \neg HoldsAt(f, t)$.

BEC6. $Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StoppedIn(t_1, f, t_2) \supset HoldsAt(f, t_2)$.

BEC7. $Happens(e, t_1) \wedge Terminates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StartedIn(t_1, f, t_2) \supset \neg HoldsAt(f, t_2)$.

Let BEC be the conjunction of BEC1 through BEC7.

Example 17.3. Consider once again the example of turning on and off a light. We replace (17.12) with the following:

$$InitiallyN(On) \tag{17.15}$$

We add the following:

$$\neg Releases(e, f, t) \tag{17.16}$$

Let Σ be the conjunction of (17.10), (17.11), (17.13), (17.14), (17.15), and (17.16). We then have the same results as for SEC.

Proposition 17.6. $\Sigma \wedge \text{BEC} \models \neg HoldsAt(On, 1)$.

Proof. From (17.13) and BEC2, we have $\neg StartedIn(0, On, 1)$. From this, (17.15), and BEC5, we have $\neg HoldsAt(On, 1)$. □

Proposition 17.7. $\Sigma \wedge \text{BEC} \models HoldsAt(On, 3)$.

Proof. From (17.13) and BEC1, we have $\neg StoppedIn(2, On, 3)$. From this, $Happens(TurnOn, 2)$ (which follows from (17.13)), $Initiates(TurnOn, On, 2)$ (which follows from (17.10)), $2 < 3$, and BEC6, we have $HoldsAt(On, 3)$. □

Proposition 17.8. $\Sigma \wedge \text{BEC} \models \neg HoldsAt(On, 5)$.

Proof. From (17.13) and BEC2, we have $\neg StartedIn(4, On, 5)$. From this, $Happens(TurnOff, 4)$ (which follows from (17.13)), $Terminates(TurnOff, On, 4)$ (which follows from (17.11)), $4 < 5$, and BEC7, we have $\neg HoldsAt(On, 5)$. □

Example 17.4. We can use *Releases* and *Trajectory* to represent a light that alternately emits red and green when it is turned on. If a light is turned on, it will be on:

$$Initiates(e, f, t) \equiv (e = TurnOn \wedge f = On) \tag{17.17}$$

If a light is turned on, whether it is red or green will be released from the commonsense law of inertia:

$$\mathit{Releases}(e, f, t) \equiv (e = \mathit{TurnOn} \wedge (f = \mathit{Red} \vee f = \mathit{Green})) \tag{17.18}$$

If a light is turned off, it will not be on, red, or green:

$$\mathit{Terminates}(e, f, t) \equiv (e = \mathit{TurnOff} \wedge (f = \mathit{On} \vee f = \mathit{Red} \vee f = \mathit{Green})) \tag{17.19}$$

After a light is turned on, it will alternately emit red for two seconds and green for two seconds:

$$(t_2 \bmod 4) < 2 \supset \mathit{Trajectory}(\mathit{On}, t_1, \mathit{Red}, t_2) \tag{17.20}$$

$$(t_2 \bmod 4) \geqslant 2 \supset \mathit{Trajectory}(\mathit{On}, t_1, \mathit{Green}, t_2) \tag{17.21}$$

The light is not simultaneously red and green:

$$\neg \mathit{HoldsAt}(\mathit{Red}, t) \vee \neg \mathit{HoldsAt}(\mathit{Green}, t) \tag{17.22}$$

The light is turned on at timepoint 2:

$$\mathit{Happens}(e, t) \equiv (e = \mathit{TurnOn} \wedge t = 2) \tag{17.23}$$

We also assume

$$\mathit{TurnOn} \neq \mathit{TurnOff} \tag{17.24}$$

We can then show that the light will be red at timepoint 3, green at timepoint 5, red at timepoint 7, and so on. Let Σ be the conjunction of (17.17) through (17.24).

Proposition 17.9. $\Sigma \wedge \mathrm{BEC} \models \mathit{HoldsAt}(\mathit{Red}, 3)$.

Proof. From (17.20) by universal instantiation, we have

$$\mathit{Trajectory}(\mathit{On}, 2, \mathit{Red}, 1) \tag{17.25}$$

From (17.23) and BEC1, we have $\neg \mathit{StoppedIn}(2, \mathit{On}, 3)$. From this, $\mathit{Happens}(\mathit{TurnOn}, 2)$ (which follows from (17.23)), $\mathit{Initiates}(\mathit{TurnOn}, \mathit{On}, 2)$ (which follows from (17.17)), $0 < 1$, (17.25), and BEC3, we have $\mathit{HoldsAt}(\mathit{Red}, 3)$. □

Proposition 17.10. $\Sigma \wedge \mathrm{BEC} \models \mathit{HoldsAt}(\mathit{Green}, 5)$.

Proof. From (17.21) by universal instantiation, we have

$$\mathit{Trajectory}(\mathit{On}, 2, \mathit{Green}, 3) \tag{17.26}$$

From (17.23) and BEC1, we have $\neg \mathit{StoppedIn}(2, \mathit{On}, 5)$. From this, $\mathit{Happens}(\mathit{TurnOn}, 2)$ (which follows from (17.23)), $\mathit{Initiates}(\mathit{TurnOn}, \mathit{On}, 2)$ (which follows from (17.17)), $0 < 3$, (17.26), and BEC3, we have $\mathit{HoldsAt}(\mathit{Green}, 5)$. □

Proposition 17.11. $\Sigma \wedge \mathrm{BEC} \models \mathit{HoldsAt}(\mathit{Red}, 7)$.

Table 17.4. EC and DEC predicates (e = event, f, f_1, f_2 = fluents, t, t_1, t_2 = timepoints)

Predicate	Meaning
HoldsAt(f, t)	f is true at t
Happens(e, t)	e occurs at t
ReleasedAt(f, t)	f is released from the commonsense law of inertia at t
Initiates(e, f, t)	if e occurs at t, then f is true and not released from the commonsense law of inertia after t
Terminates(e, f, t)	if e occurs at t, then f is false and not released from the commonsense law of inertia after t
Releases(e, f, t)	if e occurs at t, then f is released from the commonsense law of inertia after t
Trajectory(f_1, t_1, f_2, t_2)	if f_1 is initiated by an event that occurs at t_1, then f_2 is true at $t_1 + t_2$
AntiTrajectory(f_1, t_1, f_2, t_2)	if f_1 is terminated by an event that occurs at t_1, then f_2 is true at $t_1 + t_2$

Proof. From (17.20) by universal instantiation, we have

$$Trajectory(On, 2, Red, 5) \tag{17.27}$$

From (17.23) and BEC1, we have $\neg StoppedIn(2, On, 7)$. From this, $Happens(TurnOn, 2)$ (which follows from (17.23)), $Initiates(TurnOn, On, 2)$ (which follows from (17.17)), $0 < 5$, (17.27), and BEC3, we have $HoldsAt(Red, 7)$. □

17.2.4 Event Calculus (EC)

Miller and Shanahan [65, 66] introduced several alternative formulations of the basic event calculus. A number of their axioms can be combined [70] to produce what we call EC, which differs from the basic event calculus in the following ways:

- It allows negative time. Timepoints are either integers or real numbers.
- It eliminates the *InitiallyN* and *InitiallyP* predicates.
- It explicitly represents that a fluent is released from the commonsense law of inertia using the *ReleasedAt* predicate.
- It adds *AntiTrajectory*.
- It treats *StoppedIn* and *StartedIn* as abbreviations rather than predicates, and introduces other abbreviations.

ReleasedAt(f, t) represents that fluent f is released from the commonsense law of inertia at timepoint t. *AntiTrajectory*(f_1, t_1, f_2, t_2) represents that, if fluent f_1 is terminated by an event that occurs at timepoint t_1, then fluent f_2 will be true at timepoint $t_1 + t_2$.

The predicates of EC are shown in Table 17.4. The axioms and definitions of EC are as follows.

EC1. $Clipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 \leqslant t < t_2 \wedge Terminates(e, f, t))$.

EC2. $Declipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 \leqslant t < t_2 \wedge Initiates(e, f, t))$.

EC3. $StoppedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 < t < t_2 \wedge Terminates(e, f, t))$.

EC4. $StartedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 < t < t_2 \wedge Initiates(e, f, t))$.

EC5. $Happens(e, t_1) \wedge Initiates(e, f_1, t_1) \wedge 0 < t_2 \wedge Trajectory(f_1, t_1, f_2, t_2) \wedge \neg StoppedIn(t_1, f_1, t_1 + t_2) \supset HoldsAt(f_2, t_1 + t_2)$.

EC6. $Happens(e, t_1) \wedge Terminates(e, f_1, t_1) \wedge 0 < t_2 \wedge AntiTrajectory(f_1, t_1, f_2, t_2) \wedge \neg StartedIn(t_1, f_1, t_1 + t_2) \supset HoldsAt(f_2, t_1 + t_2)$.

EC7. $PersistsBetween(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \neg\exists t\ (ReleasedAt(f, t) \wedge t_1 < t \leqslant t_2)$.

EC8. $ReleasedBetween(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 \leqslant t < t_2 \wedge Releases(e, f, t))$.

EC9. $HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge PersistsBetween(t_1, f, t_2) \wedge \neg Clipped(t_1, f, t_2) \supset HoldsAt(f, t_2)$.

EC10. $\neg HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge PersistsBetween(t_1, f, t_2) \wedge \neg Declipped(t_1, f, t_2) \supset \neg HoldsAt(f, t_2)$.

EC11. $ReleasedAt(f, t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1, f, t_2) \wedge \neg Declipped(t_1, f, t_2) \supset ReleasedAt(f, t_2)$.

EC12. $\neg ReleasedAt(f, t_1) \wedge t_1 < t_2 \wedge \neg ReleasedBetween(t_1, f, t_2) \supset \neg ReleasedAt(f, t_2)$.

