Some Alternative Formulations of the Event Calculus

Rob Miller[1] and Murray Shanahan[2]

[1] University College London, London WC1E 6BT, U.K.
rsm@ucl.ac.uk
http://www.ucl.ac.uk/~uczcrsm/
[2] Imperial College of Science, Technology and Medicine, London SW7 2BT, U.K.
m.shanahan@ic.ac.uk
http://www-ics.ee.ic.ac.uk/~mpsha/

Abstract. The Event Calculus is a narrative based formalism for reasoning about actions and change originally proposed in logic programming form by Kowalski and Sergot. In this paper we summarise how variants of the Event Calculus may be expressed as classical logic axiomatisations, and how under certain circumstances these theories may be reformulated as "action description language" domain descriptions using the Language $\mathcal{E}$. This enables the classical logic Event Calculus to inherit various provably correct automated reasoning procedures recently developed for $\mathcal{E}$.

1 Introduction

The "Event Calculus" was originally introduced by Bob Kowalski and Marek Sergot [33] as a logic programming framework for representing and reasoning about actions (or events) and their effects, especially in database applications. Since then many alternative formulations, implementations and applications have been developed. The Event Calculus has been reformulated in various logic programming forms (e.g. [11], [12], [21], [23], [29], [53], [58], [71], [72], [73],[74]), in classical logic (e.g. [62], [42], [43]), in modal logic (e.g. [2], [3], [4], [5], [6], [7]) and as an "action description language" ([21], [22]). In one form or another it has been extended and applied, for example, in the context of planning (e.g. [15], [8], [19], [44], [45], [63], [65], [20]), cognitive robotics (e.g. [60], [61], [65], [67]), abductive reasoning (e.g. [11], [44], [45], [71] and [72]), database updates (e.g. [29], [72]), accident report processing [35], legal reasoning [30], modelling continuous change and mathematical modelling (e.g. [42], [58], [71]), modelling and reasoning about agent beliefs [35], reasoning about programming constructs [10, 68], and software engineering [52].

In spite of this growing menagerie of Event Calculus formulations and applications, relatively little work has been done to show how the various versions correspond. (Indeed, much more work has been done on showing how the Event Calculus corresponds to the Situation Calculus, see e.g. [21], [31], [32], [41], [48], [49], [73], [74].) This article is an attempt to begin to address this issue. We first

summarise recent work (e.g. [62], [42], [43]) on axiomatising the Event Calculus in classical logic, using circumscription as a method for default reasoning to solve the frame and related problems. We then describe how under certain circumstances such classical logic theories may be reformulated as "action description language" domain descriptions using the Language $\mathcal{E}$ [21, 22]. This enables the classical logic Event Calculus to inherit various provably correct, logic programming and/or argumentation based automated reasoning procedures developed for $\mathcal{E}$ in [21], [22], [23] and [24].

Even if attention is restricted to classical logic formulations of the Event Calculus, there are a number of different choices or variations for the core set of axioms. The various alternatives are each geared to classes of domains with particular restrictions or features; for example to describe systems most naturally viewed as deterministic, as involving both continuous and discrete change, or which require reasoning about the future but not the past. In view of this, we first present (in Section 2) one particular (basic) form of the Event Calculus with six domain independent axioms labeled (EC1) to (EC6), and then (Section 3) list and motivate some alternatives. When describing these, possible substitutes for (for example) axiom (EC1) are labeled (EC1a), (EC1b), etc.

A central feature of the Event Calculi presented here are that they are *narrative-based*, i.e. a time structure which is independent of any action occurrences is established or assumed, and then statements about when various actions occur within this structure are incorporated in the description of the domain under consideration. The time structure is usually assumed or stated to be linear – typically the real or integer number line – although the underlying ideas can equally be applied to other (possibly branching) temporal structures. For the purposes of simplicity, unless otherwise stated we will assume in this article that time is represented either by the real numbers, the integers, the non-negative reals or the non-negative integers, and that appropriate axioms are included in the theory which establish one of these time structures.

Sections 2 and 3 of this article are mostly taken from [43].

2 A Classical Logic Event Calculus Axiomatisation

Informally, the basic idea of the Event Calculus is to state that *fluents* (time-varying properties of the world) are *true* at particular time-points if they have been *initiated* by an action occurrence at some earlier time-point, and not *terminated* by another action occurrence in the meantime. Similarly, a fluent is *false* at a particular time-point if it has been previously terminated and not initiated in the meantime. Domain dependent axioms are provided to describe which actions initiate and terminate which fluents under various circumstances, and to state which actions occur when. In the context of the Event Calculus, individual action occurrences are often referred to as "events", so that "actions" are "event types".

The Event Calculus given here is written in a sorted predicate calculus with equality, with a sort $\mathcal{A}$ for *actions* (variables $a, a_1, a_2, \ldots$), a sort $\mathcal{F}$ for flu-

ents (variables $f, f_1, f_2, \ldots$), a sort $\mathcal{T}$ for timepoints (here either real numbers or integers, variables $t, t_1, t_2, \ldots$) and a sort $\mathcal{X}$ for domain objects (variables $x, x_1, x_2, \ldots$). To describe a very basic calculus we need five predicates (other than equality); $\mathit{Happens} \subseteq \mathcal{A} \times \mathcal{T}$, $\mathit{HoldsAt} \subseteq \mathcal{F} \times \mathcal{T}$, $\mathit{Initiates} \subseteq \mathcal{A} \times \mathcal{F} \times \mathcal{T}$, $\mathit{Terminates} \subseteq \mathcal{A} \times \mathcal{F} \times \mathcal{T}$ and $< \subseteq \mathcal{T} \times \mathcal{T}$. $\mathit{Happens}(A, T)$ indicates that action A occurs at time T, $\mathit{HoldsAt}(F, T)$ means that fluent F is true at time T, and $\mathit{Initiates}(A, F, T)$ (respectively $\mathit{Terminates}(A, F, T)$) expresses that if A occurs at T it will initiate (respectively terminate) the fluent F. "$<$" is the standard order relation for time.

