CHAPTER

The Effects of Events 3

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

3.1 POSITIVE AND NEGATIVE EFFECT AXIOMS

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

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

Definition 3.1. If γ is a condition representing the context, α is an event term, β is a fluent term, and τ is a timepoint term, then

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

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

Definition 3.2. If γ is a condition representing the context, α is an event term, β is a fluent term, and τ is a timepoint term, then

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

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

Using the conjunction of axioms EC, a fluent is true (false) for times greater than the time of the initiating (terminating) event. Using the conjunction of axioms DEC, a fluent is true (false) starting one timepoint after the time of the initiating (terminating) event. For example, suppose we have $Initiates(Init, Fluent, t)$, $Happens(Init, 1)$,

Terminates(*Term*, *Fluent*, *t*), and *Happens*(*Term*, 4). Figure 3.1 shows when *Fluent* is true using the conjunction of axioms EC; Figure 3.2 shows when *Fluent* is true using DEC.

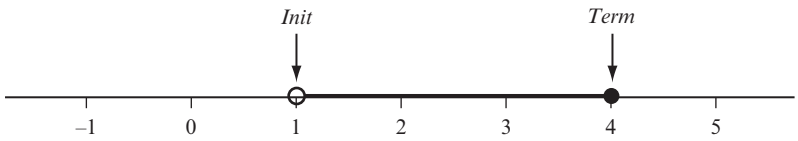

FIGURE 3.1

Truth value of a fluent in EC.

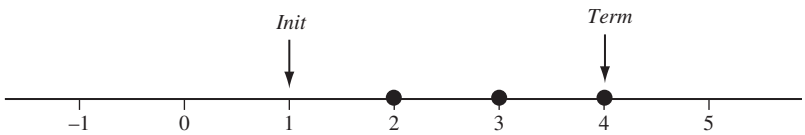

FIGURE 3.2

Truth value of a fluent in DEC.

3.1.1 EXAMPLE: TELEPHONE

We perform complex commonsense reasoning about the effects of events whenever we use a telephone. The behavior of a phone is highly context-sensitive. If we pick up an idle phone, we expect to get a dial tone. But if we pick up a phone that is ringing, we expect to be connected to the caller.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$Happens(PickUp(Agent1, Phone1), 0) \tag{3.32}$$

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

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

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

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

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

$$Initiates(e, f, t) \Leftrightarrow \tag{3.35}$$

$$\exists a, p\,(e = PickUp(a, p) \wedge f = DialTone(p) \wedge HoldsAt(Idle(p), t)) \vee$$

$$\exists a, p\,(e = SetDown(a, p) \wedge f = Idle(p) \wedge HoldsAt(DialTone(p), t)) \vee$$

$$\exists a, p_1, p_2\,(e = Dial(a, p_1, p_2) \wedge$$

$$f = Ringing(p_1, p_2) \wedge$$

$$HoldsAt(DialTone(p_1), t) \wedge$$

$$HoldsAt(Idle(p_2), t)) \vee$$

$$\exists a, p_1, p_2\,(e = Dial(a, p_1, p_2) \wedge$$

$$f = BusySignal(p_1) \wedge$$

$$HoldsAt(DialTone(p_1), t) \wedge$$

$$\neg HoldsAt(Idle(p_2), t)) \vee$$

$$\exists a, p\,(e = SetDown(a, p) \wedge f = Idle(p) \wedge HoldsAt(BusySignal(p), t)) \vee$$