EC13. $ReleasedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 < t < t_2 \wedge Releases(e, f, t))$.

EC14. $Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StoppedIn(t_1, f, t_2) \wedge \neg ReleasedIn(t_1, f, t_2) \supset HoldsAt(f, t_2)$.

EC15. $Happens(e, t_1) \wedge Terminates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StartedIn(t_1, f, t_2) \wedge \neg ReleasedIn(t_1, f, t_2) \supset \neg HoldsAt(f, t_2)$.

EC16. $Happens(e, t_1) \wedge Releases(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StoppedIn(t_1, f, t_2) \wedge \neg StartedIn(t_1, f, t_2) \supset ReleasedAt(f, t_2)$.

EC17. $Happens(e, t_1) \wedge (Initiates(e, f, t_1) \vee Terminates(e, f, t_1)) \wedge t_1 < t_2 \wedge \neg ReleasedIn(t_1, f, t_2) \supset \neg ReleasedAt(f, t_2)$.

Let EC be the formula generated by conjoining axioms EC5, EC6, EC9, EC10, EC11, EC12, EC14, EC15, EC16, and EC17 and then expanding the predicates *Clipped*, *Declipped*, *StoppedIn*, *StartedIn*, *PersistsBetween*, *ReleasedBetween*, and *ReleasedIn* using definitions EC1, EC2, EC3, EC4, EC7, EC8, and EC13.

Example 17.5. Consider again the light example. We replace (17.15) with the following:

$$\neg HoldsAt(On, 0) \tag{17.28}$$

We add the following:

$$\neg ReleasedAt(f, t) \tag{17.29}$$

Let Σ be the conjunction of (17.10), (17.11), (17.13), (17.14), (17.16), (17.28), and (17.29). Again, we get the same results.

Proposition 17.12. $\Sigma \wedge \text{EC} \models \neg HoldsAt(On, 1)$.

Proof. From (17.13) and EC2, we have $\neg Declipped(0, On, 1)$. From this, (17.28), $0 < 1$, $PersistsBetween(0, On, 1)$ (which follows from (17.29) and EC7), and EC10, we have $\neg HoldsAt(On, 1)$. □

Proposition 17.13. $\Sigma \wedge \text{EC} \models HoldsAt(On, 3)$.

Proof. From (17.13) and EC3, we have $\neg StoppedIn(2, On, 3)$. From this, $Happens(TurnOn, 2)$ (which follows from (17.13)), $Initiates(TurnOn, On, 2)$ (which follows from (17.10)), $2 < 3$, $\neg ReleasedIn(2, On, 3)$ (which follows from (17.13) and EC13), and EC14, we have $HoldsAt(On, 3)$. □

Proposition 17.14. $\Sigma \wedge \text{EC} \models \neg HoldsAt(On, 5)$.

Proof. From (17.13) and EC4, we have $\neg StartedIn(4, On, 5)$. From this, $Happens(TurnOff, 4)$ (which follows from (17.13)), $Terminates(TurnOff, On, 4)$ (which follows from (17.11)), $4 < 5$, $\neg ReleasedIn(4, On, 5)$ (which follows from (17.13) and EC13), and EC15, we have $\neg HoldsAt(On, 5)$. □

17.2.5 Discrete Event Calculus (DEC)

Mueller [70, 74] developed the discrete event calculus (DEC) to improve the efficiency of automated reasoning in the event calculus. DEC improves efficiency by limiting time to the integers, and eliminating triply quantified time from many of the axioms.

The predicates of DEC are the same as those of EC, as shown in Table 17.4. The axioms and definitions of DEC are as follows.

DEC1. $StoppedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 < t < t_2 \wedge Terminates(e, f, t))$.

DEC2. $StartedIn(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists e, t\ (Happens(e, t) \wedge t_1 < t < t_2 \wedge Initiates(e, f, t))$.

DEC3. $Happens(e, t_1) \wedge Initiates(e, f_1, t_1) \wedge 0 < t_2 \wedge Trajectory(f_1, t_1, f_2, t_2) \wedge \neg StoppedIn(t_1, f_1, t_1 + t_2) \supset HoldsAt(f_2, t_1 + t_2)$.

DEC4. $Happens(e, t_1) \wedge Terminates(e, f_1, t_1) \wedge 0 < t_2 \wedge AntiTrajectory(f_1, t_1, f_2, t_2) \wedge \neg StartedIn(t_1, f_1, t_1 + t_2) \supset HoldsAt(f_2, t_1 + t_2)$.

DEC5. $HoldsAt(f, t) \wedge \neg ReleasedAt(f, t + 1) \wedge \neg\exists e\ (Happens(e, t) \wedge Terminates(e, f, t)) \supset HoldsAt(f, t + 1)$.

DEC6. $\neg HoldsAt(f, t) \wedge \neg ReleasedAt(f, t + 1) \wedge \neg\exists e\ (Happens(e, t) \wedge Initiates(e, f, t)) \supset \neg HoldsAt(f, t + 1)$.

DEC7. $ReleasedAt(f, t) \wedge \neg\exists e\ (Happens(e, t) \wedge (Initiates(e, f, t) \vee Terminates(e, f, t))) \supset ReleasedAt(f, t + 1)$.

DEC8. $\neg ReleasedAt(f, t) \wedge \neg\exists e\ (Happens(e, t) \wedge Releases(e, f, t)) \supset \neg ReleasedAt(f, t + 1)$.

DEC9. $Happens(e, t) \wedge Initiates(e, f, t) \supset HoldsAt(f, t + 1)$.

DEC10. $Happens(e, t) \wedge Terminates(e, f, t) \supset \neg HoldsAt(f, t + 1)$.

DEC11. $Happens(e, t) \wedge Releases(e, f, t) \supset ReleasedAt(f, t + 1)$.

DEC12. $Happens(e, t) \wedge (Initiates(e, f, t) \vee Terminates(e, f, t)) \supset \neg ReleasedAt(f, t + 1)$.

Let DEC be the formula generated by conjoining axioms DEC3 through DEC12 and then expanding the predicates *StoppedIn* and *StartedIn* using definitions DEC1 and DEC2.

The difference between EC and DEC is that EC operates over spans of timepoints, whereas DEC operates timepoint by timepoint. For example, EC14 states that a fluent that is initiated remains true until it is terminated or released. This corresponds to several DEC axioms. DEC9 states that a fluent that is initiated is true at the next timepoint. DEC5 states that a fluent that is true, not released from the commonsense law of inertia, and not terminated, is true at the next timepoint. The axioms dealing with *Trajectory* and *AntiTrajectory*, DEC3 and DEC4, are the same as EC5 and EC6. The definitions of *StoppedIn* and *StartedIn*, DEC1 and DEC2, are the same as EC3 and EC4.

Example 17.6. Consider again the light example. Let Σ be as for EC.

Proposition 17.15. $\Sigma \wedge \text{DEC} \models \neg HoldsAt(On, 1)$.

Proof. From (17.13), we have $\neg\exists e\ (Happens(e, 0) \wedge Initiates(e, On, 0))$. From this, (17.28), $\neg ReleasedAt(On, 1)$ (which follows from (17.29)), and DEC6, we have $\neg HoldsAt(On, 1)$. □

Proposition 17.16. $\Sigma \wedge \text{DEC} \models \mathit{HoldsAt}(\mathit{On}, 3)$.

Proof. From $\mathit{Happens}(\mathit{TurnOn}, 2)$ (which follows from (17.13)), $\mathit{Initiates}(\mathit{TurnOn}, \mathit{On}, 2)$ (which follows from (17.10)), and DEC9, we have $\mathit{HoldsAt}(\mathit{On}, 3)$. □

Proposition 17.17. $\Sigma \wedge \text{DEC} \models \neg\mathit{HoldsAt}(\mathit{On}, 5)$.

Proof. From $\mathit{Happens}(\mathit{TurnOff}, 4)$ (which follows from (17.13)), $\mathit{Terminates}(\mathit{TurnOff}, \mathit{On}, 4)$ (which follows from (17.11)), and DEC10, we have $\neg\mathit{HoldsAt}(\mathit{On}, 5)$. □

17.2.6 Equivalence of DEC and EC

We have the following equivalence between DEC and EC.

Proposition 17.18. *If the domain of the timepoint sort is restricted to the integers, then DEC is logically equivalent to EC.*

Proof. (EC $\models$ DEC) By universal instantiation, substituting t_1+1 for t_2. For example, DEC9 is obtained from EC14 via the following chain of equivalences:

$$
\begin{aligned}
&\mathit{Happens}(e, t_1) \wedge \mathit{Initiates}(e, f, t_1) \wedge t_1 < t_1 + 1 \wedge \neg\mathit{StoppedIn}(t_1, f, t_1 + 1) \wedge \\
&\quad \neg\mathit{ReleasedIn}(t_1, f, t_1 + 1) \supset \mathit{HoldsAt}(f, t_1 + 1) \\
&\equiv \\
&\mathit{Happens}(e, t_1) \wedge \mathit{Initiates}(e, f, t_1) \wedge \\
&\quad \neg\exists e, t\,(\mathit{Happens}(e, t) \wedge t_1 < t < t_1 + 1 \wedge \mathit{Terminates}(e, f, t)) \wedge \\
&\quad \neg\exists e, t\,(\mathit{Happens}(e, t) \wedge t_1 < t < t_1 + 1 \wedge \mathit{Releases}(e, f, t)) \supset \\
&\quad \mathit{HoldsAt}(f, t_1 + 1) \\
&\equiv \text{ (\textit{for integer time})} \\
&\mathit{Happens}(e, t_1) \wedge \mathit{Initiates}(e, f, t_1) \supset \mathit{HoldsAt}(f, t_1 + 1)
\end{aligned}
$$

(DEC $\models$ EC) By a series of lemmas showing that each EC axiom follows from DEC. See [70] or [74]. □

17.2.7 Other Versions

Other versions of the event calculus have been developed to support

- causal constraints for instantaneously interacting indirect effects [101].
- continuously changing parameters using differential equations [64, 66].
- events with duration[4] [66, 97, 100].
- hierarchical or compound events [97, 100].