It is convenient to also define auxiliary predicates $\mathit{Clipped} \subseteq \mathcal{T} \times \mathcal{F} \times \mathcal{T}$ and $\mathit{Declipped} \subseteq \mathcal{T} \times \mathcal{F} \times \mathcal{T}$ in terms of *Happens*, *Initiates*, *Terminates*, and $<$. $\mathit{Clipped}(T_1, F, T_2)$ (respectively $\mathit{Declipped}(T_1, F, T_2)$) means "the fluent F is terminated (respectively initiated) between times T_1 and T_2." The corresponding definitional axioms[1] are

$$\mathit{Clipped}(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[\mathit{Happens}(a, t) \wedge t_1 \leq t < t_2 \wedge \mathit{Terminates}(a, f, t)] \quad \text{(EC1)}$$

$$\mathit{Declipped}(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[\mathit{Happens}(a, t) \wedge t_1 \leq t < t_2 \wedge \mathit{Initiates}(a, f, t)] \quad \text{(EC2)}$$

We can now axiomatise the two principles stated in the introduction to this section. Fluents which have been initiated by an occurrence of an action continue to hold until an occurrence of an action which terminates them

$$\mathit{HoldsAt}(f, t_2) \leftarrow [\mathit{Happens}(a, t_1) \wedge \mathit{Initiates}(a, f, t_1) \wedge t_1 < t_2 \wedge \neg \mathit{Clipped}(t_1, f, t_2)] \quad \text{(EC3)}$$

and fluents which have been terminated by an occurrence of an action continue not to hold until an occurrence of an action which initiates them:

$$\neg \mathit{HoldsAt}(f, t_2) \leftarrow [\mathit{Happens}(a, t_1) \wedge \mathit{Terminates}(a, f, t_1) \wedge t_1 < t_2 \wedge \neg \mathit{Declipped}(t_1, f, t_2)] \quad \text{(EC4)}$$

The four axioms above capture the behaviour of fluents once initiated or terminated by an action. But we need also to describe fluents' behaviour before the occurrence of any actions which affect them. We therefore axiomatise a general principle of persistence for fluents; fluents change their truth values only via the occurrence of initiating and terminating actions:

$$\mathit{HoldsAt}(f, t_2) \leftarrow [\mathit{HoldsAt}(f, t_1) \wedge t_1 < t_2 \wedge \neg \mathit{Clipped}(t_1, f, t_2)] \quad \text{(EC5)}$$

[1] By $E_1 \stackrel{\text{def}}{\equiv} E_2$ we mean that expression E_1 is notational shorthand for expression E_2.

$$\neg HoldsAt(f,t_2) \leftarrow [\neg HoldsAt(f,t_1) \wedge t_1 < t_2 \wedge \neg Declipped(t_1,f,t_2)] \quad \text{(EC6)}$$

Definitions of the predicates *Happens*, *Initiates* and *Terminates* are given in the domain-dependent part of the theory, as illustrated in the following example.

2.1 An Example Domain Dependent Axiomatisation

As an example domain dependent theory, we axiomatise a simple scenario of a robot going outside a room by moving through a door, which can be locked and unlocked using an electronic key. For this example we will assume a real number time-line. We will use three fluents, *Inside* (the robot is inside the room), *HasKey* (the robot is holding the electronic key), and *Locked* (the door is locked), and three actions, *Insert* (insert the key in the door), *GoThrough* (move through the door), and *Pickup* (pick up the key). We assume uniqueness-of-names axioms[2] which confirm that all of these constant symbols refer to distinct fluents or actions. Inserting the key alternately locks and unlocks the door. Picking up the key causes the robot to be holding the key, and, going through the unlocked door causes the robot to swap from being inside to outside or vice-versa. We use the predicates *Initiates* and *Terminates* to express these effects:

$$\begin{aligned} Initiates(a,f,t) \equiv\ & [[a = Pickup \wedge f = HasKey] \\ & \vee [a = Insert \wedge f = Locked \\ & \quad \wedge \neg HoldsAt(Locked,t) \\ & \quad \wedge HoldsAt(HasKey,t)] \\ & \vee [a = GoThrough \wedge f = Inside \\ & \quad \wedge \neg HoldsAt(Locked,t) \\ & \quad \wedge \neg HoldsAt(Inside,t)]] \end{aligned} \quad \text{(R1)}$$

$$\begin{aligned} Terminates(a,f,t) \equiv\ & [[a = GoThrough \wedge f = Inside \\ & \quad \wedge \neg HoldsAt(Locked,t) \\ & \quad \wedge HoldsAt(Inside,t)] \\ & \vee [a = Insert \wedge f = Locked \\ & \quad \wedge HoldsAt(Locked,t) \\ & \quad \wedge HoldsAt(HasKey,t)]] \end{aligned} \quad \text{(R2)}$$

[2] In this case the collection of uniqueness-of-names axioms will consist of a sentence such as $Inside \neq HasKey$ for each pair of fluent names and action names. In domains where parameterised fluents or actions are used, e.g. a $Lower(x)$ action to represent the act of lowering an object x meters, it might also typically include sentences such as $Lower(x_1) = Lower(x_2) \rightarrow x_1 = x_2$. The inclusion of such uniqueness-of-names axioms is not obligatory (we might for example wish to deliberately use two names to refer to the same action), but their omission will generally lead to unexpected results.

Let us suppose that the door is locked and the robot is inside at time 0, and that the robot picks up the key, unlocks the door and goes through the door at times 2, 4 and 6 respectively:

$$HoldsAt(Locked, 0) \ \wedge \ HoldsAt(Inside, 0) \tag{R3}$$

$$\begin{aligned} Happens(a, t) \ \equiv \ & [[a = Pickup \ \wedge \ t = 2] \vee \\ & [a = Insert \ \wedge \ t = 4] \vee \\ & [a = GoThrough \ \wedge \ t = 6]] \end{aligned} \tag{R4}$$

The reader is invited to check that from (EC1)-(EC6) and (R1)-(R4), together with uniqueness-of-names axioms for fluents and actions and an appropriate axiomatisation of the real numbers, it is for example possible to deduce that the robot is no longer inside the room at time 8, i.e. $\neg HoldsAt(Inside, 8)$.

2.2 Circumscription and the Frame Problem

In Event Calculus terms, the frame problem is the problem of expressing in a succinct and elaboration tolerant way that in most cases a given action will not initiate or terminate a given fluent. The description of which actions initiate and terminate which fluents via single biconditionals (as in axioms (R1) and (R2) above), although succinct, is rather unsatisfactory from the point of view of elaboration tolerance. For example, if new information about the initiating effects of a new action needs to be included in the robot domain (e.g. $Initiates(PressDoorBell, RingingNoise, t)$) this cannot be simply added to the theory, since it would be inconsistent with axiom (R1) (from which it is possible to infer $\neg Initiates(PressDoorBell, RingingNoise, t)$).

