The Event Calculus Explained

Murray Shanahan

Department of Electrical and Electronic Engineering,
Imperial College,
Exhibition Road, London SW7 2BT,
England.
Email: m.shanahan@ic.ac.uk

Abstract

This article presents the event calculus, a logic-based formalism for representing actions and their effects. A circumscriptive solution to the frame problem is deployed which reduces to monotonic predicate completion. Using a number of benchmark examples from the literature, the formalism is shown to apply to a variety of domains, including those featuring actions with indirect effects, actions with non-deterministic effects, concurrent actions, and continuous change.

Introduction

Central to many complex computer programs that take decisions about how they or other agents should act is some form of representation of the effects of actions, both their own and those of other agents. To the extent that the design of such programs is to be based on sound engineering principles, rather than *ad hoc* methods, it's vital that the subject of how actions and their effects are represented has a solid theoretical basis. Hence the need for logical formalisms for representing action, which, although they may or may not be realised directly in computer programs, nevertheless offer a theoretical yardstick against which any actually deployed system of representation can be measured.

This article presents one such formalism, namely the *event calculus*. There are many others, the most prominent of which is probably the *situation calculus* [McCarthy & Hayes, 1969], and the variant of the event calculus presented here should be thought of as just one point in a space of possible action formalisms which the community has yet to fully understand.

The calculus described here is based on first-order predicate calculus, and is capable of representing a variety of phenomena, including actions with indirect effects, actions with non-deterministic effects, compound actions, concurrent actions, and continuous change. It incorporates a straightforward solution to the frame problem which is robust insofar as it works in the presence of each of these phenomena. Although this solution employs a non-monotonic formalism, namely circumscription, in most of the cases of interest here, the circumscriptions reduce to monotonic predicate completions.

The article is tutorial in form, and presents a large number of examples, illustrating how different benchmark scenarios can be represented in the event calculus. No proofs are given of the propositions asserted here, as these, or proofs of similar propositions, can be found elsewhere, mainly in [Shanahan, 1997a]. To make the presentation more digestible, three versions of the formalism, of increasing sophistication, are given in turn — the *simple* event calculus, the *full* event calculus, and the *extended* event calculus. Most of the material is drawn directly from three sources: [Shanahan, 1997a], [Shanahan, 1997b], and [Shanahan, 1999].

1 Event Calculus Basics

The event calculus was introduced by Kowalski and Sergot as a logic programming formalism for representing events and their effects, especially in database applications [Kowalski & Sergot, 1986]. Though still couched in logic programming terms, a later simplified version presented by Kowalski is closer to the one presented here [Kowalski, 1992]. A number of event calculus dialects have sprung up since Kowalski and Sergot's original paper. The one described here, which is expressed in first-order predicate calculus with circumscription, is drawn from Chapter 16 of [Shanahan, 1997a].

1.1 What the Event Calculus Does

Figure 1 summarises the way the event calculus functions. The event calculus is a logical mechanism that infers what's true when given what happens when and what actions do. The "what happens when" part is a *narrative* of events, and the "what actions do" part describes the *effects* of actions. For example, given that eating makes me happy and that I eat at 12:00, the event calculus licenses the conclusion that I'm happy at 12:05.[1]

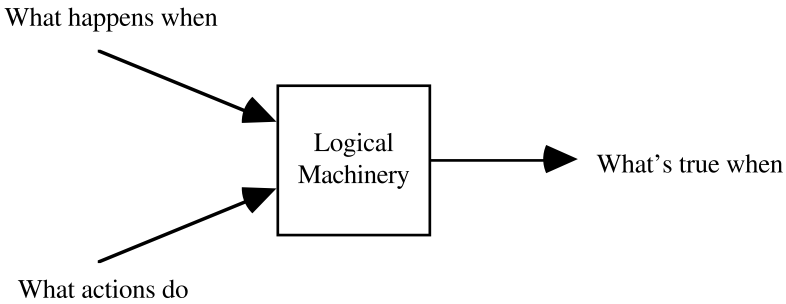

Figure 1: How the Event Calculus Functions

From Figure 1, without fleshing out the formal details, we can see how the event calculus can supply a logical foundation for a number of reasoning tasks. These can be broadly categorised into *deductive* tasks, *abductive* tasks, and *inductive* tasks. In a deductive task, "what happens when" and "what actions do" are given and "what's true when" is required. Deductive tasks include temporal *projection* or *prediction*, where the outcome of a known sequence of actions is sought.

In an abductive task, "what actions do" and "what's true when" are supplied, and "what happens when" is required. In other words, a sequence of actions is sought that leads to a given outcome. Examples of such tasks include temporal *explanation* or *postdiction*, certain kinds of *diagnosis*, and *planning*.

Finally, in an inductive task, "what's true when" and "what happens when" are supplied, but "what actions do" is required. In this case, we're seeking a set of general rules, a theory of the effects of actions, that accounts for observed data. Inductive tasks include certain kinds of *learning*, *scientific discovery*, and *theory formation*.

How can we render an informal characterisation such as that depicted in Figure 1 into something mathematically precise? A variety of design choices confronts us. To begin

[1] Of course, the conclusion rests on the assumption that nothing else happens between 12:00 and 12:05 to upset me. The event calculus makes such assumptions by default.

with, we have to make some meta-level decisions. What sort of logic are we going to employ? We could take the modal route, and build a new logic from scratch, defining a special language and semantics for handling actions. On the other hand, we could build on first-order predicate calculus, introducing suitable predicates and functions for representing the kind of action-related information we're interested in, and possibly presenting a set of axioms constraining the set of models we want. The event calculus adopts the latter approach.

1.2 The Ontology and Predicates of the Event Calculus

The first choice to be made in designing a first-order language for representing actions and their effects is the underlying *ontology*, that is to say the types of things over which quantification is permitted. The basic ontology of the event calculus comprises *actions* or *events* (or rather action or event *types*), *fluents* and *time points*.[2] A fluent is anything whose value is subject to change over time. This could be a quantity, such as "the temperature in the room", whose numerical value is subject to variation, or a proposition, such as "it is raining", whose truth value changes from time to time. We'll confine our attention here to propositional fluents.

Going hand in hand with the choice of ontology is the choice of basic *predicates*. In this section, we'll restrict ourselves to a simple event calculus, which has all the fundamental characteristics of the full version we'll study later but is easier to understand. Both versions of the calculus include predicates for saying what happens when, for describing the initial situation, for describing the effects of actions, and for saying what fluents hold at what times. Table 1 introduces the language elements of the simple event calculus.[3]

Formula	Meaning
Initiates(α,β,τ)	Fluent β starts to hold after action α at time τ
Terminates(α,β,τ)	Fluent β ceases to hold after action α at time τ
$\text{Initially}_P(\beta)$	Fluent β holds from time 0
$\tau1 < \tau2$	Time point $\tau1$ is before time point $\tau2$
Happens(α,τ)	Action α occurs at time τ
HoldsAt(β,τ)	Fluent β holds at time τ
Clipped($\tau1$,β,$\tau2$)	Fluent β is terminated between times $\tau1$ and $\tau2$

Table 1: Some Event Calculus Predicates

As this table shows, in the event calculus, fluents are *reified*. That is to say, fluents are first-class objects, which can be quantified over and can appear as the arguments to predicates. Formalisms in which fluents are unreified are, of course, possible, as we'll see later.

The commitments we've now made lead to Figure 2, which is the same as Figure 1, but made more precise using the newly introduced predicate symbols.

---

[2] I use the terms action and event interchangeably.

[3] Many-sorted predicate calculus is used throughout this article. Time points are assumed to be interpreted by the reals, and the corresponding comparative predicates and arithmetic functions are taken for granted. Predicate and function symbols always start with an upper-case letter, and variables always start with a lower-case letter.