[4] Events with duration may also be represented as fluents that are initiated and terminated by instantaneous events. For example, a moving event with duration can be represented using the axioms $\mathit{Initiates}(\mathit{StartMoving}, \mathit{Moving}, t)$ and $\mathit{Terminates}(\mathit{StopMoving}, \mathit{Moving}, t)$. See also the discussion of continuous change in Section 17.5.7.

17.3 Relationship to other Formalisms

The event calculus is closely related to the situation calculus (see Chapter 16) and temporal action logics (see Chapter 18). The relation between the event calculus and the situation calculus is treated by Kowalski and Sadri [43, 44] and Van Belleghem, Denecker, and De Schreye [112]. The relation between the event calculus and temporal action logics is treated by Mueller [75]. Bennett and Galton [4] define a versatile event logic (VEL) and use it to describe versions of the situation calculus and the event calculus. A problem for future research is the relation of the event calculus and nonmonotonic causal logic.

17.4 Default Reasoning

An axiomatization is *elaboration tolerant* to the degree that it can be extended easily [61]. In the examples given so far, we have fully specified the effects of events and the event occurrences. That is, we have supplied bi-implications for the predicates *Initiates*, *Terminates*, *Releases*, and *Happens*. This is not very elaboration tolerant, because whenever we wish to add event effects and occurrences, we must modify these bi-implications.

Instead, we should be able to write individual axioms specifying what effects particular events have on particular fluents and what events occur. But then we have two problems:

1. how to derive what effects particular events do *not* have on particular fluents, or the *frame problem* [8, 62, 98, 105] (see also Section 16.2), and

2. how to derive what events do *not* occur.

These problems can be solved using any framework for default or nonmonotonic reasoning [5, 7] (see also Chapters 6 and 7). In this section, we discuss the use of circumscription [51, 56, 57, 59] (see Section 6.4) and negation as failure [11].

17.4.1 Circumscription

Consider the light example. Instead of writing the single axiom

$$Happens(e, t) \equiv (e = TurnOn \wedge t = 2) \vee (e = TurnOff \wedge t = 4) \tag{17.30}$$

we write several axioms:

$$Happens(TurnOn, 2) \tag{17.31}$$

$$Happens(TurnOff, 4) \tag{17.32}$$

Then we circumscribe *Happens* in (17.31) $\wedge$ (17.32), which yields (17.30).

Circumscription allows us to assume by default that the events known to occur are the only events that occur. That is, there are no extraneous events. If we allowed extraneous events, then we could no longer prove, say, that the light is off at timepoint 6, because we could no longer prove the absence of events turning on the light between timepoints 4 and 6. If we later add the axiom

$$Happens(TurnOn, 5)$$

then we recompute the circumscription, which allows us to prove that in fact the light is on at timepoint 6.

Similarly, we write separate axioms for *Initiates*, *Terminates*, and *Releases*, and circumscribe these predicates, which allows us to assume by default that the known effects of events are the only effects of events. That is, there are no extraneous event effects. If we allowed extraneous event effects, then we could no longer prove that the light is off at timepoint 6 if some unrelated event occurred between timepoints 4 and 6, because we could no longer prove that the unrelated event does not turn on the light.

17.4.2 Computing Circumscription

It is difficult to compute circumscription in general [16]. The circumscription of a predicate in a formula, which is defined by a formula of second-order logic, does not always reduce to a formula of first-order logic [47]. In many cases, however, we can compute circumscription using the following two propositions. The first proposition asserts that certain circumscriptions reduce to predicate completion. (See Section 7.3.2.)

Proposition 17.19. *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 circumscription CIRC$[\forall x_1, \ldots, x_n\, (\Delta(x_1, \ldots, x_n) \supset \rho(x_1, \ldots, x_n)]; \rho)$ is equivalent to $\forall x_1, \ldots, x_n\, (\Delta(x_1, \ldots, x_n) \equiv \rho(x_1, \ldots, x_n))$.*

Proof. See the proof of Proposition 2 of Lifschitz [51]. (See also [84].) □

This gives us the following method for computing circumscription of ρ in a formula:

1. Rewrite the formula in the form $\forall x_1, \ldots, x_n\, (\Delta(x_1, \ldots, x_n) \supset \rho(x_1, \ldots, x_n))$, where $\Delta(x_1, \ldots, x_n)$ does not contain ρ.
2. Apply Proposition 17.19.

Example 17.7. Let $\Sigma = \mathit{Initiates}(E_1, F_1, t) \wedge \mathit{Initiates}(E_2, F_2, t)$. We compute $\mathit{CIRC}[\Sigma; \mathit{Initiates}]$ by rewriting Σ as

$$(e = E_1 \wedge f = F_1) \vee (e = E_2 \wedge f = F_2) \supset \mathit{Initiates}(e, f, t)$$

and then applying Proposition 17.19, which gives

$$\mathit{Initiates}(e, f, t) \equiv (e = E_1 \wedge f = F_1) \vee (e = E_2 \wedge f = F_2)$$

The second proposition allows us to compute the circumscription of several predicates, called parallel circumscription. We start with a definition.

Definition 17.1. *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 $\supset$ and $\equiv$ from Δ.*

Proposition 17.20. *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 $\bigwedge_{i=1}^{n} CIRC[\Delta; \rho_i]$.*

Proof. See the proof of Proposition 14 of Lifschitz [51]. □

Further methods for computing circumscription are discussed in Section 6.4.4.

Example 17.8. Let Σ be the conjunction of the following axioms:

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

$$Terminates(TurnOff, On, t)$$

Let Δ be the conjunction of the following axioms:

$$Happens(TurnOn, 2)$$

$$Happens(TurnOff, 4)$$

Let Γ be the conjunction of (17.14), (17.28), and (17.29). We can use circumscription to prove that the light is on at timepoint 3.

Proposition 17.21.

$$CIRC[\Sigma; Initiates, Terminates, Releases] \wedge CIRC[\Delta; Happens] \wedge \Gamma \wedge \text{EC} \models HoldsAt(On, 3)$$

Proof. From $CIRC[\Sigma; Initiates, Terminates, Releases]$, Propositions 17.20 and 17.19, we have

$$\begin{aligned}&(Initiates(e, f, t) \equiv (e = TurnOn \wedge f = On)) \wedge \\ &(Terminates(e, f, t) \equiv (e = TurnOff \wedge f = On)) \wedge \\ &\neg Releases(e, f, t)\end{aligned} \tag{17.33}$$

From $CIRC[\Delta; Happens]$ and Proposition 17.19, we have

$$Happens(e, t) \equiv (e = TurnOn \wedge t = 2) \vee (e = TurnOff \wedge t = 4) \tag{17.34}$$

From this and EC3, we have $\neg StoppedIn(2, On, 3)$. From this, $Happens(TurnOn, 2)$ (which follows from (17.34)), $Initiates(TurnOn, On, 2)$ (which follows from (17.33)), $2 < 3$, $\neg ReleasedIn(2, On, 3)$ (which follows from (17.34) and EC13), and EC14, we have $HoldsAt(On, 3)$. □

17.4.3 Historical Note

Notice that we keep the event calculus axioms EC outside the scope of any circumscription. This technique, known as *filtering*, was introduced in the features and fluents framework [14, 15, 89, 90] (see also Chapter 18) and incorporated into the event calculus by Shanahan [96, 98]. The need for filtering became clear after Hanks and McDermott [32] introduced the Yale shooting scenario, which exposed problems with

simply circumscribing the entire situation calculus axiomatization of the scenario. Shanahan [98] describes treatments of the Yale shooting scenario in the situation calculus and the event calculus; Shanahan [105] and Lifschitz [52] provide a modern perspective.

17.4.4 Negation as Failure

Instead of using circumscription for default reasoning in the event calculus, logic programming with the negation as failure operator **not** may be used [41, 45, 93, 94]. For example, we write rules such as the following:

$$\begin{aligned} &\mathit{clipped}(T1, F, T2) \leftarrow \mathit{happens}(E, T),\ T1 <= T,\ T < T2, \\ &\quad \mathit{terminates}(E, F, T). \\ &\mathit{holds\_at}(F, T2) \leftarrow \mathit{holds\_at}(F, T1),\ T1 < T2,\ \textbf{not}\ \mathit{clipped}(T1, F, T2). \end{aligned}$$

These rules are similar to axioms EC1 and EC9. Then we add rules such as the following to our domain description:

$$\begin{aligned} &\mathit{initiates}(\mathit{turn\_on}, \mathit{on}, T). \\ &\mathit{terminates}(\mathit{turn\_off}, \mathit{on}, T). \\ &\mathit{happens}(\mathit{turn\_on}, 2). \\ &\mathit{happens}(\mathit{turn\_off}, 4). \end{aligned}$$

Mueller [73] provides complete lists of event calculus rules for use with answer set solvers [3, 79] along with sample domain descriptions.

17.5 Event Calculus Knowledge Representation

This section describes methods for representing knowledge using the event calculus. These methods can be used with BEC, EC, and DEC. Those that do not involve trajectories or release from the commonsense law of inertia can also be used with SEC.