Hence most versions of the Event Calculus describe each fact or rule about initiation and termination in a separate axiom or clause, and provide an extra transformation or non-monotonic reasoning method to infer negative information about *Initiates* and *Terminates* from the collection of such rules. In the context of our classical logic Event Calculus and robot example, the individual rules would be

$$Initiates(Pickup, HasKey, t) \tag{R5}$$

$$\begin{aligned} Initiates(Insert, Locked, t) \ \leftarrow \\ [\neg HoldsAt(Locked, t) \wedge HoldsAt(HasKey, t)] \end{aligned} \tag{R6}$$

$$\begin{aligned} Initiates(GoThrough, Inside, t) \ \leftarrow \\ [\neg HoldsAt(Locked, t) \wedge \neg HoldsAt(Inside, t)] \end{aligned} \tag{R7}$$

$$\begin{aligned} Terminates(GoThrough, Inside, t) \ \leftarrow \\ [\neg HoldsAt(Locked, t) \wedge HoldsAt(Inside, t)] \end{aligned} \tag{R8}$$

$$Terminates(Insert, Locked, t) \leftarrow [HoldsAt(Locked, t) \wedge HoldsAt(HasKey, t)] \quad \text{(R9)}$$

Predicate completion or circumscription [39] can then be used to transform this collection of axioms into expressions such as (R1) and (R2). In this article we use the notation described in [37] to indicate circumscriptions of particular conjunctions of sentences. In particular, the circumscription

$$CIRC[(\text{R5}) \wedge (\text{R6}) \wedge (\text{R7}) \wedge (\text{R8}) \wedge (\text{R9}) \ ; \ Initiates, Terminates]$$

yields exactly (R1) and (R2). For simple domains such as the above, this type of transformation (whether described in terms of circumscription or predicate completion) is analogous to the solution to the frame problem developed by Reiter for the Situation Calculus [50].

To make useful deductions using axioms (EC1)-(EC6), it is also necessary to be able to infer both positive and negative information about *Happens* from the domain dependent part of the theory. Again the issue of elaboration tolerance arises, so that, as for *Initiates* and *Terminates*, most versions of the Event Calculus encapsulate each individual action occurrence in a separate *Happens* assertion (rather than using a biconditional such as (R4)), and then use some form of non-monotonic reasoning to infer negative information about this predicate. For example, in the case of our robot example the assertions would be

$$Happens(Pickup, 2) \quad \text{(R10)}$$

$$Happens(Insert, 4) \quad \text{(R11)}$$

$$Happens(GoThrough, 6) \quad \text{(R12)}$$

The circumscription $CIRC[(\text{R10}) \wedge (\text{R11}) \wedge (\text{R12}) \ ; \ Happens]$ then gives (R4).

If we now wish to add more information about *Happens*, we can do so without altering axioms (R10)-(R12) and then reapply the circumscription operator. This information need not just be in the form of ground literals – we may have less precise information about the order or timing of action occurrences. For example, we might know that the robot pressed the door bell either just before, just after or at the same time as inserting the key, in which case we could add

$$\exists t_1.[Happens(PressDoorBell, t_1) \ \wedge \ 2 < t_1 < 6] \quad \text{(R13)}$$

The circumscription $CIRC[(\text{R10}) \wedge (\text{R11}) \wedge (\text{R12}) \wedge (\text{R13}) \ ; \ Happens]$ then gives

$$\exists t_1.[2 < t_1 < 6 \ \wedge \ [Happens(a,t) \ \equiv \ [[a = Pickup \ \wedge \ t = 2] \vee \ [a = Insert \ \wedge \ t = 4] \vee \ [a = GoThrough \ \wedge \ t = 6] \vee \ [a = PressDoorBell \ \wedge \ t = t_1]]]] \quad \text{(R14)}$$

enabling us to deduce facts such as $\neg HoldsAt(Inside, 8)$ as before.

More generally, complete Event Calculus domain descriptions of this basic type are of the form

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

where Σ is a conjunction of *Initiates* and *Terminates* formulae, Δ is a conjunction of *Happens* and temporal ordering formulae, Ω is a conjunction of fluent-specific *HoldsAt* formulae such as (R3) and time-independent formulae (such as uniqueness-of-names axioms for actions and fluents), and EC is the conjunction of axioms (EC1) to (EC6) together an appropriate axiomatisation of the sort $\mathcal{T}$. 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 thus the splitting of the theory into different parts, which are circumscribed separately. This technique, sometimes referred to as *forced separation*, is also employed in [9], [14] and [28], and is akin to what Sandewall calls *filtering* [56].

2.3 Narrative Information and Planning

In some circumstances it is convenient to define *Happens* in terms of other predicates representing different categories of action occurrence. For example, in the context of planning we may wish to distinguish between actions that have (definitely) happened in the past and actions that the agent will (possibly) perform in the future[3]. In this case we may include a domain independent axiom such as

$$Happens(a,t) \ \equiv \ [Occurred(a,t) \ \vee \ Perform(a,t)] \quad \text{(EC7)}$$

We can now maintain a complete definition for *Occurred* (in the same way that we previously had a complete definition for *Happens*) based on our knowledge of actions that have already taken place, whilst keeping *Perform* undefined within the theory. In this way we can formulate a deductive specification of the planning task in terms of *Perform*. For example, in the context of the robot example suppose that at time 3 we know that a *Pickup* action has already taken place (at time 2), and wish to plan for the goal $\neg HoldsAt(Inside, 8)$. We would

[3] Or we may wish to distinguish between actions performed by the agent and events occurring in the environment and outside the agent's control.

include the axiom

$$Occurred(a,t) \equiv [a = Pickup \wedge t = 2] \tag{R15}$$

(or the equivalent expression $CIRC[Occurred(Pickup, 2) ; Occurred]$) in the domain description, and then show that the sentence

$$Perform(a,t) \equiv [[a = Insert \wedge t = 4] \vee [a = GoThrough \wedge t = 6]] \tag{P1}$$

is a plan for $\neg HoldsAt(Inside, 8)$ in the sense that

$$[(\text{EC1}) \wedge \ldots \wedge (\text{EC7}) \wedge (\text{R1}) \wedge (\text{R2}) \wedge (\text{R3}) \wedge (\text{R15})] \models [(\text{P1}) \rightarrow \neg HoldsAt(Inside, 8)]$$

More generally, planning can be viewed as the deduction[4] of sentences of the form $[Plan \rightarrow Goal]$ from an Event Calculus domain description, where *Plan* is a sentence such as (P1) defining the predicate *Perform*, and *Goal* is a sentence containing just the predicates *HoldsAt* and $<$ (we need also to establish via general theorems or a specific check that *Plan* is consistent with the Event Calculus theory). By the Deduction Theorem (see e.g. [18]) $Theory \models [Plan \rightarrow Goal]$ is equivalent to $[Theory \wedge Plan] \models Goal$ so that planning in the context of the Event Calculus can also be understood in terms of abduction (i.e. finding plans to add to the theory so that the goal is entailed). Indeed, it is this abductive view which is taken in the majority of work on Event Calculus planning, e.g. in [15], [8], [19], [44], [45], [63], [65] and [20].