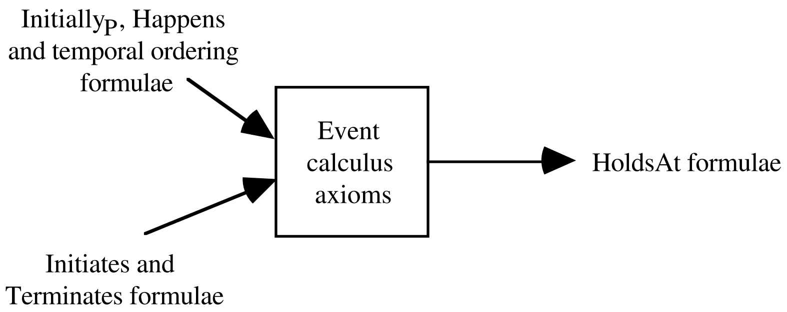

Figure 2: A More Precise Version of Figure 1

1.3 The Axioms of the Simple Event Calculus

We now require a suitable collection of axioms relating the various predicates together. The following three, whose conjunction will be denoted SC, will do the job.[4]

$$\mathrm{HoldsAt}(f,t) \leftarrow \mathrm{Initially}_P(f) \land \neg\, \mathrm{Clipped}(0,f,t) \qquad \text{(SC1)}$$

$$\mathrm{HoldsAt}(f,t2) \leftarrow \qquad \text{(SC2)}$$
$$\quad \mathrm{Happens}(a,t1) \land \mathrm{Initiates}(a,f,t1) \land t1 < t2 \land \neg\, \mathrm{Clipped}(t1,f,t2)$$

$$\mathrm{Clipped}(t1,f,t2) \leftrightarrow \qquad \text{(SC3)}$$
$$\quad \exists\, a,t\, [\mathrm{Happens}(a,t) \land t1 < t < t2 \land \mathrm{Terminates}(a,f,t)]$$

Axiom (SC1) says that a fluent holds at a time t if it held at time 0, and hasn't been terminated between 0 and t. Axiom (SC2) says that a fluent holds at time t if it was initiated at some time before t and hasn't been terminated between then and t.

Note that, according to these axioms, *a fluent does not hold at the time of the event that initiates it but does hold at the time of the event that terminates it*. In other words, the intervals over which fluents hold are open on the left and closed on the right.

A superficial look at the logical machinery we've now assembled might be enough to convince someone that it was sufficient for its intended role. But an important issue has been neglected, namely the *frame problem*. In the next section, we'll take a look at the frame problem as it arises with the event calculus.

2 The Frame Problem in the Event Calculus

How do we use logic to represent the effects of actions, without having to explicitly represent all their non-effects? This, in a nutshell, is the frame problem. First brought to light by McCarthy and Hayes in the late Sixties [McCarthy & Hayes, 1969], it has exercised the minds of numerous AI researchers over the years. To see how it arises in the context of the event calculus, let's consider an example, namely the well-known Yale shooting scenario [Hanks & McDermott, 1987]. This will also serve to illustrate the style in which event calculus formulae are usually written.

2.1 The Yale Shooting Scenario

In this version of the Yale shooting domain there are three types of action — a Load action, a Sneeze action, and a Shoot action — and three fluents — Loaded, Alive and

[4] Throughout the article, variables are assumed to be universally quantified with maximum possible scope, unless otherwise indicated.

Dead. The effect of a Load action is to make Loaded hold, a Shoot action makes Dead hold and Alive not hold so long as Loaded holds at the time, and a Sneeze action has no effects. The following three Initiates and Terminates formulae describe these effects.

Initiates(Load,Loaded,t) (Y1.1)

Initiates(Shoot,Dead,t) $\leftarrow$ HoldsAt(Loaded,t) (Y1.2)

Terminates(Shoot,Alive,t) $\leftarrow$ HoldsAt(Loaded,t) (Y1.3)

The Yale shooting scenario comprises a Load action followed by a Sneeze action followed by a Shoot action. Using some arbitrarily chosen time point constant symbols, this can be represented by the following Happens and temporal ordering formulae.

Initially$_P$(Alive) (Y2.1)

Happens(Load,T1) (Y2.2)

Happens(Sneeze,T2) (Y2.3)

Happens(Shoot,T3) (Y2.4)

T1 < T2 (Y2.5)

T2 < T3 (Y2.6)

T3 < T4 (Y2.7)

Now let Σ be the conjunction of (Y1.1) to (Y1.3), and let Δ be the conjunction of (Y2.1) to (Y2.7). The intention is that we should have,

$\Sigma \wedge \Delta \wedge$ SC $\vDash$ HoldsAt(Dead,T4).

Unfortunately this sequent is not valid. This is because we've neglected to describe explicitly the non-effects of actions. In particular, we haven't said that the Sneeze action doesn't unload the gun. So there are, for example, models of SC $\wedge\ \Sigma \wedge \Delta$ in which Terminates(Sneeze,Loaded,T2) is true, Holds(Alive,T4) is true, and HoldsAt(Dead,T4) is false.

In fact, there's a whole spectrum of annoying possibilities that we must rule out before we have a theory from which the intended conclusions follow. In addition to describing the non-effects of actions, we must describe the non-occurrence of actions. And, more trivially, we must include formulae that rule out the possibility that, say, the Sneeze action and the Shoot action are identical.

The first of these issues is easily dealt with. In general, when describing the effects of actions, we always need to include a set of uniqueness-of-names axioms for fluents and actions. In the present case, we have the following formulae, which use a notation taken from Baker [1991].

UNA[Load, Sneeze, Shoot] (Y3.1)

UNA[Loaded, Alive, Dead] (Y3.2)

These entail that Load $\neq$ Sneeze, Loaded $\neq$ Alive, and so on.

2.2 Using Predicate Completion

The non-effects of actions and the non-occurrence of events can be made explicit by supplying the *completions* of the Initiates, Terminates and Happens predicates. Formulae (Y1.1) and (Y1.2) are replaced by the following.

Initiates(a,f,t) $\leftrightarrow$ (Y4.1)

[a = Load $\wedge$ f = Loaded] $\vee$ [a = Shoot $\wedge$ f = Dead $\wedge$ HoldsAt(Loaded,t)]

Terminates(a,f,t) $\leftrightarrow$ a = Shoot $\wedge$ f = Dead $\wedge$ HoldsAt(Loaded,t) (Y4.2)

We retain formulae (Y2.1) and (Y2.5) to (Y2.7), but (Y2.2) to (Y2.4) are replaced by the completion of the Happens predicate.

$\mathrm{Initially_P(Alive)}$ (Y5.1)

$\mathrm{Happens}(a,t) \leftrightarrow$ (Y5.2)

$[a = \mathrm{Load} \wedge t = \mathrm{T1}] \vee [a = \mathrm{Sneeze} \wedge t = \mathrm{T2}] \vee [a = \mathrm{Shoot} \wedge t = \mathrm{T3}]$

$\mathrm{T1} < \mathrm{T2}$ (Y5.3)

$\mathrm{T2} < \mathrm{T3}$ (Y5.4)

$\mathrm{T3} < \mathrm{T4}$ (Y5.5)

Now let Ω be the conjunction of (Y3.1) and (Y3.2), let Σ be the conjunction of (Y4.1) and (Y4.2), and let Δ be the conjunction of (Y5.1) to (Y5.5). Now, as desired, we have,

$\Sigma \wedge \Delta \wedge \mathrm{SC} \wedge \Omega \vDash \mathrm{HoldsAt(Dead,T4)}$.

Here we have the seeds of a satisfactory solution to the frame problem. Generally, though, especially in non-trivial domains, it's highly desirable to have some logical mechanism that *automatically* constructs the completions of the Initiates, Terminates and Happens predicates from individual clauses like those in (Y1.1) to (Y1.3) and (Y2.2) to (Y2.4). As well as being notationally more convenient, this allows a theory to be constructed in a more modular fashion. It also makes our theories more *elaboration tolerant* [McCarthy, 1988], in the sense that new actions, new fluents, new effects of actions, and new event occurrences can easily be accommodated by an extant theory.

The usual way to address this issue is to adopt some form of *non-monotonic* formalism, such as *default logic* [Reiter, 1980], or *circumscription* [McCarthy, 1980] to formalise the *common sense law of inertia*, whereby a fluent is assumed to persist unless there is reason to believe otherwise. In doing so, care must be taken to avoid the so-called *Hanks-McDermott problem*, which arises when a formalisation of the common sense law of inertia admits unexpected change [Hanks & McDermott, 1987].

Although the Hanks-McDermott problem dominated research on reasoning about action for some years, there's no need to investigate it too closely here. (For a detailed discussion, see [Shanahan, 1997a].) This is because, in the context of the event calculus, a simple approach suffices to construct the predicate completions of Initiates, Terminates and Happens, and avoids the Hanks-McDermott problem altogether.

2.3 A Circumscriptive Solution to the Frame Problem

This simple approach is based on circumscription [McCarthy, 1980]. The idea of circumscription is to minimise the extensions of certain named predicates. That is to say, the circumscription of a formula Φ yields a theory in which these predicates have the smallest extension allowable according to Φ. The circumscription of Φ minimising the predicate ρ is written,

$\mathrm{CIRC}[\Phi ; \rho]$.