17.5.1 Parameters

We represent events and fluents with parameters as functions that return event and fluent terms. For example, we may represent the event of person p turning on light l using a function $\mathit{TurnOn}(p, l)$, and the property that light l is turned on using a function $\mathit{On}(l)$. We then require the following unique names axioms:

$$\mathit{TurnOn}(p_1, l_1) = \mathit{TurnOn}(p_2, l_2) \supset p_1 = p_2 \wedge l_1 = l_2 \tag{17.35}$$

$$\mathit{On}(l_1) = \mathit{On}(l_2) \supset l_1 = l_2 \tag{17.36}$$

If we have another event $\mathit{TurnOff}(p, l)$, then we also require the unique names axiom:

$$\mathit{TurnOn}(p_1, l_1) \neq \mathit{TurnOff}(p_2, l_2) \tag{17.37}$$

The U notation [48] is convenient for defining unique names axioms. 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_p is the sort of the pth argument position of ϕ_i and the sort of y_p is the sort of the pth argument position of ϕ_j, for each $1 \leqslant i < j \leqslant k$, and the conjunction of the formulas

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

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_p and y_p is the sort of the pth argument position of ϕ_i, for each $1 \leqslant i \leqslant k$.

We may then use this notation to replace (17.35), (17.36), and (17.37) with

$$U[\mathit{TurnOn}, \mathit{TurnOff}] \wedge U[\mathit{On}]$$

In the remainder of this section, we assume that appropriate unique names axioms are defined.

17.5.2 Event Effects

The effects of events are represented using *effect axioms*, which are of the form

$$\gamma \supset \mathit{Initiates}(\alpha, \beta, \tau), \quad \text{or}$$
$$\gamma \supset \mathit{Terminates}(\alpha, \beta, \tau)$$

where γ is a condition, α is an event, β is a fluent, and τ is a timepoint. A condition is a formula containing atoms of the form $\mathit{HoldsAt}(\beta, \tau)$ and $\neg\mathit{HoldsAt}(\beta, \tau)$, where β is a fluent and τ is a timepoint.

Example 17.9. Consider a counter that can be incremented and reset. The fluent $\mathit{Value}(c, v)$ represents that counter c has value v. The event $\mathit{Increment}(c)$ represents that counter c is incremented, and the event $\mathit{Reset}(c)$ represents that the counter c is reset. We use two effect axioms to represent that, if the value of a counter is v and the counter is incremented, its value will be $v + 1$ and will no longer be v:

$$\mathit{HoldsAt}(\mathit{Value}(c, v), t) \supset \mathit{Initiates}(\mathit{Increment}(c), \mathit{Value}(c, v + 1), t) \quad (17.38)$$
$$\mathit{HoldsAt}(\mathit{Value}(c, v), t) \supset \mathit{Terminates}(\mathit{Increment}(c), \mathit{Value}(c, v), t) \quad (17.39)$$

We use two more effect axioms to represent that, if the value of a counter is v and the counter is reset, its value will be 0 and will no longer be v:

$$\mathit{Initiates}(\mathit{Reset}(c), \mathit{Value}(c, 0), t) \quad (17.40)$$
$$\mathit{HoldsAt}(\mathit{Value}(c, v), t) \wedge c \neq 0 \supset \mathit{Terminates}(\mathit{Reset}(c), \mathit{Value}(c, v), t) \quad (17.41)$$

The effect of an event can depend on the context in which it occurs. The condition γ represents the context. In the example of the counter, the effect of incrementing the counter depends on its current value.

17.5.3 Preconditions

We might represent the effect of turning on a device as follows:

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

But there are many things that could prevent a device from going on. It could be unplugged or broken, its on-off switch could be broken, and so on. A *qualification* is a condition that prevents an event from having its intended effects. The *qualification problem* is the problem of representing and reasoning about qualifications.

A partial solution to the qualification problem is to use *preconditions*. The condition γ of effect axioms can be used to represent preconditions.

Example 17.10. If a person turns on a device, then, provided the device is not broken, the device will be on:

$$\neg HoldsAt(Broken(d), t) \supset Initiates(TurnOn(p, d), On(d), t) \tag{17.42}$$

But this is not elaboration tolerant, because whenever we wish to add qualifications, we must modify (17.42). Instead we can use default reasoning.

Example 17.11. Instead of writing (17.42), we write

$$\neg Ab_1(d, t) \supset Initiates(TurnOn(p, d), On(d), t) \tag{17.43}$$

$Ab_1(d, t)$ is an *abnormality predicate* [28, 58–60]. It represents that at timepoint t, device d is abnormal in some way that prevents it from being turned on. In general, we use a distinct abnormality predicate for each type of abnormality. We then add qualifications by writing *cancellation axioms* [23, 51, 59]:

$$HoldsAt(Broken(d), t) \supset Ab_1(d, t) \tag{17.44}$$

$$\neg HoldsAt(PluggedIn(d), t) \supset Ab_1(d, t) \tag{17.45}$$

We then circumscribe the abnormality predicate Ab_1 in the conjunction of cancellation axioms (17.44) and (17.45), which yields

$$Ab_1(d, t) \equiv HoldsAt(Broken(d), t) \vee \neg HoldsAt(PluggedIn(d), t) \tag{17.46}$$

We then reason using (17.43) and (17.46). Whenever we wish to add additional qualifications, we add cancellation axioms and recompute the circumscription of the abnormality predicates in the cancellation axioms.

17.5.4 State Constraints

Law-like relationships among properties are represented using *state constraint*, which are formulas containing atoms of the form $HoldsAt(\beta, \tau)$ and $\neg HoldsAt(\beta, \tau)$, where β is a fluent and τ is a timepoint.

For example, we may use two state constraints to represent that a counter has exactly one value at a time:

$$\exists v\ HoldsAt(Value(c, v), t) \tag{17.47}$$

$$HoldsAt(Value(c, v_1), t) \wedge HoldsAt(Value(c, v_2), t) \supset v_1 = v_2 \tag{17.48}$$

17.5.5 Concurrent Events

In the event calculus, events may occur concurrently. That is, we may have $Happens(e_1, t_1)$ and $Happens(e_2, t_2)$ where $e_1 \neq e_2$ and $t_1 = t_2$. We represent the effects of concurrent events using effect axioms whose conditions contain atoms of the form $Happens(\alpha, \tau)$ and $\neg Happens(\alpha, \tau)$, where α is an event and τ is a timepoint [66].

Example 17.12. Consider again the example of the counter. Suppose that the value of a counter C is 5 at timepoint 0:

$$HoldsAt(Value(C, 5), 0) \tag{17.49}$$

Further suppose that the counter is simultaneously incremented and reset at timepoint 1:

$$Happens(Increment(C), 1) \tag{17.50}$$

$$Happens(Reset(C), 1) \tag{17.51}$$

(17.50) leads us to conclude

$$HoldsAt(Value(C, 6), 2)$$

whereas (17.51) leads us to conclude

$$HoldsAt(Value(C, 0), 2)$$

These formulas contradict the state constraint (17.48).

In order to deal with this, we may specify exactly what happens when a counter is simultaneously incremented and reset. There are a number of possibilities.

One possibility is that nothing happens. We replace the effect axioms (17.38), (17.39), (17.40), and (17.41) with the following effect axioms:

$$\neg Happens(Reset(c), t) \wedge HoldsAt(Value(c, v), t) \supset Initiates(Increment(c), Value(c, v + 1), t) \tag{17.52}$$

$$\neg Happens(Reset(c), t) \wedge HoldsAt(Value(c, v), t) \supset Terminates(Increment(c), Value(c, v), t) \tag{17.53}$$

$$\neg Happens(Increment(c), t) \supset Initiates(Reset(c), Value(c, 0), t) \tag{17.54}$$

$$\neg Happens(Increment(c), t) \wedge HoldsAt(Value(c, v), t) \wedge c \neq 0 \supset Terminates(Reset(c), Value(c, v), t) \tag{17.55}$$

Another possibility is that the counter is neither incremented nor reset, but that the counter enters an error state. We use the effect axioms (17.52), (17.53), (17.54), and (17.55), and a further effect axiom that represents that, if a counter is simultaneously

reset and incremented, it will be in an error state:

$$Happens(Reset(c), t) \supset$$
$$\quad Initiates(Increment(c), Error(c), t)$$

We could also have written this as

$$Happens(Increment(c), t) \supset$$
$$\quad Initiates(Reset(c), Error(c), t)$$

Another possibility is that, if a counter is simultaneously reset and incremented, the incrementing takes priority and the counter is incremented:

$$HoldsAt(Value(c, v), t) \supset$$
$$\quad Initiates(Increment(c), Value(c, v + 1), t)$$

$$HoldsAt(Value(c, v), t) \supset$$
$$\quad Terminates(Increment(c), Value(c, v), t)$$

$$\neg Happens(Increment(c), t) \supset$$
$$\quad Initiates(Reset(c), Value(c, 0), t)$$

$$\neg Happens(Increment(c), t) \wedge HoldsAt(Value(c, v), t) \wedge c \neq 0 \supset$$
$$\quad Terminates(Reset(c), Value(c, v), t)$$

Similarly, we could represent that, if a counter is simultaneously reset and incremented, the resetting takes priority and the counter is reset.

17.5.6 Triggered Events

Events that are triggered under certain circumstances are represented using *trigger axioms* [94, 96, 98], which are of the form

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

where γ is a condition, α is an event, and τ is a timepoint.

Example 17.13. Consider a thermostat that turns on a heater when the temperature drops below A, and turns off the heater when the temperature rises above B. We represent this using two effect axioms and two trigger axioms:

$$Initiates(TurnOn, On, t)$$
$$Terminates(TurnOff, On, t)$$
$$HoldsAt(Temperature(v), t) \wedge v < A \wedge \neg HoldsAt(On, t) \supset$$
$$\quad Happens(TurnOn, t)$$
$$HoldsAt(Temperature(v), t) \wedge v > B \wedge HoldsAt(On, t) \supset$$
$$\quad Happens(TurnOff, t)$$

The conditions $\neg HoldsAt(On, t)$ and $HoldsAt(On, t)$ are required to prevent *TurnOn* and *TurnOff* from repeatedly triggering.