2.4 Non-determinism

In contrast to many versions of the Event Calculus, the axiomatisation described in (EC1)–(EC6) is non-deterministic, in the sense that simultaneously initiating and terminating a fluent simply gives rise to two sets of models (one in which the fluent is true immediately afterwards and one in which it is false), rather than resulting in an inconsistent theory. This is because of the requirement in axioms (EC1) and (EC2) that $t_1 \leq t$, rather than $t_1 < t$.

For example, let us suppose that the action of tossing a coin is represented as *TossCoin*, and that each occurrence of this action results in the fluent *HeadsUp* being either true or false. We can represent this with an *Initiates* and a *Terminates* literal:

$$Initiates(TossCoin, HeadsUp, t) \tag{C1}$$

$$Terminates(TossCoin, HeadsUp, t) \tag{C2}$$

[4] That is to say, planning can be specified as a deductive task. We do not wish to claim that general purpose classical theorem provers are practical as planning systems.

Suppose the time is represented as the reals, and that a single *TossCoin* action happens at time 2:

$$Happens(TossCoin, 2) \tag{C3}$$

The theory which consists of axioms (EC1)-(EC6), $CIRC[(\text{C1}) \wedge (\text{C2})\ ;\ Initiates, Terminates]$ and $CIRC[(\text{C3})\ ;\ Happens]$ has four classes of models with respect to the fluent *HeadsUp* – one in which *HeadsUp* holds for all timepoints, one in which *HeadsUp* holds for no timepoints, one in which *HeadsUp* changes from true to false for all timepoints greater than 2, and one in which *HeadsUp* changes from false to true for all timepoints greater than 2. This is because using axioms (EC1) and (EC2) we can show $Clipped(2, HeadsUp, T)$ and $Declipped(2, HeadsUp, T)$ for all $T > 2$, so that (EC3) and (EC4) are trivially satisfied and the truth value of *HeadsUp* at different timepoints is constrained only by axioms (EC5) and (EC6).

The narrative-based nature of the Event Calculus (i.e. the fact that action occurrences are explicitly represented) facilitates a simple alternative to representing non-determinism. We can for example regard the action *TossCoin* as representing a choice of two deterministic actions *TossHead* and *TossTail*:

$$Happens(TossCoin, t) \rightarrow [Happens(TossHead, t) \vee Happens(TossTail, t)] \tag{C4}$$

We can then rewrite axioms (C1) and (C2) as

$$Initiates(TossHead, HeadsUp, t) \tag{C1a}$$

$$Terminates(TossTail, HeadsUp, t) \tag{C2a}$$

The theory consisting of (EC1)-(EC6), (C5), $CIRC[(\text{C3}) \wedge (\text{C4})\ ;\ Happens]$ and $CIRC[(\text{C1a}) \wedge (\text{C2a})\ ;\ Initiates, Terminates]$ now also gives rise to the desired classes of models described above. (The circumscription of *Happens* eliminates models where both a *TossHead* and a *TossTail* action occur at time 2.) For tasks such as planning, it is straightforward to specify that the agent in question can attempt some actions (such as *TossCoin*) but not others (such as *TossHead* or *TossTail*).

2.5 Concurrent Actions

The syntax of the Event Calculus makes it straightforward to express that two or more actions have occurred or will occur simultaneously, since different *Happens* literals in the domain description may refer to the same timepoint.

In some domains, concurrently performed actions may cancel each others' effects, and may combine to cause effects which none of the actions performed in isolation would achieve. A standard example is that if a bowl is filled with water, lifting just the left side of the bowl or just the right side will cause the

water to spill. Lifting both sides simultaneously will not cause the water to spill but will cause the bowl to be raised.

In the Event Calculus, we can describe cancellations and combinations of effects with *Happens* preconditions in the domain dependent axioms defining *Initiates* and *Terminates*. For example:

$$Initiates(LiftLeft, Spilt, t) \leftarrow \neg Happens(LiftRight, t) \quad \text{(B1)}$$

$$Initiates(LiftRight, Spilt, t) \leftarrow \neg Happens(LiftLeft, t) \quad \text{(B2)}$$

$$Initiates(LiftRight, Raised, t) \leftarrow Happens(LiftLeft, t) \quad \text{(B3)}$$

To illustrate the effect of such statements, suppose that our domain description also includes the following narrative information:

$$\neg HoldsAt(Spilt, 0) \quad \text{(B4)}$$

$$Happens(LiftLeft, 2) \quad \text{(B5)}$$

$$Happens(LiftRight, 2) \quad \text{(B6)}$$

The theory consisting of (EC1)-(EC6), (B4), $CIRC[(\text{B1}) \wedge (\text{B2}) \wedge (\text{B3}) \ ; \ Initiates, Terminates]$ and $CIRC[(\text{B5}) \wedge (\text{B6}) \ ; \ Happens]$ entails, for example, both $\neg HoldsAt(Spilt, 4)$ and $HoldsAt(Raised, 4)$.

3 Alternative and Extended Classical Logic Event Calculus Axiomatisations

The version of the Event Calculus described in Section 2 has a number of characteristics; it is geared to time-lines extending infinitely backwards as well as forwards, it is "non-deterministic" (in the sense described in Section 2.4), it regards all actions as possible under all circumstances, it regards all fluents' truth values as persisting between all relevant action occurrences, and it regards all action occurrences as instantaneous. However, the choice of which of these characteristics to include in a given Event Calculus axiomatisation is to a large extent arbitrary, and in this section we describe alternative axiomatisations which each negate one or more of these properties.

For ease of presentation, sub-sections 3.1 to 3.7 below each alter the axiomatisation (EC1)–(EC6) as little as possible to illustrate the particular point under discussion. But unless otherwise stated these alterations can be combined in a straightforward and obvious manner. For example we can combine the modifications described in sub-sections 3.2, 3.3 and 3.6 below to produce a "deterministic" Event Calculus with facilities to describe when it is impossible for particular actions to occur, and including actions of a non-zero duration.

Where in a particular sub-section no alternative to one of the axioms (EC1)–(EC6) is given, it should be assumed that the axiom in question remains unchanged.