This is equivalent to the following second-order formula.

$\Phi \wedge \neg \exists q\, [\Phi(q) \wedge q < \rho]$

where,

- $q = \rho$ means $\forall \bar{x}\, [q(\bar{x}) \leftrightarrow \rho(\bar{x})]$,
- $q \leq \rho$ means $\forall \bar{x}\, [q(\bar{x}) \rightarrow \rho(\bar{x})]$,
- $q < \rho$ means $[q \leq \rho] \wedge \neg [q = \rho]$, and
- $\Phi(q)$ is the formula obtained by replacing all occurrences of ρ in Φ by q.

However, there's no need to understand this formula here. For the interested reader, a detailed discussion of circumscription can be found in [Lifschitz, 1994], and an extensive history of its application to the frame problem is presented in [Shanahan, 1997a].

This definition is straightforwardly extended to cover the minimisation of multiple predicates. (For the actual definition, see either of the works just cited). The circumscription of Φ minimising a tuple of predicates ρ^* is written,

$$\text{CIRC}[\Phi \,;\, \rho^*].$$

Now let's return to the event calculus. Given,

- a conjunction Σ of Initiates and Terminates formulae,
- a conjunction Δ of Initially_P, Happens and temporal ordering formulae, and
- a conjunction Ω of uniqueness-of-names axioms for actions and fluents,

we're interested in,

$$\text{CIRC}[\Sigma \,;\, \text{Initiates, Terminates}] \wedge \text{CIRC}[\Delta \,;\, \text{Happens}] \wedge \text{SC} \wedge \Omega.$$

The minimisation of Initiates and Terminates corresponds to the default assumption that actions have no unexpected effects, and the minimisation of Happens corresponds to the default assumption that there are no unexpected event occurrences. The key to this solution to the frame problem is the splitting of the theory into different parts, which are circumscribed separately. This technique is also employed in [Crawford & Etherington, 1992], [Doherty, 1994], [Kartha & Lifschitz, 1995], and [Lin, 1995], and is akin to what Sandewall calls *filtering* [Sandewall, 1994].

In most of the cases we're interested in here, Σ and Δ will be conjunctions of Horn clauses, and, according to a theorem of Lifschitz [1994], the separate circumscriptions therefore reduce to predicate completions. While we remain within the class of formulae to which Lifschitz's theorem is applicable, this solution is effectively monotonic, and can be likened to the frame problem solution proposed by Reiter [1991] based on the work of Haas [1987] and Schubert [1990].

We can now use the original, uncompleted formulae to formalise the Yale shooting scenario and, in the context of circumscription, we get the desired results. Let Σ be the conjunction of (Y1.1) to (Y1.3), and let Δ be the conjunction of (Y2.1) to (Y2.7). We have,

$$\begin{array}{l}\text{CIRC}[\Sigma \,;\, \text{Initiates, Terminates}] \wedge \\ \quad \text{CIRC}[\Delta \,;\, \text{Happens}] \wedge \text{SC} \wedge \Omega \vDash \text{HoldsAt(Dead,T4)}.\end{array}$$

This is because (Y4.1) and (Y4.2) follow from CIRC[Σ ; Initiates, Terminates], and (Y5.2) follows from CIRC[Δ ; Happens].

3 The Full Event Calculus

The event calculus of Section 1 is very limited in its applicability, and is only really meant to introduce the formalism's basic concepts. This section presents the full version of the formalism, which builds on the simple version of Section 1 in the following ways.

- It includes three new axioms, (EC4) to (EC6), which mirror (SC1) to (SC3), but which describe when a fluent does *not* hold. New predicates Initially_N and Declipped are introduced as counterparts to Initially_P and Clipped.
- It incorporates a three-argument version of Happens. This allows actions with duration, and facilitates the representation of compound actions.

- It incorporates a new predicate, Releases, which is used to disable the common sense law of inertia. This predicate was first introduced in [Kartha & Lifschitz, 1994], and is related to Sandewall's idea of *occlusion* [Sandewall, 1994].

As we'll see, the full formalism can also be used to represent domains involving actions with indirect effects and actions with non-deterministic effects.

3.1 New Predicates and New Axioms

Table 2 describes those predicates used in the full event calculus that weren't part of the simple version.

Formula	Meaning
Releases(α,β,τ)	Fluent β is not subject to inertia after action α at time τ
$\text{Initially}_N(\beta)$	Fluent β does not hold from time 0
Happens(α,τ1,τ2)	Action α occurs starts at time τ1 and ends at time τ2
Declipped(τ1,β,τ2)	Fluent β is initiated between times τ1 and τ2

Table 2: Four New Predicates

Here is the new set of axioms, whose conjunction will be denoted EC.

$$\text{HoldsAt}(f,t) \leftarrow \text{Initially}_P(f) \wedge \neg\, \text{Clipped}(0,f,t) \qquad \text{(EC1)}$$

$$\begin{aligned}&\text{HoldsAt}(f,t3) \leftarrow \qquad \text{(EC2)}\\ &\quad \text{Happens}(a,t1,t2) \wedge \text{Initiates}(a,f,t1) \wedge\\ &\qquad t2 < t3 \wedge \neg\, \text{Clipped}(t1,f,t3)\end{aligned}$$

$$\begin{aligned}&\text{Clipped}(t1,f,t4) \leftrightarrow \qquad \text{(EC3)}\\ &\quad \exists\, a,t2,t3\ [\text{Happens}(a,t2,t3) \wedge t1 < t3 \wedge t2 < t4 \wedge\\ &\qquad [\text{Terminates}(a,f,t2) \vee \text{Releases}(a,f,t2)]]\end{aligned}$$

$$\neg\, \text{HoldsAt}(f,t) \leftarrow \text{Initially}_N(f) \wedge \neg\, \text{Declipped}(0,f,t) \qquad \text{(EC4)}$$

$$\begin{aligned}&\neg\, \text{HoldsAt}(f,t3) \leftarrow \qquad \text{(EC5)}\\ &\quad \text{Happens}(a,t1,t2) \wedge \text{Terminates}(a,f,t1) \wedge\\ &\qquad t2 < t3 \wedge \neg\, \text{Declipped}(t1,f,t3)\end{aligned}$$

$$\begin{aligned}&\text{Declipped}(t1,f,t4) \leftrightarrow \qquad \text{(EC6)}\\ &\quad \exists\, a,t2,t3\ [\text{Happens}(a,t2,t3) \wedge t1 < t3 \wedge t2 < t4 \wedge\\ &\qquad [\text{Initiates}(a,f,t2) \vee \text{Releases}(a,f,t2)]]\end{aligned}$$

$$\text{Happens}(a,t1,t2) \rightarrow t1 \leq t2 \qquad \text{(EC7)}$$

The two-argument version of Happens is now defined in terms of the three-argument version, as follows.

$$\text{Happens}(a,t) \equiv_{def} \text{Happens}(a,t,t)$$

Note that if Releases is always false, then (SC1) to (SC3) follow from (EC1) to (EC7).

The frame problem is overcome in much the same way as with the simple event calculus. Given,

- a conjunction Σ of Initiates, Terminates and Releases formulae,
- a conjunction Δ of Initially_P, Initially_N, Happens and temporal ordering formulae, and
- a conjunction Ω of uniqueness-of-names axioms for actions and fluents,

we're interested in,

CIRC[Σ ; Initiates, Terminates, Releases] $\wedge$ CIRC[Δ ; Happens] $\wedge$ EC $\wedge$ Ω.

3.2 Using the Full Formalism

To see the full formalism working, let's take a look at the so-called Russian turkey shoot [Sandewall, 1991], an extension of the Yale shooting problem that includes a Spin action instead of a Sneeze, which, as in the "game" of Russian roulette may or may not result in the gun being unloaded. In addition, since we can now reason about when fluents do not hold, we only need the Alive fluent and can dispense with the fluent Dead. (This example doesn't highlight the use of the three-argument Happens predicate, whose primary application is to the representation of compound actions. These will be covered later.) Here are the effect axioms.

Initiates(Load,Loaded,t) (R1.1)

Terminates(Shoot,Alive,t) $\leftarrow$ HoldsAt(Loaded,t) (R1.2)

Releases(Spin,Loaded,t) (R1.3)

Here are the formulae describing the narrative of events.