17.5.7 Continuous Change

Examples of continuous change include falling objects, expanding balloons, and containers being filled. Continuous change is represented using *trajectory axioms* [94, 95, 98], which are of the form

$$\gamma \supset Trajectory(\beta_1, \tau_1, \beta_2, \tau_2), \quad \text{or}$$
$$\gamma \supset AntiTrajectory(\beta_1, \tau_1, \beta_2, \tau_2)$$

where γ is a condition, β_1 and β_2 are fluents, and τ_1 and τ_2 are timepoints. *Trajectory* is used to determine the truth value of fluent β_2 after fluent β_1 is initiated, until β_1 is terminated. *AntiTrajectory* is used to determine the truth value of fluent β_2 after fluent β_1 is terminated, until β_1 is initiated.

Although DEC does not support continuous time, we may still use *Trajectory* and *AntiTrajectory* in DEC to represent gradual change. Gradual change is a discrete approximation to continuous change in which the value of a changing fluent is only represented for integer timepoints.

Example 17.14. Consider a falling object. We use effect axioms to represent that, if a person drops an object, then it will be falling, and if an object hits the ground, then it will no longer be falling:

$$Initiates(Drop(p, o), Falling(o), t) \tag{17.56}$$
$$Terminates(HitGround(o), Falling(o), t) \tag{17.57}$$

We represent that, if a person drops an object, then its height will be released from the commonsense law of inertia:

$$Releases(Drop(p, o), Height(o, h), t) \tag{17.58}$$

(For an object o, $Height(o, h)$ is released for all h.) We use a trajectory axiom to represent that the height of the object is given by an equation of free-fall motion, where G is the acceleration due to gravity (9.8 m/sec^2):

$$HoldsAt(Height(o, h), t_1) \supset$$
$$Trajectory(Falling(o), t_1, Height(o, h - \tfrac{1}{2}Gt_2^2), t_2) \tag{17.59}$$

We use a trigger axiom to represent that, when an object is falling and its height is 0, it hits the ground:

$$HoldsAt(Falling(o), t) \wedge HoldsAt(Height(o, 0), t) \supset$$
$$Happens(HitGround(o), t) \tag{17.60}$$

We specify that, if an object hits the ground and its height is h, then its height will be h and its height will no longer be released from the commonsense law of inertia:

$$HoldsAt(Height(o, h), t) \supset Initiates(HitGround(o), Height(o, h), t) \tag{17.61}$$

We specify that an object has a unique height:

$$HoldsAt(Height(o, h_1), t) \wedge HoldsAt(Height(o, h_2), t) \supset h_1 = h_2 \tag{17.62}$$

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

$$\neg HoldsAt(Falling(Apple), 0) \tag{17.63}$$

$$HoldsAt(Height(Apple, G/2), 0) \tag{17.64}$$

$$Happens(Drop(Nathan, Apple), 0) \tag{17.65}$$

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

Proposition 17.22. *Let* $\Sigma =$ (17.56) $\wedge$ (17.57) $\wedge$ (17.58) $\wedge$ (17.61), $\Delta =$ (17.60) $\wedge$ (17.65), $\Omega = U[Drop, HitGround] \wedge U[Falling, Height]$, $\Gamma =$ (17.59) $\wedge$ (17.62) $\wedge$ (17.63) $\wedge$ (17.64). *Then we have*

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

Proof. See the proofs of Propositions 7.2 and 7.3 of Mueller [74]. □

17.5.8 Nondeterministic Effects

Nondeterministic effects of events can be represented in the event calculus using *determining fluents* [98], or fluents released from the commonsense law of inertia that are used within the conditions of effect axioms.

Example 17.15. Consider the example of rolling a die with six sides. We define a determining fluent $DieDF(d, s)$ which represents that die d will land on side s. This fluent is released from the commonsense law of inertia. In EC and DEC, we require the axiom

$$ReleasedAt(DieDF(d, s), t)$$

In BEC, a fluent that is never initiated or terminated and is neither *InitiallyN* nor *InitiallyP* is released from the commonsense law of inertia, so no further axioms are required to released *DieDF* from the commonsense law of inertia.

We use state constraints to represent that, at any timepoint, $DieDF(d, s)$ assigns one of the sides $\{1, \ldots, 6\}$ to a die:

$$\begin{aligned}
&\exists s\ HoldsAt(DieDF(d, s), t) \\
&HoldsAt(DieDF(d, s_1), t) \wedge HoldsAt(DieDF(d, s_2), t) \supset s_1 = s_2 \\
&HoldsAt(DieDF(d, s), t) \supset s = 1 \vee s = 2 \vee s = 3 \vee s = 4 \vee s = 5 \vee s = 6
\end{aligned}$$

We use effect axioms to represent that, if a die is rolled at a timepoint, it will land on the side assigned to the die by *DieDF* at that timepoint:

$$HoldsAt(DieDF(d, s), t) \supset Initiates(Roll(d), Side(d, s), t)$$

$$HoldsAt(Side(d, s_1), t) \wedge HoldsAt(DieDF(d, s_2), t) \wedge s_1 \neq s_2 \supset Terminates(Roll(d), Side(d, s_1), t)$$

Suppose a die D is rolled at timepoint 0:

$$Happens(Roll(D), 0)$$

What side of the die faces up at timepoint 1? Because *DieDF* is free to take on any of six values at timepoint 0, we get six classes of models: one in which $HoldsAt(DieDF(D, 1), 0)$ and therefore $HoldsAt(Side(D, 1), 1)$, one in which $HoldsAt(DieDF(D, 2), 0)$ and therefore $HoldsAt(Side(D, 2), 1)$, and so on.

17.5.9 Indirect Effects

Suppose that a person and a book are in the living room of a house. When the person walks out of the living room, the book will normally remain in the living room. But if the person is holding the book and walks out of the living room, then the book will no longer be in the living room. That is, an *indirect effect* or *ramification* of the person walking out of the living room is that the book the person is holding changes location. The *ramification problem* [19, 24, 101] is the problem of representing and reasoning about the indirect effects of events. Much research has been performed on the ramification problem [2, 19, 24, 29, 30, 35, 48, 50, 53–55, 85, 91, 101, 111]. Several methods can be used for solving this problem in the event calculus.

Example 17.16 (State constraints). Consider again the example of a light. We represent the direct effect of turning on a light using an effect axiom:

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

We may use a state constraint to represent the indirect effect of turning on the light, namely that the light is not off:

$$\neg HoldsAt(Off(l), t) \equiv HoldsAt(On(l), t)$$

The fluent $Off(l)$ must be released from the commonsense law of inertia. In EC and DEC, we require the axiom

$$ReleasedAt(Off(l), t)$$

This method of representing indirect effects works if it is possible to divide fluents into primitive and derived fluents [39, 48, 50, 101]. Here *On* is primitive and *Off* is derived. The direct effects of events on primitive fluents are represented using effect axioms, whereas the indirect effects of events on derived fluents are represented using state constraints.

Example 17.17 (Release from inertia and state constraints). Suppose that we wish to represent the indirect effects of walking while holding an object, namely that the object moves along with the person holding it. We create a simple axiomatization of space. We start by representing the direct effects of walking. If a person walks from location l_1 to location l_2, then the person will be at l_2 and will no longer be at l_1:

$$Initiates(Walk(p, l_1, l_2), At(p, l_2), t)$$

$$l_1 \neq l_2 \supset Terminates(Walk(p, l_1, l_2), At(p, l_1), t)$$

We also represent the direct effects of picking up and setting down an object. If a person and an object are at the same location and the person picks up the object, then the person will be holding the object:

$$HoldsAt(At(p, l), t) \wedge HoldsAt(At(o, l), t) \supset$$
$$Initiates(PickUp(p, o), Holding(p, o), t)$$

If a person sets down an object, then the person will no longer be holding it:

$$Terminates(SetDown(p, o), Holding(p, o), t)$$

We then represent the indirect effects of walking with a *Releases* axiom, a state constraint, and an effect axiom. If a person and an object are at the same location and the person picks up the object, then the object's location will be released from the commonsense law of inertia:

$$HoldsAt(At(p, l), t) \wedge HoldsAt(At(o, l), t) \supset$$
$$Releases(PickUp(p, o), At(o, l'), t) \tag{17.66}$$

(For any given object o, $At(o, l')$ is released for all l'.) If a person who is holding an object is located at l, then the object is also located at l:

$$HoldsAt(Holding(p, o), t) \wedge HoldsAt(At(p, l), t) \supset HoldsAt(At(o, l), t) \tag{17.67}$$

If a person is holding an object, the person is located at l, and the person sets down the object, then the object will be located at l and the object's location will no longer be released from the commonsense law of inertia:

$$HoldsAt(Holding(p, o), t) \wedge HoldsAt(At(p, l), t) \supset$$
$$Initiates(SetDown(p, o), At(o, l), t) \tag{17.68}$$

Example 17.18 *(Effect axioms).* Another way of representing indirect effects is simply to add more effect axioms. We replace (17.66), (17.67), and (17.68) with effect axioms that state that, if a person who is holding an object walks from location l_1 to location l_2, then the object will be at location l_2 and will no longer be at l_1:

$$HoldsAt(Holding(p, o), t) \supset Initiates(Walk(p, l_1, l_2), At(o, l_2), t)$$

$$HoldsAt(Holding(p, o), t) \wedge l_1 \neq l_2 \supset$$
$$Terminates(Walk(p, l_1, l_2), At(o, l_1), t)$$

Example 17.19 (Effect constraints). Another way of representing indirect effects is to use *effect constraints* [98, 101], which are of the form

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

where γ is a condition, π_1 and π_2 are *Initiates* or *Terminates*, α is an event variable, β_1 and β_2 are fluents, and τ is a timepoint. We use effect constraints to represent that an object moves along with the person holding it:

$$HoldsAt(Holding(p, o), t) \wedge Initiates(e, At(p, l), t) \supset Initiates(e, At(o, l), t)$$

$$HoldsAt(Holding(p, o), t) \wedge Terminates(e, At(p, l), t) \supset$$

$$Terminates(e, At(o, l), t)$$

The event calculus can also be extended to deal with instantaneously interacting indirect effects [101].