3.1 An Alternative Axiomatisation for Non-Negative Time

Where time is modeled as the non-negative reals or integers, it is often convenient to introduce two new predicates[5] $\mathit{InitiallyP} \subseteq \mathcal{F}$ and $\mathit{InitiallyN} \subseteq \mathcal{F}$ ("P" for "positive" and "N" for "negative"), and to replace axioms (EC5) and (EC6) with the following three axioms:

$$\mathit{HoldsAt}(f,t) \leftarrow [\mathit{InitiallyP}(f) \wedge \neg \mathit{Clipped}(0,f,t)] \tag{EC5a}$$

$$\neg\mathit{HoldsAt}(f,t) \leftarrow [\mathit{InitiallyN}(f) \wedge \neg \mathit{Declipped}(0,f,t)] \tag{EC6a}$$

$$\mathit{InitiallyP}(f) \vee \mathit{InitiallyN}(f) \tag{EC8a}$$

Indeed, for non-negative time (EC5a), (EC6a) and (EC8a) may be deduced from (EC5) and (EC6) together with the assertion

$$[\mathit{HoldsAt}(f,0) \equiv \mathit{InitiallyP}(f)] \wedge [\neg\mathit{HoldsAt}(f,0) \equiv \mathit{InitiallyN}(f)]$$

Axioms (EC5a) and (EC6a) have an advantage over (EC5) and (EC6) in that they can readily be converted to logic program clauses without causing obvious looping problems.

However, this alternative axiomatisation is slightly weaker. For example, in the non-deterministic domain described in Sub-section 2.4 by axioms (C1), (C2) and (C3), axioms (EC5a), (EC6a) and (EC8a) would license models where *HeadsUp* fluctuated arbitrarily between true and false at times after 2. Although this characteristic is problematic for this particular example, it can be an advantage for representing other types of domain where it is convenient to "dynamically manage the frame", i.e. to regard some fluents as having an inherent persistence during some intervals of time but not during others. Indeed, for such domains axiom (EC8a) may not be appropriate. These issues are discussed in more detail in Section 3.7.

Since this particular axiomatisation does not include the general principle of persistence encapsulated in (EC5) and (EC6) (which describes how fluents persist independently of initiating and terminating action occurrences), adding individual *HoldsAt* literals to a given domain description (see for example axiom (R3) in Section 2.1) no longer necessarily has the same effect, particularly in axiomatisations where (EC8a) is omitted. Instead, individual observations of the form $\mathit{HoldsAt}(F,T)$ and $\neg\mathit{HoldsAt}(F,T)$ can be assimilated indirectly into the theory (perhaps automatically by a process of abduction) by appropriate addition of *Happens*, *InitiallyP* and *InitiallyN* literals. Axiom (R3), for example,

[5] These predicates are referred to as *InitiallyTrue* and *InitiallyFalse* in [42].

can be replaced by

$$\mathit{InitiallyP}(\mathit{Locked}) \ \wedge \ \mathit{InitiallyP}(\mathit{Inside}) \tag{R3a}$$

so that (R3) is now entailed by the theory consisting of (R3a), (R1), (R2), (R4), (EC1)–(EC4), (EC5a) and (EC6a).

3.2 Deterministic Event Calculus

A strictly deterministic Event Calculus (in the sense that simultaneously initiating and terminating a fluent results in inconsistency) may be formulated by replacing (EC3) and (EC4) by the following two axioms:

$$\begin{aligned}\mathit{HoldsAt}(f,t_2) \ \leftarrow \ [&\mathit{Happens}(a,t_1) \ \wedge \ \mathit{Initiates}(a,f,t_1) \\ &\wedge \ t_1<t_2 \ \wedge \ \neg\mathit{StoppedIn}(t_1,f,t_2)]\end{aligned} \tag{EC3b}$$

$$\begin{aligned}\neg\mathit{HoldsAt}(f,t_2) \ \leftarrow \ [&\mathit{Happens}(a,t_1) \ \wedge \ \mathit{Terminates}(a,f,t_1) \\ &\wedge \ t_1<t_2 \ \wedge \ \neg\mathit{StartedIn}(t_1,f,t_2)]\end{aligned} \tag{EC4b}$$

where the predicates *StoppedIn* and *StartedIn* are defined as follows:

$$\begin{aligned}\mathit{StoppedIn}(t_1,f,t_2) \ \stackrel{\text{def}}{\equiv} \ \exists a,t[&\mathit{Happens}(a,t) \ \wedge \ t_1<t<t_2 \\ &\wedge \ \mathit{Terminates}(a,f,t)]\end{aligned} \tag{EC9b}$$

$$\begin{aligned}\mathit{StartedIn}(t_1,f,t_2) \ \stackrel{\text{def}}{\equiv} \ \exists a,t[&\mathit{Happens}(a,t) \ \wedge \ t_1<t<t_2 \\ &\wedge \ \mathit{Initiates}(a,f,t)]\end{aligned} \tag{EC10b}$$

Note that *StoppedIn* and *StartedIn* are identical to *Clipped* and *Declipped* except for the inequality relations between the time-point variables. Strictly speaking (EC1) and (EC2) defining *Clipped* and *Declipped* are still required since these predicates are used in (EC5) and (EC6), or in their substitutes (EC5a) and (EC6a). But for domains using non-negative time and axioms (EC5a) and (EC6a), the definitions of *Clipped* and *Declipped* may be straightforwardly replaced by those for *StoppedIn* and *StartedIn*, provided no actions occur at time 0 which either terminate initially-positive fluents or initiate initially-negative fluents.

The effective meanings of *Initiates* and *Terminates* are slightly different in the deterministic Event Calculus (i.e. in axiomatisations including (EC3b) and (EC4b)) from their meanings in non-deterministic Event Calculus. (EC3b) and (EC4b) ensure that $\mathit{Initiates}(A,F,T)$ can be read as "F holds immediately after an occurrence of A at time T", whereas with axioms (EC3) and (EC4) $\mathit{Initiates}(A,F,T)$ corresponds to the slightly weaker assertion that "an occurrence of A at time T has an initiating influence on F" (which may or may not be overridden by a simultaneously occurring terminating influence).

There are several ways in which non-determinism may be reintroduced into what we have described here as deterministic Event Calculus. For example, the

technique exemplified in axioms (C1a), (C2a) and (C4) (see Section 2.4) is still applicable. Other methods include the use of *determining fluents* or a *Releases* predicate (see [62] and Section 3.7).