Initially_P(Alive) (R2.1)

Happens(Load,T1) (R2.2)

Happens(Spin,T2) (R2.3)

Happens(Shoot,T3) (R2.4)

T1 < T2 (R2.5)

T2 < T3 (R2.6)

T3 < T4 (R2.7)

Finally we have,

UNA[Load, Spin, Shoot] (R3.1)

UNA[Loaded, Alive] (R3.2)

Let Σ be the conjunction of (R1.1) to (R1.3), let Δ be the conjunction of (R2.1) to (R2.7), and let Ω be the conjunction of (R3.1) and (R3.2). Now, although we have,

CIRC[Σ ; Initiates, Terminates, Releases] $\wedge$
CIRC[Δ ; Happens] $\wedge$ EC $\wedge$ $\Omega \vDash$ HoldsAt(Loaded,T2) $\wedge$ HoldsAt(Alive,T3)

we do *not* have either of the following.

CIRC[Σ ; Initiates, Terminates, Releases] $\wedge$
CIRC[Δ ; Happens] $\wedge$ EC $\wedge$ $\Omega \vDash$ HoldsAt(Alive,T4)

CIRC[Σ ; Initiates, Terminates, Releases] $\wedge$
CIRC[Δ ; Happens] $\wedge$ EC $\wedge$ $\Omega \vDash \neg$ HoldsAt(Alive,T4)

This is because the Spin action has "released" the Loaded fluent from the common sense law of inertia. So in some models the gun is loaded at the time of the Shoot action, while in others it is not.

In fact, this is a somewhat flawed representation of the Russian turkey shoot scenario, since the Loaded fluent, after being released, is completely wild — the axioms permit, for example, models in which it oscillates from true to false many times between T2 and T3. A better formalisation is possible using the techniques described below for representing actions with non-deterministic effects.

3.3 State Constraints

The *ramification problem* is the frame problem for actions with indirect effects, that is to say actions with effects beyond those described explicitly by their associated effect axioms. Although it's always possible to encode these indirect effects as direct effects instead, the use of constraints describing indirect effects ensures a modular representation and can dramatically shorten an axiomatisation. One way to represent actions with indirect effects is through *state constraints*. These express logical relationships that have to hold between fluents at all times. We'll look into other aspects of the ramification problem in a later section, but for now we'll focus solely on state constraints.

In the event calculus, state constraints are HoldsAt formulae with a universally quantified time argument. Here's an example, whose intended meaning should be obvious.

$$\text{HoldsAt(Happy(x),t)} \leftrightarrow \neg\, \text{HoldsAt(Hungry(x),t)} \wedge \neg\, \text{HoldsAt(Cold(x),t)} \qquad \text{(H1.1)}$$

Note that this formula incorporates of fluents with arguments. Actions may also be parameterised, as in the following effect axioms.

$$\text{Terminates(Feed(x),Hungry(x),t)} \qquad \text{(H2.1)}$$

$$\text{Terminates(Clothe(x),Cold(x),t)} \qquad \text{(H2.2)}$$

Here's a narrative for this example.

$$\text{Initially}_{\text{P}}\text{(Hungry(Fred))} \qquad \text{(H3.1)}$$

$$\text{Initially}_{\text{N}}\text{(Cold(Fred))} \qquad \text{(H3.2)}$$

$$\text{Happens(Feed(Fred),10)} \qquad \text{(H3.3)}$$

Finally we have the customary uniqueness-of-names axioms.

$$\text{UNA[Feed, Clothe]} \qquad \text{(H4.1)}$$

$$\text{UNA[Hungry, Cold]} \qquad \text{(H4.2)}$$

The incorporation of state constraints has negligible impact on the solution to the frame problem already presented. However, state constraints must be conjoined to the theory outside the scope of any of the circumscriptions. Given,

- a conjunction Σ of Initiates, Terminates and Releases formulae,
- a conjunction Δ of $\text{Initially}_{\text{P}}$, $\text{Initially}_{\text{N}}$, Happens and temporal ordering formulae,
- a conjunction Ψ of state constraints, and
- a conjunction Ω of uniqueness-of-names axioms for actions and fluents,

we're interested in,

$$\text{CIRC}[\Sigma\ ;\ \text{Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta\ ;\ \text{Happens}] \wedge \text{EC} \wedge \Psi \wedge \Omega.$$

For the current example, if we let Σ be the conjunction of (H2.1) and (H2.2), Δ be the conjunction of (H3.1) to (H3.3), Ψ be (H1.1), and Ω be the conjunction of (H4.1) and (H4.2), we have,

$$\text{CIRC}[\Sigma\ ;\ \text{Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta\ ;\ \text{Happens}] \wedge \text{EC} \wedge \Psi \wedge \Omega \vDash \text{HoldsAt(Happy(Fred),11)}.$$

State constraints must be used with caution. As can be seen by inspection, Axioms (EC1) to (EC7) enforce the following principle: *a fluent that has been initiated/terminated directly through an effect axiom cannot then be terminated/initiated indirectly through a state constraint, unless it is released beforehand*. Similarly, a fluent that holds at time 0 because of an $\text{Initially}_{\text{P}}$ formula cannot then be terminated indirectly through a state constraint, unless it's released beforehand, and a fluent that does not hold at time 0 because of an $\text{Initially}_{\text{N}}$ formula cannot then be initiated indirectly through a state constraint, unless it's released beforehand.

Suppose, in the present example, we introduced an Upset(x) event whose effect is to terminate Happy(x). Then the addition of Happens(Upset(Fred),12) would lead to contradiction. Similarly, the addition of $\text{Initially}_\text{N}$(Happy(Fred)) would lead to contradiction.

State constraints are most useful when there is a clear division of fluents into *primitive* and *derived*. Effect axioms are used to describe the dynamics of the primitive fluents and state constraints are used to describe the derived fluents in terms of the primitive ones.

3.4 Actions with Non-Deterministic Effects

The full event calculus can also be used to represent actions with non-deterministic effects. There are several different ways to do this. Here we'll confine our attention to the method of *determining fluents*. Some discussion of other techniques can be found in [Shanahan, 1997a]. A determining fluent is one which is not subject to the common sense law of inertia, yet whose value determines whether or not some other fluent is initiated or terminated by an event.

For example, suppose we have an action Toss, which non-deterministically results in either Heads holding or Heads not holding. (Tails could be defined as not Heads, but we don't need a Tails fluent for the examples.) To formalise the Toss action, we introduce a determining fluent, ItsHeads. ItsHeads is never initiated or terminated by an event, and is therefore not subject to the common sense law of inertia. We have the following effect axioms.

$$\text{Initiates(Toss,Heads,t)} \leftarrow \text{HoldsAt(ItsHeads,t)} \tag{C1.1}$$

$$\text{Terminates(Toss,Heads,t)} \leftarrow \neg\, \text{HoldsAt(ItsHeads,t)} \tag{C1.2}$$

Now suppose a series of Toss actions is performed.

$$\text{Initially}_\text{P}\text{(Heads)} \tag{C2.1}$$

$$\text{Happens(Toss,10)} \tag{C2.2}$$

$$\text{Happens(Toss,20)} \tag{C2.3}$$

$$\text{Happens(Toss,30)} \tag{C2.4}$$

Since there's just one action, the only uniqueness-of-names axiom we need is for fluents.

$$\text{UNA[Heads, ItsHeads]} \tag{C3.1}$$

Let Σ be the conjunction of (C1.1) and (C1.2), Δ be the conjunction of (C2.1) to (C2.4), and Ω be (C3.1). Now, there are some models of,

$$\text{CIRC}[\Sigma \text{ ; Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta \text{ ; Happens}] \wedge \text{EC} \wedge \Omega$$

in which we have, for example,

$$\text{HoldsAt(Heads,15)} \wedge \neg\, \text{HoldsAt(Heads,25)} \wedge \text{HoldsAt(Heads,35)}$$

and others in which we have, for example,

$$\neg\, \text{HoldsAt(Heads,15)} \wedge \text{HoldsAt(Heads,25)} \wedge \neg\, \text{HoldsAt(Heads,35)}.$$

However, in all models, the Heads fluent retains its value from one Toss event to the next, as we would expect.

Here's a variation on this example due to Ray Reiter. Suppose we throw a coin onto a chess board. Before this action, the coin isn't touching any squares, but when it comes to rest on the chess board, it could be touching just a white square, it could be touching just a black square, or it could be touching both. This example exposes flaws in attempts to solve the frame problem which naively minimise the change brought about by an action. Such formalisms are prone to reject the possibility of the coin touching both black and white squares, as this is a non-minimal change. But the following event calculus formalisation, using the determining fluents ItsBlack and ItsWhite, works fine.