$$\exists a, p_1, p_2\,(e = SetDown(a, p_1) \wedge$$

$$f = Idle(p_1) \wedge$$

$$HoldsAt(Ringing(p_1, p_2), t)) \vee$$

$$\exists a, p_1, p_2\,(e = SetDown(a, p_1) \wedge$$

$$f = Idle(p_2) \wedge$$

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

$$
\begin{aligned}
&(e = Dial(a, p_1, p_2) \wedge \\
&f = Idle(p_2) \wedge \\
&HoldsAt(DialTone(p_1), t) \wedge \\
&HoldsAt(Idle(p_2), t)) \vee \\
&\exists a, p_1, p_2 \\
&(e = Dial(a, p_1, p_2) \wedge \\
&f = DialTone(p_1) \wedge \\
&HoldsAt(DialTone(p_1), t) \wedge \\
&\neg HoldsAt(Idle(p_2), t)) \vee \\
&\exists a, p\, (e = SetDown(a, p) \wedge f = BusySignal(p) \wedge HoldsAt(BusySignal(p), t)) \vee \\
&\exists a, p_1, p_2\, (e = SetDown(a, p_1) \ \wedge f = Ringing(p_1, p_2) \wedge \\
&HoldsAt(Ringing(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\, (e = PickUp(a, p_2) \wedge \\
&f = Ringing(p_1, p_2) \wedge \\
&HoldsAt(Ringing(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\, (e = SetDown(a, p_1) \wedge \\
&f = Connected(p_1, p_2) \wedge \\
&HoldsAt(Connected(p_1, p_2), t)) \vee \\
&\exists a, p_1, p_2\, (e = SetDown(a, p_2) \wedge \\
&f = Connected(p_1, p_2) \wedge \\
&HoldsAt(Connected(p_1, p_2), t)) \vee \\
&\exists a, p\, (e = SetDown(a, p) \wedge \\
&f = Disconnected(p) \wedge \\
&HoldsAt(Disconnected(p), t)) \\
&\neg Releases(e, f, t)
\end{aligned}
\tag{3.37}
$$

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

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

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

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

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

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

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

3.2 EFFECT AXIOM IDIOMS

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

Setting and resetting

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

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

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

Flipping

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

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

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

Selection

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

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

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

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

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

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

Functional modification

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

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

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

A common example is incrementing a value:

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

Many-to-many mapping

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

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

Many events may initiate (or terminate) one fluent:

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

3.3 PRECONDITIONS

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

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

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

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

3.3.1 FLUENT PRECONDITIONS

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

Definition 3.3. If γ is a condition, α is an event term, β is a fluent term, and τ is a timepoint term, then

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

and

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

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

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

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

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

3.3.2 ACTION PRECONDITIONS

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

Definition 3.4. If γ is a condition, α is an event term, and τ is a timepoint term, then

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

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

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

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

3.3.3 EXAMPLE: WALKING THROUGH A DOOR

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

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

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

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

We then get inconsistency.

Proposition 3.2. *The conjunction of (3.41)–(3.43) is inconsistent.*

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

3.4 STATE CONSTRAINTS

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

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

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

Irreflexive and antisymmetric relation

Suppose we have a fluent $On(o_1, o_2)$, which represents that an object o_1 is on top of an object o_2. We can use state constraints to specify that On denotes an irreflexive and antisymmetric relation. That is, an object can never be on top of itself:

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

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

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

Functional and total relation

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

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

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

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

State constraints can be used to address the qualification problem, as demonstrated in the following example: Let $Occupies(p, s)$ represent that a chess piece p occupies

Table 3.1 Typical State Constraints

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

a square s of a chessboard. We wish to represent that it is not possible to move a piece onto a square that is already occupied. We may do this by specifying that *Occupies* denotes an injective relation. That is, at most one piece occupies a square at a time:

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

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

Negation

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

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

Intersection

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

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

Exactly one

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

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

3.4.1 EXAMPLE: TELEPHONE REVISITED

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

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

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

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

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

A phone cannot be connected to itself:

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

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

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

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

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

These state constraints simplify specification of initial conditions. For example, from $HoldsAt(Idle(Phone1), 0)$, $HoldsAt(Idle(Phone2), 0)$, (3.45), (3.46), (3.47), (3.48), and (3.49), we have all of the following:

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

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

BIBLIOGRAPHIC NOTES

GPS

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

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

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

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

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

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

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

QA3

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

STRIPS

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

Operator: o

Precondition: c

Delete list: $d_1, \ldots, d_n$

Add list: $a_1, \ldots, a_n$

is similar to the set of event calculus formulas

$$c \Rightarrow \mathit{Terminates}(o, d_1, t)$$
$$\vdots$$
$$c \Rightarrow \mathit{Terminates}(o, d_n, t)$$
$$c \Rightarrow \mathit{Initiates}(o, a_1, t)$$
$$\vdots$$
$$c \Rightarrow \mathit{Initiates}(o, a_n, t)$$

Inconsistency

In EC and DEC, simultaneously initiating and terminating a fluent produces inconsistency. This is also the case in most of the versions of the classical logic event calculus (Shanahan, 1996, 1997b, 1999a, 2004). But in the basic axiomatization of

the event calculus of R. Miller and Shanahan (1999, pp. 1-3; 2002, pp. 453-455) (not the particular version of R. Miller and Shanahan used in this book called EC), simultaneously initiating and terminating a fluent results in two classes of models: one in which the fluent is true and one in which the fluent is false. This can be used to represent nondeterminism. R. Miller and Shanahan (2002, pp. 459 and 460) give the following example:

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

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

Qualification problem

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

Preconditions

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

State constraints

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

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

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

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

EXERCISES

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

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

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

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

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

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

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

3.8 Formalize lighting and putting out a fire.

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

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

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

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

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