The aforementioned methods for dealing with ramifications have various advantages and disadvantages. The method of state constraints is simple, but it requires a clear separation of fluents into those directly affected by events (primitive fluents) and those indirectly affected by events (derived fluents).

The method of releasing a fluent from the commonsense law of inertia allows a fluent to be primitive at some timepoints and derived at other timepoints. But then more bookkeeping is required. We must release the fluent from the commonsense law of inertia, and later make the fluent again subject to this law.

The method of using effect axioms is also simple, but it is less elaboration tolerant. In our example, if we add another way for a person to change location, such as running, we must also add axioms for the indirect effects of running:

$$HoldsAt(Holding(p, o), t) \supset Initiates(Run(p, l_1, l_2), At(o, l_2), t)$$

$$HoldsAt(Holding(p, o), t) \wedge l_1 \neq l_2 \supset Terminates(Run(p, l_1, l_2), At(o, l_1), t)$$

The method of using effect constraints is the most elaboration tolerant. But we cannot apply Proposition 17.19 in order to compute the circumscription of *Initiates* and *Terminates* in effect constraints.

17.5.10 Partially Ordered Events

We may represent partially ordered events using inequalities involving timepoints. For example, we may represent that John picked up a pen and a pad in some unspecified order, and then walked from the office to the living room as follows:

$$Happens(PickUp(John, Pen), T_1)$$

$$Happens(PickUp(John, Pad), T_2)$$

$$Happens(Walk(John, Office, LivingRoom), T_3)$$

$$T_1 < T_3$$

$$T_2 < T_3$$

Using the simple axiomatization of space of Example 17.17 in Section 17.5.9, we can conclude that John was holding both the pen and the pad at T_3, and that the pen and

the pad are both in the living room after T_3. But we cannot conclude that John was holding the pad when he picked up the pen, or that John was holding the pen when he picked up the pad. There are three classes of models:

1. those in which John picks up the pen and then the pad ($T_1 < T_2$),
2. those in which John picks up the pad and then the pen ($T_2 < T_1$), and
3. those in which John picks up the pen and pad simultaneously ($T_1 = T_2$).

17.6 Action Language $\mathcal{E}$

Instead of using classical logic for reasoning about action and change, specialized action languages [22, 25, 26, 81] can be used. The $\mathcal{E}$ action language introduced by Antonis C. Kakas and Rob Miller [35, 36] is closely related to the event calculus.

A language of $\mathcal{E}$ is specified by a set of fluents, a set of events, a set of timepoints, and a partial order on the set of timepoints. An $\mathcal{E}$ domain description consists of a set of statements, which are defined as follows.

Definition 17.2. *If β is a fluent, then β and $\neg\beta$ are fluent literals.*

Definition 17.3. *If γ is a fluent literal and τ is a timepoint, then*

γ **holds-at** τ

is a statement.

Definition 17.4. *If α is an event and τ is a timepoint, then*

α **happens-at** τ

is a statement.

Definition 17.5. *If α is an event, β is a fluent, and Γ is a set of fluent literals, then*

α **initiates** β **when** Γ

and

α **terminates** β **when** Γ

are statements.

The notation α **initiates** β is an abbreviation for α **initiates** β **when** $\emptyset$, and the notation α **terminates** β is an abbreviation for α **terminates** β **when** $\emptyset$.

Example 17.20. We represent the example of turning on and off a light using the following $\mathcal{E}$ domain description:

TurnOn **initiates** *On*

TurnOff **terminates** *On*

$\neg$*On* **holds-at** 0

TurnOn **happens-at** 2

TurnOff **happens-at** 4

This domain description entails the following:

$\neg$*On* **holds-at** 1

On **holds-at** 3

$\neg$*On* **holds-at** 5

Kakas and Miller [35, 36] specify the semantics of $\mathcal{E}$ using simple definitions of structures and models. Miller and Shanahan [66] show that $\mathcal{E}$ corresponds to the EC of Section 17.2.4 without the predicates *ReleasedAt*, *Releases*, *Trajectory*, and *AntiTrajectory*. They define conditions under which an $\mathcal{E}$ domain description matches an EC domain description and prove that, if an $\mathcal{E}$ domain description matches an EC domain description, the domain descriptions entail the same fluent truth values. Dimopoulos, Kakas, and Michael [13] give a translation of $\mathcal{E}$ domain descriptions into answer set programs [3, 20, 21] (see also Chapter 7).

An $\mathcal{E}$ domain description can be translated into an EC or DEC domain description as follows. We assume that the timepoints are the integers and the partial order is $\leqslant$. We divide the $\mathcal{E}$ domain description into sets of **holds-at**, **happens-at**, **initiates**, and **terminates** statements. We translate each **holds-at** statement

$[\neg]\beta$ **holds-at** τ

into the formula

$$[\neg]HoldsAt(\beta, \tau)$$

We translate the set of **happens-at** statements

α_1 **happens-at** τ_1

$\vdots$

α_n **happens-at** τ_n

into the formula

$$Happens(e, t) \equiv (e = \alpha_1 \wedge t = \tau_1) \vee \cdots \vee (e = \alpha_n \wedge t = \tau_n)$$

We translate the set of **initiates/terminates** statements

α_1 **initiates/terminates** β_1 **when** $[\neg]\gamma_{1,1}, \ldots, [\neg]\gamma_{1,p}$

$\vdots$

α_n **initiates/terminates** β_n **when** $[\neg]\gamma_{n,1}, \ldots, [\neg]\gamma_{n,q}$

Table 17.5. Online resources for automated event calculus reasoning

Event calculus planner [103, 104] http://www.iis.ee.ic.ac.uk/~mpsha/planners.html
Event calculus answer set programming [73] http://www.signiform.com/csr/ecas/ (event calculus rules) http://www.tcs.hut.fi/Software/smodels/ (solver)
Discrete Event Calculus Reasoner [70, 71] http://decreasoner.sourceforge.net
TPTP problem library [110] http://www.cs.miami.edu/~tptp/ (see CSR problem domain)
$\mathcal{E}$-RES [37, 38] http://www2.cs.ucy.ac.cy/~pslogic/

into the formula

$$\begin{aligned}&\mathit{Initiates}/\mathit{Terminates}(e, f, t) \equiv\\ &(e = \alpha_1 \wedge f = \beta_1 \wedge [\neg]\mathit{HoldsAt}(\gamma_{1,1}, t) \wedge \cdots \wedge [\neg]\mathit{HoldsAt}(\gamma_{1,p}, t)) \vee \cdots \vee\\ &(e = \alpha_n \wedge f = \beta_n \wedge [\neg]\mathit{HoldsAt}(\gamma_{n,1}, t) \wedge \cdots \wedge [\neg]\mathit{HoldsAt}(\gamma_{n,q}, t))\end{aligned}$$

An extension to $\mathcal{E}$ [35] provides support for indirect effects. The statement

γ **whenever** Γ

where γ is a fluent literal and Γ is a set of fluent literals represents that (1) γ holds at every timepoint at which Γ holds, and (2) every event occurrence that brings about Γ also brings about γ. The language $\mathcal{E}$ has been further developed into the language $\mathcal{M}$odular-$\mathcal{E}$ [34], which addresses the ramification and qualification problems along with the issues of elaboration tolerance and modularity.

17.7 Automated Event Calculus Reasoning

A number of techniques can be used to perform automated reasoning in the event calculus, including logic programming in Prolog, answer set programming, satisfiability solving, and first-order logic automated theorem proving. Table 17.5 provides pointers to online resources for event calculus reasoning.

17.7.1 Prolog

The original event calculus was formulated as a logic program, and logic programming in Prolog can be used to perform event calculus reasoning. If Prolog is used, however, special care must be taken to avoid infinite loops [10, 45, 87, 93, 94, 98, 103]. Event calculus reasoning can be performed through abductive logic programming [6, 12, 17, 103].

17.7.2 Answer Set Programming

Answer set solvers [3] such as smodels [79] can be used to solve event calculus deduction problems [73]. Answer set solvers can also be used for reasoning in the $\mathcal{E}$ language [13].

17.7.3 Satisfiability (SAT) Solving

As a result of the growth in the capabilities of propositional satisfiability (SAT) solvers [92], several event calculus reasoning programs have been built that exploit off-the-shelf SAT solvers. The program of Shanahan and Witkowski [109] solves planning problems using SAT solvers. The Discrete Event Calculus Reasoner [70, 71] uses SAT solvers to perform various types of event calculus reasoning including deduction, abduction, postdiction, and model finding. The $\mathcal{E}$-RES program [37, 38] for solving $\mathcal{E}$ reasoning problems uses SAT solvers to generate classical models of state constraints.

The Discrete Event Calculus Reasoner uses several techniques to reduce the size of the SAT encoding of event calculus problems [70]:

1. The domains of arguments to predicates are restricted by using many-sorted logic.
2. Atom definitions are expanded [80, p. 361] in order to eliminate a large number of *Initiates*, *Terminates*, *Releases*, *Trajectory*, and *AntiTrajectory* ground atoms.
3. Triply quantified time is eliminated from most event calculus axioms by using DEC [70].
4. A compact conjunctive normal form is computed using the technique of renaming subformulas [27, 80, 83].

The Discrete Event Calculus Reasoner distribution includes a library of 99 event calculus reasoning problems that can be solved using the program.

17.7.4 First-Order Logic Automated Theorem Proving

Although first-order logic entailment is undecidable, first-order logic automated theorem proving (ATP) systems [86] have been applied successfully to event calculus deduction problems [77, 78]. But in some cases, the systems require human guidance in the form of lemmas. Event calculus problems are included in the TPTP problem library [110] along with the results of running ATP systems on them.