Initiates(Throw,OnWhite,t) ← HoldsAt(ItsWhite,t) (R1.1)

Initiates(Throw,OnBlack,t) ← HoldsAt(ItsBlack,t) (R1.2)

Initially_N(OnWhite) (R2.1)

Initially_N(OnBlack) (R2.2)

Happens(Throw,10) (R2.3)

HoldsAt(ItsWhite,t) ∨ HoldsAt(ItsBlack,t) (R3.1)

UNA[OnWhite, OnBlack, ItsWhite, ItsBlack] (R4.1)

Let Σ be the conjunction of (R1.1) and (R1.2), Δ be the conjunction of (R2.1) to (R2.3), let Ψ be (R3.1), and Ω be (R3.1). As we would expect, in some models of,

CIRC[Σ ; Initiates, Terminates, Releases] ∧ CIRC[Δ ; Happens] ∧ EC ∧ Ψ ∧ Ω

we have, for example,

HoldsAt(OnWhite,15) ∧ ¬ HoldsAt(OnBlack,15)

while in others we have,

HoldsAt(OnWhite,15) ∧ HoldsAt(OnBlack,15).

In all models at least one of the fluents OnBlack or OnWhite holds after time 10, and in all models these fluents retain their values forever after time 10.

3.5 Compound Actions

The final topic for this section is compound actions, that is to say actions which are composed of other actions. These are particularly useful in hierarchical planning (see [Shanahan, 1997b]). Let's take a look at an example of a compound action definition describing a commuter's daily journey. Suppose we have two atomic actions: WalkTo(x) and TrainTo(x), whose effects are described by the following formulae.

Initiates(WalkTo(x),At(x),t) (J1.1)

Terminates(WalkTo(x),At(y),t) ← HoldsAt(At(y),t) ∧ x ≠ y (J1.2)

Initiates(TrainTo(x),At(x),t) ← HoldsAt(At(y),t) ∧ Train(y,x) (J1.3)

Terminates(TrainTo(x),At(y),t) ← HoldsAt(At(y),t) ∧ Train(y,x) (J1.4)

There are trains from Herne Hill to Victoria and from Victoria to South Kensington.

Train(HerneHill,Victoria) (J1.5)

Train(Victoria,SouthKen) (J1.6)

The following is a flawed example of a compound event definition describing a compound action, GoToWork, in terms of a sequence of WalkTo and TrainTo sub-actions.

Happens(GoToWork,t1,t4) ←
  Happens(WalkTo(HerneHill),t1) ∧ Happens(TrainTo(Victoria),t2) ∧
    Happens(TrainTo(SouthKen),t3) ∧ Happens(WalkTo(Work),t4) ∧
      t1 < t2 ∧ t2 < t3 ∧ t3 < t4

This formula is problematic for the following reason. Normally, in hierarchical planning for example, we would expect to be able to work out the effects of a compound action given the effects of its sub-actions. As it stands, this formula doesn't allow this, as it doesn't exclude the possibility that other events occur in between the sub-events mentioned in the definition, which undo the effects of those sub-events. For example, if I'm arrested at Herne Hill station and taken away by the police, then the

TrainTo(Victoria) action will be ineffective, and the GoToWork action won't have its expected outcome. Here's a modified form of the formula incorporating extra $\neg$ Clipped conditions that rule out intervening events.

$$\begin{aligned}&\text{Happens}(\text{GoToWork},t1,t4) \leftarrow \qquad\qquad\qquad\qquad \text{(J2.1)}\\ &\quad \text{Happens}(\text{WalkTo}(\text{HerneHill}),t1) \wedge \text{Happens}(\text{TrainTo}(\text{Victoria}),t2) \wedge\\ &\qquad \text{Happens}(\text{TrainTo}(\text{SouthKen}),t3) \wedge \text{Happens}(\text{WalkTo}(\text{Work}),t4) \wedge\\ &\qquad\quad t1 < t2 \wedge t2 < t3 \wedge t3 < t4 \wedge \neg \text{Clipped}(t1,\text{At}(\text{HerneHill}),t2) \wedge\\ &\qquad\qquad \neg \text{Clipped}(t2,\text{At}(\text{Victoria}),t3) \wedge \neg \text{Clipped}(t3,\text{At}(\text{SouthKen}),t4)\end{aligned}$$

Now, given (J1.1) to (J1.4), we can confidently write the following effect axioms.

$$\text{Initiates}(\text{GoToWork},\text{At}(\text{Work}),t) \qquad \text{(J3.1)}$$

$$\text{Terminates}(\text{GoToWork},\text{At}(x),t) \leftarrow \text{HoldsAt}(\text{At}(x),t) \wedge x \neq \text{Work} \qquad \text{(J3.2)}$$

The only required uniqueness-of-names axiom is for actions.

$$\text{UNA}[\text{WalkTo, TrainTo, GoToWork}] \qquad \text{(J4.1)}$$

Now consider the following narrative of actions.

$$\text{Happens}(\text{WalkTo}(\text{HerneHill}),10) \qquad \text{(J5.1)}$$

$$\text{Happens}(\text{TrainTo}(\text{Victoria}),15) \qquad \text{(J5.2)}$$

$$\text{Happens}(\text{TrainTo}(\text{SouthKen}),20) \qquad \text{(J5.3)}$$

$$\text{Happens}(\text{WalkTo}(\text{Work}),25) \qquad \text{(J5.4)}$$

Let Σ be the conjunction of (J1.1) to (J1.6) plus (J2.1). Let Δ be the conjunction of (J5.1) to (J5.4), and Ω be (J4.1). Notice that (J3.1) and (J3.2) have been omitted. We have,[5]

$$\begin{aligned}&\text{CIRC}[\Sigma ; \text{Initiates, Terminates, Releases}] \wedge\\ &\quad \text{CIRC}[\Delta ; \text{Happens}] \wedge \text{EC} \wedge \Omega \vDash \text{Happens}(\text{GoToWork},10,25)\end{aligned}$$

and,

$$\begin{aligned}&\text{CIRC}[\Sigma ; \text{Initiates, Terminates, Releases}] \wedge\\ &\quad \text{CIRC}[\Delta ; \text{Happens}] \wedge \text{EC} \wedge \Omega \vDash \text{HoldsAt}(\text{At}(\text{Work}),30)\end{aligned}$$

The inclusion of (J3.1) and (J3.2) would yield the same logical consequences.

Although not illustrated in this small example, it's worth noting that both conditional and recursive compound action definitions are also possible. Further discussion of compound events can be found in [Shanahan, 1997b], which also includes examples featuring such standard program constructs.

4 The Ramification Problem

As already mentioned, state constraints aren't the only way to represent actions with indirect effects, and often they aren't the right way. To see this, we'll take a look at the so-called "walking turkey shoot" [Baker, 1991], a variation of the Yale shooting problem in which the Shoot action, as well as directly terminating the Alive fluent, indirectly terminates a fluent Walking. The effect axioms are inherited from the Yale shooting problem.

$$\text{Initiates}(\text{Load},\text{Loaded},t) \qquad \text{(W1.1)}$$

$$\text{Terminates}(\text{Shoot},\text{Alive},t) \leftarrow \text{HoldsAt}(\text{Loaded},t) \qquad \text{(W1.2)}$$

The narrative of events is as follows.

---

[5] Examples with compound actions are among the few useful cases of event calculus formulae that don't reduce straightforwardly to predicate completion. Examples involving recursion are especially tricky.

$\text{Initially}_P(\text{Alive})$ (W2.1)

$\text{Initially}_P(\text{Loaded})$ (W2.2)

$\text{Initially}_P(\text{Walking})$ (W2.3)

Happens(Shoot,T1) (W2.4)

T1 < T2 (W2.5)

We have two uniqueness-of-names axioms.

UNA[Load, Shoot] (W3.1)

UNA[Loaded, Alive, Walking] (W3.2)

Now, how do we represent the dependency between the Walking and Alive fluents so as to get the required indirect effect of a Shoot action? The obvious, but incorrect, way is to use a state constraint.

$\text{HoldsAt}(\text{Alive},t) \leftarrow \text{HoldsAt}(\text{Walking},t)$

The addition of this state constraint to the above formalisation would yield a contradiction, because it violates the rule that a fluent, in this case Walking, that holds directly through an Initially_P formula cannot be terminated indirectly through a state constraint. (The same problem would arise if the Walking fluent had been initiated directly by an action.)

4.1 Effect Constraints

Instead, the way to represent the relationship between the Walking fluent and the Alive fluent in the walking turkey shoot is through an *effect constraint*. Effect constraints are Initiates and Terminates formulae with a single universally quantified action variable. The constraint we require for this example is the following.