3.3 Action Preconditions and the Qualification Problem in the Event Calculus

We have already illustrated with axioms such as (R6)-(R9) (See Section 2.2) how preconditions for particular effects of actions may be expressed within the Event Calculus. These types of precondition are often referred to as *fluent preconditions*. There are also various ways in which *action preconditions* (i.e. conditions necessary for actions to be possible at all) can be expressed. One method is to introduce a new predicate $Impossible \subseteq \mathcal{A} \times \mathcal{T}$ and write an appropriate definition for *Impossible* with respect to each action in the domain in question. For instance, in our example domain we may wish to express that it is impossible for the robot to pickup the key if it is not fitted with a grabber, and it is impossible for the robot to go through a locked door:

$$Impossible(Pickup, t) \leftarrow \neg HoldsAt(HasGrabber, t) \tag{R16}$$

$$Impossible(GoThrough, t) \leftarrow HoldsAt(Locked, t) \tag{R17}$$

We can regard the qualification problem (at least in part) as the problem of expressing, in a succinct and elaboration tolerant way, that under most circumstances most actions are possible. To achieve this in the Event Calculus, we can minimise the predicate *Impossible*. $CIRC[(\text{R16}) \wedge (\text{R17}) \ ; \ Impossible]$ gives

$$\begin{aligned} Impossible(a, t) \equiv \ & [[a = Pickup \wedge \neg HoldsAt(HasGrabber, t)] \\ & \vee [a = GoThrough \wedge HoldsAt(Locked, t)]] \end{aligned} \tag{R18}$$

For narrative formalisms such as the Event Calculus, the way in which this type of knowledge is to be interpreted, and thus the way in which the domain independent axioms need to be adapted, depends to some extent on the individual domain and mode of reasoning under consideration. For tasks such as planning, which involves (hypothetical) reasoning about future events, it makes sense to regard the assertion $Impossible(A, T)$ as stating "it is impossible to predict the effects of attempting to perform action A at time T" (so that $\neg Impossible(A, T)$ can be regarded as analogous to $Poss(A, S)$ in Reiter's Situation Calculus [50]). In this case it is necessary only to block any inferences about what holds or does not hold at any time after an (attempt at an) 'impossible' action occurrence. This can be done by appropriately modifying the definitions of *Clipped* and *Declipped*:

$$Clipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \quad \text{(EC1c)}$$
$$[\exists a, t[Happens(a, t) \wedge t_1 \leq t < t_2 \wedge Terminates(a, f, t)]$$
$$\vee \exists a, t[Happens(a, t) \wedge t < t_2 \wedge Impossible(a, t)]]$$

$$Declipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \quad \text{(EC2c)}$$
$$[\exists a, t[Happens(a, t) \wedge t_1 \leq t < t_2 \wedge Initiates(a, f, t)]$$
$$\vee \exists a, t[Happens(a, t) \wedge t < t_2 \wedge Impossible(a, t)]]$$

On the other hand, if for example the domain includes certain knowledge about actions or events that have actually occurred in the past, it makes sense to regard the assertion $Impossible(A, T)$ as stating "action A could not have occurred at time T". Hence where the definition of *Happens* is split as in axiom (EC7) (see Section 2.3), we can include additional constraints such as

$$\neg Occurred(a, t) \leftarrow Impossible(a, t) \quad \text{(EC11c)}$$

Notice for example that from (R15), (R16) and (EC11c) we can infer the action precondition $HoldsAt(HasGrabber, 2)$ for the known occurrence of *Pickup* at time 2. This illustrates why we would not want to include constraints analogous to (EC11c) for hypothetical future performances of actions – at time 0 we would not for example want $Perform(Pickup, 1)$ to constitute a plan for the goal $HoldsAt(HasGrabber, 1)$. We could however safely state that nothing happens when an agent attempts to perform an impossible action, by replacing (EC7) with

$$Happens(a, t) \equiv [[Perform(a, t) \wedge \neg Impossible(a, t)] \quad \text{(EC7c)}$$
$$\vee Occurred(a, t)]$$

Finally, note that rules such as (R16) and (R17) partially defining *Impossible* can have *Happens* (and *Perform* and *Occurred*) preconditions as well as *HoldsAt* preconditions. This can be useful, for example, for expressing that it is impossible to perform certain combinations of actions simultaneously. For instance, the sentence

$$Impossible(a_1, t) \leftarrow [Perform(a_2, t) \wedge a_1 \neq a_2]$$

states that it is in general impossible to perform more than one action at a time. Like (R16) and (R17), such sentences must be placed within the scope of the circumscription of *Impossible*.

3.4 Categorisation of Fluents in the Event Calculus

For some domains, it is appropriate to categorise fluents into *frame* fluents and *non-frame* fluents (or *primitive* and *derived* fluents), and then to restrict the application of the principles of persistence encapsulated in axioms (EC3)-(EC6) to frame fluents only. To do this it is necessary to introduce a new predicate

$Frame \subseteq \mathcal{F}$, and alter (EC3)-(EC6) as follows:

$$HoldsAt(f,t_2) \leftarrow [Happens(a,t_1) \wedge Initiates(a,f,t_1) \wedge Frame(f) \wedge t_1<t_2 \wedge \neg Clipped(t_1,f,t_2)] \quad \text{(EC3d)}$$

$$\neg HoldsAt(f,t_2) \leftarrow [Happens(a,t_1) \wedge Terminates(a,f,t_1) \wedge Frame(f) \wedge t_1<t_2 \wedge \neg Declipped(t_1,f,t_2)] \quad \text{(EC4d)}$$

$$HoldsAt(f,t_2) \leftarrow [HoldsAt(f,t_1) \wedge t_1<t_2 \wedge Frame(f) \wedge \neg Clipped(t_1,f,t_2)] \quad \text{(EC5d)}$$

$$\neg HoldsAt(f,t_2) \leftarrow [\neg HoldsAt(f,t_1) \wedge t_1<t_2 \wedge Frame(f) \wedge \neg Declipped(t_1,f,t_2)] \quad \text{(EC6d)}$$

This axiom set can be useful when we want to include simple types of indirect effects in domain descriptions, since we are now free to write definitions or partial definitions of non-frame fluents (i.e. *state constraints*) in terms of *HoldsAt* and frame fluents. For example, as regards the robot we may wish to introduce a non-frame fluent *Happy* and state that, although the robot is not happy at time 0, it is in general happy if it is holding the key:

$$Frame(f) \equiv [f=Inside \vee f=HasKey \vee f=Locked] \quad \text{(R19)}$$

$$\neg HoldsAt(Happy,0) \quad \text{(R20)}$$

$$HoldsAt(Happy,t) \leftarrow HoldsAt(HasKey,t) \quad \text{(R21)}$$

Using (EC1), (EC2), (EC3d)-(EC6d), (R1)-(R4) and (R19)-(R21) we can now, for example, infer $HoldsAt(Happy,5)$. Indeed, we can also infer $\neg HoldsAt(HasKey,0)$ and therefore $\neg HoldsAt(HasKey,1)$. But we can neither infer $HoldsAt(Happy,1)$ nor $\neg HoldsAt(Happy,1)$, since the non-frame fluent *Happy* has no intrinsic persistence of its own.