17.8 Applications of the Event Calculus

An important area of application of the event calculus is commonsense reasoning [74]. Event calculus formalizations have been developed for a number of commonsense domains, including beliefs [46], egg cracking [68, 99, 106], emotions [74], goals and plans [74], object identity [74], space [68, 96], and the zoo world [1, 33, 71]. The event calculus has also been used to model electronic circuits [101] and water tanks [64].

The event calculus can be applied to problems in high-level cognition including natural language understanding and vision. It has been used to build models of story

events and states in space and time [69, 72, 76], represent the semantics of natural language tense and aspect [113], and represent event occurrences in stories [31]. The event calculus has been used to implement the higher-level vision component of an upper-torso humanoid robot [102, 107, 108].

Another application area of the event calculus is business systems. The event calculus has been used to track the state of contracts for performance monitoring [18], to model workflows [10, 114], and to improve the flexibility of applications that use electronic payment systems [115]. Other applications of the event calculus include database updates [41], planning [12, 17, 67, 97, 103, 109], and representing legislation [42].

Bibliography

[1] V. Akman, S.T. Erdogan, J. Lee, V. Lifschitz, and H. Turner. Representing the zoo world and the traffic world in the language of the causal calculator. *Artificial Intelligence*, 153:105–140, 2004.

[2] A.B. Baker. Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence*, 49(1–3):5–23, 1991.

[3] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, 2003.

[4] B. Bennett and A.P. Galton. A unifying semantics for time and events. *Artificial Intelligence*, 153(1–2):13–48, 2004.

[5] D.G. Bobrow. Editor's preface. *Artificial Intelligence*, 13(1–2), 1980 (Special issue on Non-monotonic Reasoning).

[6] A. Bracciali and A.C. Kakas. Frame consistency: Computing with causal explanations. In J.P. Delgrande and T. Schaub, editors, *Proceedings of the Tenth International Workshop on Non-Monotonic Reasoning*, pages 79–87. Whistler, Canada, 2004.

[7] G. Brewka, J. Dix, and K. Konolige. *Nonmonotonic Reasoning: An Overview*. CSLI, Stanford, CA, 1997.

[8] F.M. Brown, editor. *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, Los Altos, CA, 1987. Morgan Kaufmann.

[9] I. Cervesato, M. Franceschet, and A. Montanari. A guided tour through some extensions of the event calculus. *Computational Intelligence*, 16(2):307–347, 2000.

[10] N.K. Cicekli and Y. Yildirim. Formalizing workflows using the event calculus. In M.T. Ibrahim, J. Küng, and N. Revell, editors. *Database and Expert Systems Applications*, *Lecture Notes in Computer Science*, vol. 1873, pages 222–231. Springer, Berlin, 2000.

[11] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors. *Logic and Data Bases*, pages 293–322. Plenum, New York, 1978.

[12] M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In B. Neumann, editor, *Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 384–388, Chichester, UK, 1992. John Wiley.

[13] Y. Dimopoulos, A.C. Kakas, and L. Michael. Reasoning about actions and change in answer set programming. In V. Lifschitz and I. Niemelä, editors.

Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning, *Lecture Notes in Computer Science*, vol. 2923, pages 61–73. Springer, Berlin, 2004.

[14] P. Doherty. Reasoning about action and change using occlusion. In A.G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence*, pages 401–405, Chichester, UK, 1994. John Wiley.

[15] P. Doherty and W. Łukaszewicz. Circumscribing features and fluents. In D.M. Gabbay and H.J. Ohlbach, editors. *Temporal Logic*, *Lecture Notes in Computer Science*, vol. 827, pages 82–100. Springer, Berlin, 1994.

[16] P. Doherty, W. Łukaszewicz, and A. Szałas. Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning*, 18(3):297–336, 1997.

[17] K. Eshghi. Abductive planning with event calculus. In R.A. Kowalski and K.A. Bowen, editors. *Logic Programming*: *Proceedings of the Fifth International Conference and Symposium*, vol. 1, pages 562–579. MIT Press, Cambridge, MA, 1988.

[18] A.D.H. Farrell, M.J. Sergot, M. Sallé, and C. Bartolini. Using the event calculus for tracking the normative state of contracts. *International Journal of Cooperative Information Systems*, 14(2–3):99–129, 2005.

[19] J.J. Finger. Exploiting constraints in design synthesis. PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, 1987.

[20] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R.A. Kowalski and K.A. Bowen, editors. *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, vol. 2, pages 1070–1080. MIT Press, Cambridge, MA, 1988.

[21] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–386, 1991.

[22] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17(2–4):301–321, 1993.

[23] M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, Palo Alto, CA, 1987.

[24] M.L. Ginsberg and D.E. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35(2):165–195, 1988.

[25] E. Giunchiglia, J. Lee, V. Lifschitz, N.C. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153:49–104, 2004.

[26] E. Giunchiglia and V. Lifschitz. 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*, pages 623–630, Menlo Park, CA, 1998. AAAI Press.

[27] E. Giunchiglia and R. Sebastiani. Applying the Davis–Putnam procedure to nonclausal formulas. In *Proceedings of the Sixth Congress of the Italian Association for Artificial Intelligence*, Bologna, 1999.

[28] B. Grosof. 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*, pages 115–124, Menlo Park, CA, 1984. AAAI Press.

[29] J. Gustafsson and P. Doherty. Embracing occlusion in specifying the indirect effects of actions. In L.C. Aiello, J. Doyle, and S.C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 87–98, San Francisco, 1996. Morgan Kaufmann.

[30] A.R. Haas. The case for domain-specific frame axioms. In F.M. Brown, editor, *The Frame Problem in Artificial Intelligence*: *Proceedings of the 1987 Workshop*, pages 343–348, Los Altos, CA, 1987. Morgan Kaufmann.

[31] H. Halpin, J.D. Moore, and J. Robertson. Automatic analysis of plot for story rewriting. In D. Lin and D. Wu, editors, *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 127–133, Barcelona, Spain, 2004.

[32] S. Hanks and D.V. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[33] A.C. Kakas and L. Michael. Modeling complex domains of actions and change. In S. Benferhat and E. Giunchiglia, editors, *Proceedings of the Ninth International Workshop on Non-Monotonic Reasoning*, pages 380–390, Toulouse, France, 2002.

[34] A.C. Kakas, L. Michael, and R. Miller. Modular-E: an elaboration tolerant approach to the ramification and qualification problems. In S. McIlraith, P. Peppas, and M. Thielscher, editors, *Seventh International Symposium on Logical Formalizations of Commonsense Reasoning*, Corfu, Greece, 2005.

[35] A.C. Kakas and R. Miller. Reasoning about actions, narratives and ramifications. *Linköping Electronic Articles in Computer and Information Science*, 2(012), 1997.

[36] A.C. Kakas and R. Miller. A simple declarative language for describing narratives with actions. *Journal of Logic Programming*, 31(1–3):157–200, 1997.

[37] A.C. Kakas, R. Miller, and F. Toni. An argumentation framework for reasoning about actions and change. In M. Gelfond, N. Leone, and G. Pfeifer, editors. *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning*, *Lecture Notes in Computer Science*, vol. 1730, pages 78–91. Springer, Berlin, 1999.

[38] A.C. Kakas, R. Miller, and F. Toni. E-RES—A system for reasoning about actions, events and observations. In C. Baral and M. Truszczynski, editors, *Proceedings of the Eighth International Workshop on Non-Monotonic Reasoning*, Breckenridge, CO, 2000.

[39] R.A. Kowalski. *Logic for Problem Solving*. North-Holland, New York, 1979.

[40] R.A. Kowalski. Database updates in the event calculus. Technical Report DOC 86/12, London: Imperial College of Science, Technology, and Medicine, 1986.

[41] R.A. Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146, 1992.

[42] R.A. Kowalski. Legislation as logic programs. In G. Comyn, N.E. Fuchs, and M. Ratcliffe, editors. *Logic Programming in Action, Second International Logic Programming Summer School*, *Lecture Notes in Computer Science*, vol. 636, pages 203–230. Springer, Berlin, 1992.

[43] R.A. Kowalski and F. Sadri. The situation calculus and event calculus compared. In M. Bruynooghe, editor. *Logic Programming: The 1994 International Symposium*, pages 539–553. MIT Press, Cambridge, MA, 1994.

[44] R.A. Kowalski and F. Sadri. Reconciling the event calculus with the situation calculus. *Journal of Logic Programming*, 31(1–3):39–58, 1997.

[45] R.A. Kowalski and M.J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[46] F. Lévy and J.J. Quantz. Representing beliefs in a situated event calculus. In H. Prade, editor, *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, pages 547–551, Chichester, UK, 1998. John Wiley.

[47] V. Lifschitz. Computing circumscription. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 121–127, Los Altos, CA, 1985. Morgan Kaufmann.

[48] V. Lifschitz. Formal theories of action. In F.M. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pages 35–57, Los Altos, CA, 1987. Morgan Kaufmann.

[49] V. Lifschitz. Pointwise circumscription. In M.L. Ginsberg, editor. *Readings in Nonmonotonic Reasoning*, pages 179–193. Morgan Kaufmann, Los Altos, CA, 1987.

[50] V. Lifschitz. Frames in the space of situations. *Artificial Intelligence*, 46(3):365–376, 1990.

[51] V. Lifschitz. Circumscription. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors. *Nonmonotonic Reasoning and Uncertain Reasoning*, *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. 3, pages 298–352. Oxford University Press, Oxford, 1994.

[52] V. Lifschitz. Book review: M. Shanahan, Solving the frame problem. *Artificial Intelligence*, 123(1–2):265–268, 2000.

[53] F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1985–1993, San Mateo, CA, 1995. Morgan Kaufmann.