$\text{Terminates}(a,\text{Walking},t) \leftarrow \text{Terminates}(a,\text{Alive},t)$ (W4.1)

Notice that effect constraints are weaker than state constraints: the possibility of resurrecting a corpse by making it walk, inherent in the faulty state constraint, is not inherent in this formula.

Let Σ be the conjunction of (W1.1), (W1.2) and (W4.1). Let Δ be the conjunction of (W2.1) to (W2.5), and Ω be the conjunction of (W3.1) and (W3.2). We have,

$$\text{CIRC}[\Sigma \,;\, \text{Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta \,;\, \text{Happens}] \wedge \text{EC} \wedge \Omega \vDash \neg \text{HoldsAt}(\text{Walking},\text{T2}).$$

Effect constraints can be used to represent a number of other standard benchmarks for the ramification problem. However, there remain certain examples for which they're unsuited, specifically those involving the instantaneous propagation of interacting indirect effects. Fortunately, these can be handled by *causal constraints*, as set out in the next section, which draws on techniques presented in [Shanahan, 1999].

4.2 Causal Constraints

The circuit of Figure 3 illustrates the instantaneous propagation of interacting indirect effects [Thielscher, 1997]. Closing switch 1 activates the relay, in turn opening switch 2, thereby preventing the light from coming on.

To represent examples like this, we introduce several new predicates. The formula $\text{Started}(\beta,\tau)$ means that either β already holds at τ or an event occurs at τ that initiates β. Conversely, the formula $\text{Stopped}(\beta,\tau)$ means that either β already does not hold at τ or an event occurs at τ that terminates β. The predicates Started and Stopped are defined by the following axioms, which will be conjoined to our theories outside the scope of any of the circumscriptions.

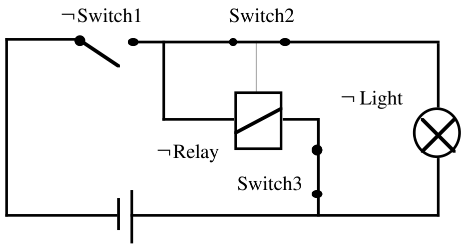

Figure 3: Thielscher's Circuit

$\text{Started}(f,t) \leftrightarrow$ (CC1)
$\quad \text{HoldsAt}(f,t) \vee \exists\, a\, [\text{Happens}(a,t) \wedge \text{Initiates}(a,f,t)]$

$\text{Stopped}(f,t) \leftrightarrow$ (CC2)
$\quad \neg\, \text{HoldsAt}(f,t) \vee \exists\, a\, [\text{Happens}(a,t) \wedge \text{Terminates}(a,f,t)]$

The formula Initiated(β,τ) means that fluent β either already holds at τ or is about to start holding. Similarly Terminated(β,τ) represents that β either already does not hold at τ or is about to cease holding at τ. These predicates are defined as follows.

$\text{Initiated}(f,t) \leftrightarrow$ (CC3)
$\quad \text{Started}(f,t) \wedge \neg \exists\, a\, [\text{Happens}(a,t) \wedge \text{Terminates}(a,f,t)]$

$\text{Terminated}(f,t) \leftrightarrow$ (CC4)
$\quad \text{Stopped}(f,t) \wedge \neg \exists\, a\, [\text{Happens}(a,t) \wedge \text{Initiates}(a,f,t)]$

To represent the dependencies between the fluents in Thielscher's circuit example, we introduce three events LightOn, Open2 and CloseRelay, which are triggered under conditions described by the following formulae.

$\text{Happens}(\text{LightOn},t) \leftarrow$ (L1.1)
$\quad \text{Stopped}(\text{Light},t) \wedge \text{Initiated}(\text{Switch1},t) \wedge \text{Initiated}(\text{Switch2},t)$

$\text{Happens}(\text{Open2},t) \leftarrow$ (L1.2)
$\quad \text{Started}(\text{Switch2},t) \wedge \text{Initiated}(\text{Relay},t)$

$\text{Happens}(\text{CloseRelay},t) \leftarrow$ (L1.3)
$\quad \text{Stopped}(\text{Relay},t) \wedge \text{Initiated}(\text{Switch1},t) \wedge \text{Initiated}(\text{Switch3},t)$

These formulae represent *causal constraints*. If a fluent is dependent on a number of other fluents, such formulae ensure that an event giving that fluent the right value is triggered whenever the fluents that influence it attain the relevant values. The effects of the new events in this example are as follows. A Close1 event is also introduced.

$\text{Initiates}(\text{LightOn},\text{Light},t)$ (L2.1)

$\text{Terminates}(\text{Open2},\text{Switch2},t)$ (L2.2)

$\text{Initiates}(\text{CloseRelay},\text{Relay},t)$ (L2.3)

$\text{Initiates}(\text{Close1},\text{Switch1},t)$ (L2.4)

The circuit's initial configuration, as shown in Figure 3, is as follows.

$\text{Initially}_N(\text{Switch1})$ (L3.1)

$\text{Initially}_P(\text{Switch2})$ (L3.2)

$\text{Initially}_P(\text{Switch3})$ (L3.3)

$\text{Initially}_N(\text{Relay})$ (L3.4)

$\text{Initially}_N(\text{Light})$ (L3.5)

The only event that occurs is a Close1 event, at time 10.

Happens(Close1,10) (L3.6)

Two uniqueness-of-names axioms are required.

UNA[LightOn, Close1, Open2, CloseRelay] (L4.1)

UNA[Switch1, Switch2, Switch3, Relay, Light] (L4.2)

Now let Σ be the conjunction of (L2.1) to (L2.4), Δ be the conjunction of (L1.1) to (L1.3) with (L3.1) to (L3.6), Ψ be the conjunction of (CC1) to (CC4), and Ω be the conjunction of (L4.1) and (L4.2). We have,

CIRC[Σ ; Initiates, Terminates, Releases] $\wedge$
  CIRC[Δ ; Happens] $\wedge$ EC $\wedge$ Ψ $\wedge$ Ω $\models$
    HoldsAt(Relay,20) $\wedge$ $\neg$ HoldsAt(Switch2,20) $\wedge$ $\neg$ HoldsAt(Light,20).

In other words, this formalisation of Thielscher's circuit yields the logical consequences we require. In particular, the relay is activated when switch 1 is closed, causing switch 2 to open, and the light does not come on.

5 The Extended Event Calculus

This section shows how the full event calculus of Section 3 can be extended to represent concurrent actions and continuous change. The calculus is presented formally first, then two examples are given, one featuring concurrent action, the other featuring continuous change.

Table 3 describes those predicates used in the extended event calculus that weren't part of the full calculus of Section 3. Three new predicates are introduced. The predicates Cancels and Cancelled, as in [Gelfond, *et al.*, 1991] and [Lin & Shoham, 1992], cater for concurrent actions that interfere with each other's effects. The Cancels predicate will be minimised via circumscription, along with Initiates, Terminates and Releases. The Trajectory predicate, first proposed in [Shanahan, 1990], is used to capture continuous change, as in the height of a falling ball or the level of liquid in a filling vessel, for example.

Formula	Meaning
Cancels(α1,α2,β)	The occurrence of α1 cancels the effect of a simultaneous occurrence of α2 on fluent β
Cancelled(α,β,τ1,τ2)	Some event occurs from time τ1 to time τ2 which cancels the effect of action α on fluent β
Trajectory(β1,τ,β2,δ)	If fluent β1 is initiated at time τ then fluent β2 becomes true at time τ+δ

Table 3: Three More New Predicates

Here is the new set of axioms, whose conjunction will be denoted XC. The first seven axioms correspond to the seven axioms of the calculus of Section 3. The only difference is the incorporation in Axioms (XC2), (XC3), (XC5) and (XC6) of $\neg$ Cancelled conditions that block the applicability of the axiom in the case of the simultaneous occurrence of events which cancel each other's effects.