3.5 Trajectories, Delayed Actions and Gradual Change

Several techniques are available within the context of the Event Calculus for describing delayed effects. The simplest approach is to write rules in terms of *Happens*. For example, if setting an alarm clock causes it to ring 8 hours later, we can write

$$Happens(StartRing,t+8) \leftarrow Happens(Set,t) \quad \text{(A1)}$$

$$Initiates(StartRing,Ringing,t) \quad \text{(A2)}$$

A disadvantage of rules such as (A1) is that it is difficult to express that the occurrence of the later action might be prevented by some intervening action (e.g. somebody might switch off the alarm during the night).

A more flexible approach involves the use of *trajectories* [58]. It is convenient to illustrate this technique here by introducing a new sort $\mathcal{P}$ of *parameters* into the language. Like fluents, parameters are time-varying properties, but unlike (frame) fluents they have no associated default persistence. More precisely, parameters are names for arbitrarily-valued functions of time, and accordingly we introduce a new function $ValueAt : \mathcal{P} \times \mathcal{T} \mapsto \mathcal{X}$. For example, we might write $ValueAt(Countdown, 5) = 2$ to indicate that at time 5 the parameter *Countdown*, representing the time remaining before the alarm clock rings, has a value of 2. To represent delayed and triggered effects, as well as simple forms of gradual or continuous change, specific parameters are associated with specific fluents via the predicate $Trajectory \subseteq \mathcal{F} \times \mathcal{T} \times \mathcal{P} \times \mathcal{T} \times \mathcal{X}$. The intended meaning of $Trajectory(F, T_1, P, T_2, X)$ is that if fluent F is initiated at time T_1 and continues to hold until time T_1+T_2, this results in parameter P having a value of X at time T_1+T_2. For example, in the case of the alarm clock we might write

$$Trajectory(SwitchedOn, t_1, Countdown, t_2, 8-t_2) \tag{A3}$$

We can translate this intended meaning into Event Calculus terms with the addition of a single extra domain independent axiom

$$\begin{aligned} ValueAt(p, t_1+t_2)=x \ \leftarrow \ & \\ & [Happens(a, t_1) \ \wedge \ Initiates(a, f, t_1) \\ & \wedge \ 0<t_2 \ \wedge \ Trajectory(f, t_1, p, t_2, x) \\ & \wedge \neg Clipped(t_1, f, t_1+t_2)] \end{aligned} \tag{EC11}$$

Continuing with our example, it is straightforward to express that when *Countdown* reaches 0 the alarm goes off:

$$Happens(StartRing, t) \ \leftarrow \ ValueAt(Countdown, t)=0 \tag{A4}$$

We can complete our description of the domain by stating that switching on the alarm activates the timing mechanism (provided it is not already activated), that the ringing event switches off the timing mechanism, that when the timing mechanism is switched off the countdown is permanently fixed at 8, that the alarm is initially not switched on and that someone switches it on at time 2:

$$Initiates(Set, SwitchedOn, t) \ \leftarrow \ \neg HoldsAt(SwitchedOn, t) \tag{A5}$$

$$Terminates(StartRing, SwitchedOn, t) \tag{A6}$$

$$ValueAt(Countdown, t)=8 \ \leftarrow \ \neg HoldsAt(SwitchedOn, t) \tag{A7}$$

$$\neg HoldsAt(SwitchedOn, 0) \tag{A8}$$

$$Happens(Set, 2) \tag{A9}$$

The theory consisting of (EC1)-(EC6), (EC11), (A3), (A7), (A8), $CIRC[(\text{A4}) \wedge (\text{A9})\ ;\ Happens]$ and $CIRC[(\text{A2}) \wedge (\text{A5}) \wedge (\text{A6})\ ;\ Initiates, Terminates]$ entails, for example, $Happens(StartRing, 10)$ and $HoldsAt(Ringing, 11)$.

The Event Calculus is symmetric as regards positive and negative *HoldsAt* literals and as regards *Initiates* and *Terminates*. Hence (EC11) has its counterpart in terms of *Terminates*:

$$\begin{aligned} ValueAt(p, t_1{+}t_2){=}x \ \leftarrow\ & \\ & [Happens(a, t_1)\ \wedge\ Terminates(a, f, t_1) \\ & \wedge\ 0{<}t_2\ \wedge\ AntiTrajectory(f, t_1, p, t_2, x) \\ & \wedge \neg Declipped(t_1, f, t_1{+}t_2)] \end{aligned} \tag{EC12}$$

This axiom uses the predicate $AntiTrajectory \subseteq \mathcal{F} \times \mathcal{T} \times \mathcal{P} \times \mathcal{T} \times \mathcal{X}$. The intended meaning of $AntiTrajectory(F, T_1, P, T_2, X)$ is that if fluent F is terminated at time T_1 and continues not to hold until time $T_1{+}T_2$, this results in parameter P having a value of X at time $T_1{+}T_2$. We can illustrate the use of anti-trajectories by representing the fact that a hot-air balloon rises when the air-heater is on, but falls when it is not:

$$\begin{aligned} Trajectory(HeaterOn, t_1, Height, t_2, x_1{+}t_2) & \\ \leftarrow ValueAt(Height, t_1){=}x_1 & \end{aligned} \tag{H1}$$

$$\begin{aligned} AntiTrajectory(HeaterOn, t_1, Height, t_2, x_1{-}t_2) & \\ \leftarrow ValueAt(Height, t_1){=}x_1 & \end{aligned} \tag{H2}$$

(Note that in the alarm clock example (A7) can also be expressed as $AntiTrajectory(SwitchedOn, t_1, Countdown, t_2, 8)$.)