[54] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, 1994.

[55] N.C. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1978–1984, San Mateo, CA, 1995. Morgan Kaufmann.

[56] J. McCarthy. Epistemological problems of artificial intelligence. In R. Reddy, editor, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 1038–1044, Los Altos, CA, 1977. William Kaufmann.

[57] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1–2):27–39, 1980.

[58] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. In *Proceedings of the Non-Monotonic Reasoning Workshop*, pages 295–324, Menlo Park, CA, 1984. AAAI Press.

[59] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[60] J. McCarthy. Generality in artificial intelligence. *Communications of the ACM*, 30(12):1030–1035, 1987.

[61] J. McCarthy. Elaboration tolerance. In R. Miller and M. Shanahan, editors, *Fourth Symposium on Logical Formalizations of Commonsense Reasoning*, London, 1998. Queen Mary and Westfield College.

[62] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors. *Machine Intelligence*, vol. 4, pages 463–502. Edinburgh University Press, Edinburgh, Scotland, 1969.

[63] R. Miller and M. Shanahan. Narratives in the situation calculus. *Journal of Logic and Computation*, 4(5):513–530, 1994.

[64] R. Miller and M. Shanahan. Reasoning about discontinuities in the event calculus. In L.C. Aiello, J. Doyle, and S.C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 63–74, San Francisco, 1996. Morgan Kaufmann.

[65] R. Miller and M. Shanahan. The event calculus in classical logic—Alternative axiomatisations. *Linköping Electronic Articles in Computer and Information Science*, 4(016), 1999.

[66] R. Miller and M. Shanahan. Some alternative formulations of the event calculus. In A.C. Kakas and F. Sadri, editors. *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part II*, *Lecture Notes in Computer Science*, vol. 2408, pages 452–490. Springer, Berlin, 2002.

[67] L. Missiaen, M. Bruynooghe, and M. Denecker. Chica, an abductive planning system based on event calculus. *Journal of Logic and Computation*, 5(5):579–602, 1995.

[68] L. Morgenstern. Mid-sized axiomatizations of commonsense problems: A case study in egg cracking. *Studia Logica*, 67:333–384, 2001.

[69] E.T. Mueller. Story understanding through multi-representation model construction. In G. Hirst and S. Nirenburg, editors, *Text Meaning: Proceedings of the HLT-NAACL 2003 Workshop*, pages 46–53. Association for Computational Linguistics, East Stroudsburg, PA, 2003.

[70] E.T. Mueller. Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, 14(5):703–730, 2004.

[71] E.T. Mueller. A tool for satisfiability-based commonsense reasoning in the event calculus. In V. Barr and Z. Markov, editors, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference*, pages 147–152, Menlo Park, CA, 2004. AAAI Press.

[72] E.T. Mueller. Understanding script-based stories using commonsense reasoning. *Cognitive Systems Research*, 5(4):307–340, 2004.

[73] E.T. Mueller. Event calculus answer set programming. http://www.signiform.com/csr/ecas/, 2005.

[74] E.T. Mueller. *Commonsense Reasoning*. Morgan Kaufmann, San Francisco, 2006.

[75] E.T. Mueller. Event calculus and temporal action logics compared. *Artificial Intelligence*, 170(11):1017–1029, 2006.

[76] E.T. Mueller. Modelling space and time in narratives about restaurants. *Literary and Linguistic Computing*, 22(1):67–84, 2007.

[77] E.T. Mueller and G. Sutcliffe. Discrete event calculus deduction using first-order automated theorem proving. In B. Konev and S. Schulz, editors. *Proceedings of the Fifth International Workshop on the Implementation of Logics,* number ULCS-05-003, pages 43–56. Department of Computer Science, University of Liverpool, Liverpool, UK, 2005.

[78] E.T. Mueller and G. Sutcliffe. Reasoning in the event calculus using first-order automated theorem proving. In I. Russell and Z. Markov, editors. *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference*, pages 840–841. AAAI Press, Menlo Park, CA, 2005.

[79] I. Niemelä and P. Simons. Smodels—an implementation of the stable model and well-founded semantics for normal logic programs. In J. Dix, U. Furbach, and A. Nerode, editors. *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning*, *Lecture Notes in Computer Science*, vol. 1265, pages 420–429. Springer, Berlin, 1997.

[80] A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In J.A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*, vol. 1, pages 335–367. Elsevier and MIT Press, Amsterdam and Cambridge, MA, 2001.

[81] E.P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332, San Mateo, CA, 1989. Morgan Kaufmann.

[82] J.A. Pinto and R. Reiter. Temporal reasoning in logic programming: A case for the situation calculus. In D.S. Warren, editor. *Logic Programming: Proceedings of the Tenth International Conference*, pages 203–221. MIT Press, Cambridge, MA, 1993.

[83] D.A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[84] R. Reiter. Circumscription implies predicate completion (sometimes). In D.L. Waltz, editors, *Proceedings of the National Conference on Artificial Intelligence*, pages 418–420, Menlo Park, CA, 1982. AAAI Press.

[85] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, MA, 2001.

[86] J.A. Robinson and A. Voronkov. *Handbook of Automated Reasoning, vols. 1 and 2*. Elsevier and MIT Press, Amsterdam and Cambridge, MA, 2001.

[87] F. Sadri. Three recent approaches to temporal reasoning. In A.P. Galton, editor. *Temporal Logics and their Applications*, pages 121–168. Academic Press, London, 1987.

[88] F. Sadri and R.A. Kowalski. Variants of the event calculus. In L. Sterling, editor, *Logic Programming: The Twelfth International Conference*, pages 67–81, Cambridge, MA, 1995. MIT Press.

[89] E. Sandewall. Filter preferential entailment for the logic of action in almost continuous worlds. In N.S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 894–899, San Mateo, CA, 1989. Morgan Kaufmann.

[90] E. Sandewall. *Features and Fluents: The Representation of Knowledge about Dynamical Systems*, vol. I. Oxford University Press, Oxford, 1994.

[91] L.K. Schubert. 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, editors. *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, Dordrecht, 1990.

[92] B. Selman, H.A. Kautz, and D.A. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 50–54, San Mateo, CA, 1997. Morgan Kaufmann.

[93] M. Shanahan. Prediction is deduction but explanation is abduction. In N.S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1055–1060, San Mateo, CA, 1989. Morgan Kaufmann.

[94] M. Shanahan. Representing continuous change in the event calculus. In L.C. Aiello, editor, *Proceedings of the Ninth European Conference on Artificial Intelligence*, pages 598–603, London, 1990. Pitman.

[95] M. Shanahan. A circumscriptive calculus of events. *Artificial Intelligence*, 77:249–284, 1995.

[96] M. Shanahan. Robotics and the common sense informatic situation. In W. Wahlster, editor, *Proceedings of the Twelfth European Conference on Artificial Intelligence*, pages 684–688, Chichester, UK, 1996. John Wiley.

[97] M. Shanahan. Event calculus planning revisited. In S. Steel and R. Alami, editors. *Recent Advances in AI Planning*, *Lecture Notes in Computer Science*, vol. 1348, pages 390–402. Springer, Berlin, 1997.

[98] M. Shanahan. *Solving the Frame Problem*. MIT Press, Cambridge, MA, 1997.

[99] M. Shanahan. A logical formalisation of Ernie Davis's egg cracking problem. In R. Miller and M. Shanahan, editors, *Fourth Symposium on Logical Formalizations of Commonsense Reasoning*, London, 1998. Queen Mary and Westfield College.

[100] M. Shanahan. The event calculus explained. In M.J. Wooldridge and M.M. Veloso, editors. *Artificial Intelligence Today*: *Recent Trends and Developments*, *Lecture Notes in Computer Science*, vol. 1600, pages 409–430. Springer, Berlin, 1999.

[101] M. Shanahan. The ramification problem in the event calculus. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 140–146, San Mateo, CA, 1999. Morgan Kaufmann.

[102] M. Shanahan. What sort of computation mediates best between perception and action? In H.J. Levesque and F. Pirri, editors. *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 352–369. Springer, Berlin, 1999.

[103] M. Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44(1–3):207–240, 2000.

[104] M. Shanahan. Abductive event calculus planners [Computer software], 2000.

[105] M. Shanahan. The frame problem. In L. Nadel, editor. *Encyclopedia of Cognitive Science*, vol. 2, pages 144–150. Nature Publishing Group, London, 2002.

[106] M. Shanahan. An attempt to formalise a non-trivial benchmark problem in common sense reasoning. *Artificial Intelligence*, 153:141–165, 2004.

[107] M. Shanahan. Perception as abduction: Turning sensor data into meaningful representation. *Cognitive Science*, 29:103–134, 2005.

[108] M. Shanahan and D.A. Randell. A logic-based formulation of active visual perception. In D. Dubois, C.A. Welty, and M.-A. Williams, editors, *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning*, pages 64–72, Menlo Park, CA, 2004. AAAI Press.

[109] M. Shanahan and M. Witkowski. Event calculus planning through satisfiability. *Journal of Logic and Computation*, 14(5):731–745, 2004.
[110] G. Sutcliffe and C.B. Suttner. The TPTP problem library for automated theorem proving, 2005.
[111] M. Thielscher. Ramification and causality. *Artificial Intelligence*, 89:317–364, 1997.
[112] K. Van Belleghem, M. Denecker, and D. De Schreye. On the relation between situation calculus and event calculus. *Journal of Logic Programming*, 31(1–3):3–37, 1997.
[113] M. van Lambalgen and F. Hamm. *The Proper Treatment of Events*. Blackwell, Malden, MA, 2005.
[114] J. Wilk. Dynamic workflow pulling the strings. Distinguished Project (MEng). Department of Computing, Imperial College London, London, 2004.
[115] P. Yolum and M.P. Singh. 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, 2004.