HoldsAt(f,t) $\leftarrow$ Initially_P(f) $\wedge$ $\neg$ Clipped(0,f,t) (XC1)

HoldsAt(f,t3) $\leftarrow$ (XC2)
  Happens(a,t1,t2) $\wedge$ Initiates(a,f,t1) $\wedge$ $\neg$ Cancelled(a,f,t1,t2) $\wedge$
    t2 < t3 $\wedge$ $\neg$ Clipped(t1,f,t3)

$$\text{Clipped}(t1,f,t4) \leftrightarrow \qquad \text{(XC3)}$$
$$\exists\, a,t2,t3\ [\text{Happens}(a,t2,t3) \wedge t1 < t3 \wedge t2 < t4 \wedge$$
$$[\text{Terminates}(a,f,t2) \vee \text{Releases}(a,f,t2)] \wedge$$
$$\neg\, \text{Cancelled}(a,f,t2,t3)]$$

$$\neg\, \text{HoldsAt}(f,t) \leftarrow \text{Initially}_N(f) \wedge \neg\, \text{Declipped}(0,f,t) \qquad \text{(XC4)}$$

$$\neg\, \text{HoldsAt}(f,t3) \leftarrow \qquad \text{(XC5)}$$
$$\text{Happens}(a,t1,t2) \wedge \text{Terminates}(a,f,t1) \wedge \neg\, \text{Cancelled}(a,f,t1,t2) \wedge$$
$$t2 < t3 \wedge \neg\, \text{Declipped}(t1,f,t3)$$

$$\text{Declipped}(t1,f,t4) \leftrightarrow \qquad \text{(XC6)}$$
$$\exists\, a,t2,t3\ [\text{Happens}(a,t2,t3) \wedge t1 < t3 \wedge t2 < t4 \wedge$$
$$[\text{Initiates}(a,f,t2) \vee \text{Releases}(a,f,t2)] \wedge$$
$$\neg\, \text{Cancelled}(a,f,t2,t3)]$$

$$\text{Happens}(a,t1,t2) \rightarrow t1 \leq t2 \qquad \text{(XC7)}$$

Axiom (XC8) defines the Cancelled predicate.

$$\text{Cancelled}(a1,f,t1,t2) \leftrightarrow \text{Happens}(a2,t1,t2) \wedge \text{Cancels}(a2,a1,f) \qquad \text{(XC8)}$$

Axiom (XC9) is the counterpart of Axiom (XC2) for continuous change.

$$\text{HoldsAt}(f2,t3) \leftarrow \qquad \text{(XC9)}$$
$$\text{Happens}(a,t1,t2) \wedge \text{Initiates}(a,f1,t1) \wedge \neg\, \text{Cancelled}(a,f,t1,t2) \wedge$$
$$t2 < t3 \wedge t3 = t2 + d \wedge \text{Trajectory}(f1,t1,f2,d) \wedge$$
$$\neg\, \text{Clipped}(t1,f1,t3)$$

As before, a two-argument Happens is defined in terms of the three-argument version.

$$\text{Happens}(a,t) \equiv_{def} \text{Happens}(a,t,t)$$

In addition to the three new predicates introduced above, the extended event calculus employs a new infix function symbol &, which will be used to express the cumulative effects of concurrent actions. The term $\alpha 1\&\alpha 2$ denotes a compound action comprising the two actions $\alpha 1$ and $\alpha 2$. We write Happens(a1&a2,$\tau 1$,$\tau 2$) to denote that actions $\alpha 1$ and $\alpha 2$ occur concurrently, that is to say they both start at $\tau 1$ and end at $\tau 2$. The final new axiom we require defines the & symbol.

$$\text{Happens}(a1\&a2,t1,t2) \leftarrow \text{Happens}(a1,t1,t2) \wedge \text{Happens}(a2,t1,t2) \qquad \text{(CA)}$$

The circumscriptive approach to the frame problem employed before extends straightforwardly to the new calculus. Since it constrains the Happens predicate, Axiom (CA) must be included inside the circumscription that minimises Happens. In general, given,

- a conjunction Σ of Initiates, Terminates, Releases, Trajectory and Cancels formulae,
- a conjunction Δ of Initially_P, Initially_N, Happens and temporal ordering formulae,
- a conjunction Ψ of state constraints, and
- a conjunction Ω of uniqueness-of-names axioms for actions and fluents,

we're interested in,

$$\text{CIRC}[\Sigma\ ;\ \text{Initiates, Terminates, Releases, Cancels}] \wedge$$
$$\text{CIRC}[\Delta \wedge \text{(CA)}\ ;\ \text{Happens}] \wedge \text{XC} \wedge \Psi \wedge \Omega.$$

Ψ is omitted if there are no state constraints.

If Cancels and Trajectory are everywhere false, then Axioms (EC1) to (EC7) follow from Axioms (XC1) to (XC9). Accordingly, the examples already presented in this article to illustrate the simple event calculus and the full event calculus also work with the extended event calculus.

The next two sections comprise examples of the use of the extended event calculus to deal with concurrent action and continuous change.

5.1 Concurrent Actions

This section formalises the soup bowl scenario from [Gelfond, *et al.*, 1991]. This example features concurrent actions with both *cumulative* and *cancelling* effects. The domain comprises two actions, LiftLeft and LiftRight, which represent respectively lifting the left side of a soup bowl and lifting the right side. Two fluents are involved: Spilled and OnTable. The soup bowl is full of soup. So a LiftLeft action on its own will initiate Spilled, as will a LiftRight action on its own. Carried out together, though, these actions cancel each other's effect on the Spilled fluent. On the other hand, carried out together, a LiftLeft action and a LiftRight action have a cumulative effect, namely to raise the bowl from the table, terminating the OnTable fluent. We have the following Initiates and Terminates formulae.

Initiates(LiftLeft,Spilled,s) (B1.1)

Initiates(LiftRight,Spilled,s) (B1.2)

Terminates(LiftLeft&LiftRight,OnTable,s) (B1.3)

Here are the required Cancels formulae.

Cancels(LiftLeft,LiftRight,Spilled) (B2.1)

Cancels(LiftRight,LiftLeft,Spilled) (B2.2)

In the initial situation, the soup bowl is on the table, and there has been no spillage. At time 10, a LiftLeft action and a LiftRight action occur simultaneously.

$\text{Initially}_P(\text{OnTable})$ (B3.1)

$\text{Initially}_N(\text{Spilled})$ (B3.2)

Happens(LiftLeft,10) (B3.4)

Happens(LiftRight,10) (B3.5)

Here are the customary uniqueness-of-names axioms.

UNA[OnTable, Spilled] (B4.1)

UNA[LiftLeft, LiftRight] (B4.2)

Now let Σ be the conjunction of (B1.1) to (B1.3) with (B2.1) and (B2.2), Δ be the conjunction of (B3.1) to (B3.4), and Ω be the conjunction of (B4.1) and (B4.2). We have,

$$\begin{aligned}&\text{CIRC}[\Sigma \text{ ; Initiates, Terminates, Releases, Cancels}] \wedge \\ &\quad \text{CIRC}[\Delta \wedge (\text{CA}) \text{ ; Happens}] \wedge \text{XC} \wedge \Omega \vDash \\ &\qquad \neg \text{HoldsAt}(\text{OnTable},20) \wedge \neg \text{HoldsAt}(\text{Spilled},20).\end{aligned}$$

In other words, the formalisation yields the desired conclusion that the bowl is no longer on the table at time 20, but in spite of the occurrence of a LiftLeft and a LiftRight action, the soup has not been spilled.

5.2 Continuous Change

This section demonstrates how the extended calculus copes with continuous change, via an example involving a vessel that fills with water. The example also features *triggered events*, that is to say events that occur when certain fluents reach certain values. These are similar to the events that are used to represent causal constraints in Section 4.2. But in the present case, the event is triggered when a continuously varying quantity attains a particular value, specifically when the water level reaches the rim of the vessel.

The domain comprises a TapOn event, which initiates a flow of liquid into the vessel. The fluent Filling holds while water is flowing into the vessel, and the fluent Level(x) represents holds if the water is at level x in the vessel, where x is a real number. An Overflow event occurs when the water reaches the rim of the vessel at level 10. The Overflow event initiates a period during which the fluent Spilling holds. A TapOff action is also included. Here are the Initiates, Terminates and Releases formulae for the domain.

Initiates(TapOn,Filling,t) (S1.1)

Terminates(TapOff,Filling,t) (S1.2)

Releases(TapOn,Level(x),t) (S1.3)

Initiates(TapOff,Level(x),t) $\leftarrow$ HoldsAt(Level(x),t) (S1.4)

Terminates(Overflow,Filling,t) (S1.5)

Initiates(Overflow,Level(10),t) (S1.6)

Initiates(Overflow,Spilling,t) (S1.7)

Note that (S1.3) has to be a Releases formula instead of a Terminates formula, so that the Level fluent is immune from the common sense law of inertia after the tap is turned on.

Now we have the Trajectory formula, which describes the continuous variation in the Level fluent while the Filling fluent holds. The level is assumed to rise at one unit per unit of time.

Trajectory(Filling,t,Level(x2),d) $\leftarrow$ (S1.8)
HoldsAt(Level(x1),t) $\wedge$ x2 = x1 + d

Next we have a state constraint that ensures that the water always has a unique level.

HoldsAt(Level(x1),t) $\wedge$ HoldsAt(Level(x2),t) $\rightarrow$ x1 = x2 (S2.1)

The next formulae ensures the Overflow event is triggered when it should be.

Happens(Overflow,t) $\leftarrow$ (S3.1)
HoldsAt(Level(10),t) $\wedge$ HoldsAt(Filling,t)

Here's a simple narrative. The level is initially 0, and the tap is turned on at time 5.

Initially_P(Level(0)) (S4.1)

Initially_N(Filling) (S4.2)

Initially_N(Spilling) (S4.3)

Happens(TapOn,5) (S4.4)

The following uniqueness-of-names axioms are required.

UNA[TapOn, TapOff, Overflow] (S5.1)

UNA[Filling, Level, Spilling] (S5.2)

Let Σ be the conjunction of (S1.1) to (S1.8), Δ be the conjunction of (S4.1) to (S4.4) with (S3.1), Ψ be the (S2.1), and Ω be the conjunction of (S5.1) and (S5.2). We have,

CIRC[Σ ; Initiates, Terminates, Releases, Cancels] $\wedge$
CIRC[$\Delta \wedge$ (CA) ; Happens] $\wedge$ XC $\wedge\ \Psi \wedge \Omega \vDash$
HoldsAt(Level(10),20) $\wedge \neg$ HoldsAt(Filling,20) $\wedge$ HoldsAt(Spilling,20).

In other words, the formalisation yields the expected result that the water stops flowing into the vessel (at time 15), when it starts spilling over the rim, and that the level is subsequently stuck at 10.

The Trajectory predicate can be used to represent a large number of problems involving continuous change. But for a more general treatment, in which arbitrary sets of differential equations can be deployed, see [Miller & Shanahan, 1996].

Concluding Remarks

The extended event calculus of the last section is a formalism for reasoning about action that incorporates a simple solution to the frame problem, and is capable of representing a diverse range of phenomena. These phenomena include,

- actions with indirect effects, including interacting indirect effects as in Thielscher's circuit example,
- actions with non-deterministic effects, including examples with non-minimal change such as Reiter's chess-board example,
- compound actions, which can include standard programming constructs such as sequence, choice and recursion,
- concurrent actions, including actions with cumulative and cancelling effects, as in the soup bowl example, and
- continuous change with triggered events, as in the filling vessel example.

Nothing has been said so far about *explanation*, that is to say reasoning from effects to causes, which is isomorphic to *planning*. The logical aspects of this topic are dealt with in Chapter 17 of [Shanahan, 1997a], where it is shown that explanation (or planning) problems can be handled via abduction. In [Shanahan, 1997b], an implementation of abductive event calculus planning is presented, which will also perform explanation. This implementation also forms the basis of a system used to control a robot [Shanahan, 1998], in which sensor data assimilation is also cast as a form of abductive reasoning with the event calculus [Shanahan, 1996].

Acknowledgments

This work was carried out as part of the EPSRC funded project GR/L20023 "Cognitive Robotics". Thanks to all those members of the reasoning about action community whose work has influenced the development of the event calculus.

References

[Baker, 1991] A.B.Baker, Nonmonotonic Reasoning in the Framework of the Situation Calculus, *Artificial Intelligence*, vol. 49 (1991), pp. 5–23.

[Crawford & Etherington, 1992] J.M.Crawford and D.W.Etherington, Formalizing Reasoning about Change: A Qualitative Reasoning Approach, *Proceedings AAAI 92*, pp. 577–583.

[Doherty, 1994] P.Doherty, Reasoning about Action and Change Using Occlusion, *Proceedings ECAI 94*, pp. 401–405.

[Gelfond, *et al.*, 1991] M.Gelfond, V.Lifschitz and A.Rabinov, What Are the Limitations of the Situation Calculus? in *Essays in Honor of Woody Bledsoe*, ed R.Boyer, Kluwer Academic (1991), pp. 167–179.

[Haas, 1987] A.R.Haas, The Case for Domain-Specific Frame Axioms, *Proceedings of the 1987 Workshop on the Frame Problem*, pp. 343–348.

[Hanks & McDermott, 1987] S.Hanks and D.McDermott, Nonmonotonic Logic and Temporal Projection, *Artificial Intelligence*, vol. 33 (1987), pp. 379–412.

[Kartha & Lifschitz, 1994] G.N.Kartha and V.Lifschitz, Actions with Indirect Effects (Preliminary Report), *Proceedings 1994 Knowledge Representation Conference (KR 94)*, pp. 341–350.

[Kartha & Lifschitz, 1995] G.N.Kartha and V.Lifschitz, A Simple Formalization of Actions Using Circumscription, *Proceedings IJCAI 95*, pp. 1970–1975.

[Kowalski, 1992] R.A.Kowalski, Database Updates in the Event Calculus, *Journal of Logic Programming*, vol. 12 (1992), pp. 121–146.

[Kowalski & Sergot, 1986] R.A.Kowalski and M.J.Sergot, A Logic-Based Calculus of Events, *New Generation Computing*, vol. 4 (1986), pp. 67–95.

[Lifschitz, 1994] V.Lifschitz, Circumscription, in *The Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, ed. D.M.Gabbay, C.J.Hogger and J.A.Robinson, Oxford University Press (1994), pp. 297–352.

[Lin & Shoham, 1992] F.Lin and Y.Shoham, Concurrent Actions in the Situation Calculus, *Proceedings AAAI 92*, pp. 590–595.

[McCarthy, 1980] J.McCarthy, Circumscription — A Form of Non-Monotonic Reasoning, *Artificial Intelligence*, vol. 13 (1980), pp. 27–39.

[McCarthy, 1988] J.McCarthy, Mathematical Logic in Artificial Intelligence, *Daedalus*, Winter 1988, pp. 297–311.

[McCarthy & Hayes, 1969] J.McCarthy and P.J.Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, in *Machine Intelligence 4*, ed. D.Michie and B.Meltzer, Edinburgh University Press (1969), pp. 463–502.

[Miller & Shanahan, 1996] R.S.Miller and M.P.Shanahan, Reasoning about Discontinuities in the Event Calculus, *Proceedings 1996 Knowledge Representation Conference (KR 96)*, pp. 63–74.

[Reiter, 1980] R.Reiter, A Logic for Default Reasoning, *Artificial Intelligence*, vol. 13 (1980), pp. 81–132.

[Reiter, 1991] R.Reiter, The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression, in *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, ed. V.Lifschitz, Academic Press (1991), pp. 359–380.

[Sandewall, 1991] E.Sandewall, *Features and Fluents*, Technical Report LiTH-IDA-R-91-29 (first review version), Department of Computer and Information Science, Linköping University, Sweden, 1991.

[Sandewall, 1994] E.Sandewall, *Features and Fluents: The Representation of Knowledge about Dynamical Systems, Volume 1*, Oxford University Press (1994).

[Schubert, 1990] L.K.Schubert, Monotonic Solution of the Frame Problem in the Situation Calculus, in *Knowledge Representation and Defeasible Reasoning*, ed. H.Kyburg, R.Loui and G.Carlson, Kluwer (1990), pp. 23–67.

[Shanahan, 1990] M.P.Shanahan, Representing Continuous Change in the Event Calculus, *Proceedings ECAI 90*, pp. 598–603.

[Shanahan, 1996] M.P.Shanahan, Robotics and the Common Sense Informatic Situation, *Proceedings ECAI 96*, pp. 684–688.

[Shanahan, 1997a] M.P.Shanahan, *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*, MIT Press, 1997.

[Shanahan, 1997b] M.P.Shanahan, Event Calculus Planning Revisited, *Proceedings 4th European Conference on Planning (ECP 97)*, Springer Lecture Notes in Artificial Intelligence no. 1348 (1997), pp. 390–402.

[Shanahan, 1998] M.P.Shanahan, Reinventing Shakey, *Working Notes of the 1998 AAAI Fall Symposium on Cognitive Robotics*, pp. 125–135.

[Shanahan, 1999] M.P.Shanahan, The Ramification Problem in the Event Calculus, *Proceedings IJCAI 99*, to appear.

[Thielscher, 1997] M.Thielscher, Ramification and Causality, *Artificial Intelligence*, vol. 89 (1997), pp. 317–364.