Note that the functions captured in individual trajectories need not be continuous or even numerically valued. For example, we can use a trajectory to model the fact that the left indicator light of a car flashes once per second while the indicator switch is depressed:

$$Trajectory(IndicatorDepressed, t_1, Light, t_2, BlinkFunction(t_2)) \tag{L1}$$

$$AntiTrajectory(IndicatorDepressed, t_1, Light, t_2, Off) \tag{L2}$$

$$BlinkFunction(t){=}On\ \leftarrow\ [t\ mod\ 2 < 1] \tag{L3}$$

$$BlinkFunction(t){=}Off\ \leftarrow\ [t\ mod\ 2 \geq 1] \tag{L4}$$

In domains which include non-deterministic actions (in the sense that actions or combinations of actions can simultaneously initiate and terminate

fluents) axioms (EC11) and (EC12) are too weak. For example, if the switching on mechanism is faulty in our alarm clock example, so that we have both (A5) and

$$Terminates(Set, SwitchedOn, t) \tag{A10e}$$

axiom (EC11) will not inform us that the countdown is activated even in the circumstance where fluent *SwitchedOn* holds immediately after time 2. One solution is to replace (EC11) and (EC12) with equivalent axioms which have an extra *HoldsAt* condition in their right-hand sides, but use *StoppedIn* and *StartedIn* (see axioms (EC9b) and (EC10b)) instead of *Clipped* and *Declipped*:

$$\begin{aligned} &ValueAt(p, t_1+t_2)=x \ \leftarrow \\ &\quad [Happens(a, t_1) \ \wedge \ Initiates(a, f, t_1) \\ &\quad \wedge \ 0<t_2 \ \wedge \ Trajectory(f, t_1, p, t_2, x) \\ &\quad \wedge HoldsAt(f, t_1+t_2) \ \wedge \ \neg StoppedIn(t_1, f, t_1+t_2)] \end{aligned} \tag{EC11e}$$

$$\begin{aligned} &ValueAt(p, t_1+t_2)=x \ \leftarrow \\ &\quad [Happens(a, t_1) \ \wedge \ Terminates(a, f, t_1) \\ &\quad \wedge \ 0<t_2 \ \wedge \ AntiTrajectory(f, t_1, p, t_2, x) \\ &\quad \wedge \neg HoldsAt(f, t_1+t_2) \ \wedge \ \neg StartedIn(t_1, f, t_1+t_2)] \end{aligned} \tag{EC12e}$$

In Event Calculus axiomatisations where a distinction is made between fluents which are (temporarily or permanently) inside or outside the frame (such as in Section 3.4), we may dispense with the extra sort $\mathcal{P}$ in favour of non-frame fluents, and replace (EC11) and (EC12) with axioms such as

$$\begin{aligned} HoldsAt(f_2, t_1+t_2) \ \leftarrow \ &[\neg Frame(f_2) \ \wedge \ Happens(a, t_1) \\ &\wedge \ Initiates(a, f_1, t_1) \ \wedge \ 0<t_2 \\ &\wedge \ Trajectory(f_1, t_1, f_2, t_2) \\ &\wedge \ \neg Clipped(t_1, f_1, t_1+t_2)] \end{aligned} \tag{EC11f}$$

$$\begin{aligned} HoldsAt(f_2, t_1+t_2) \ \leftarrow \ &[\neg Frame(f_2) \ \wedge \ Happens(a, t_1) \\ &\wedge \ Terminates(a, f_1, t_1) \ \wedge \ 0<t_2 \\ &\wedge \ AntiTrajectory(f_1, t_1, f_2, t_2) \\ &\wedge \ \neg Declipped(t_1, f_1, t_1+t_2)] \end{aligned} \tag{EC12f}$$

Here $Trajectory \subseteq \mathcal{F} \times \mathcal{T} \times \mathcal{F} \times \mathcal{T}$, and the intended meaning of $Trajectory(F_1, T_1, F_2, T_2)$ is that if fluent F_1 is initiated at time T_1 and continues to hold until time T_1+T_2, this results in F_2 holding at time T_1+T_2 (similarly for *AntiTrajectory*). In the alarm clock example *Countdown* would then be parameterised, (A3), (A4) and (A6) would be written

$$Trajectory(SwitchedOn, t_1, Countdown(8-t_2), t_2) \tag{A3f}$$

$$Happens(StartRing, t) \leftarrow HoldsAt(Countdown(0), t) \tag{A4f}$$

$$HoldsAt(Countdown(8), t) \leftarrow \neg HoldsAt(SwitchedOn, t) \tag{A7f}$$

and the domain description would include the additional constraint

$$[HoldsAt(Countdown(x_1), t) \wedge HoldsAt(Countdown(x_2), t)] \rightarrow x_1 = x_2 \tag{A10f}$$

3.6 The Event Calculus and Actions with Duration

The Event Calculus can be modified in various ways so that actions can be represented as occurring over intervals of time. To illustrate, we present here a simple modification in which actions are assigned a numerical duration using the function $Dur : \mathcal{A} \mapsto \mathcal{T}$. This avoids the need to introduce extra arguments of sort $\mathcal{T}$ in the predicates *Happens*, *Initiates* and *Terminates*. For example, we will interpret the assertion $Happens(A, T)$ to mean "the action A starts to occur at T" (so that it finishes at $T + Dur(A)$).

We will be cautious in the assumptions we make about the effects of actions. We will assume that actions may affect relevant fluents from the moment they start, but the effects only become certain after the actions have finished. Hence the values of affected fluents should be undetermined by the axiomatisation during action occurrences. To incorporate these assumptions in the domain independent axioms (EC1)-(EC6) it is necessary only to modify the various inequality relations between the timepoint variables in (EC1)-(EC4):

$$Clipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[Happens(a, t) \wedge t_1 \leq (t + Dur(a)) \wedge t < t_2 \wedge Terminates(a, f, t)] \tag{EC1g}$$

$$Declipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[Happens(a, t) \wedge t_1 \leq (t + Dur(a)) \wedge t < t_2 \wedge Initiates(a, f, t)] \tag{EC2g}$$

$$HoldsAt(f, t_2) \leftarrow [Happens(a, t_1) \wedge Initiates(a, f, t_1) \wedge (t_1 + Dur(a)) < t_2 \wedge \neg Clipped(t_1, f, t_2)] \tag{EC3g}$$

$$\neg HoldsAt(f, t_2) \leftarrow [Happens(a, t_1) \wedge Terminates(a, f, t_1) \wedge (t_1 + Dur(a)) < t_2 \wedge \neg Declipped(t_1, f, t_2)] \tag{EC4g}$$

The issue of preconditions becomes more complex when actions have duration. We may for example wish to make a distinction between preconditions which must hold at the start of the action and those which must hold throughout the action. It is therefore often convenient to define auxiliary predicates such as

$HoldsIn \subseteq \mathcal{F} \times \mathcal{T} \times \mathcal{T}$:

$$HoldsIn(f, t_1, t_3) \stackrel{\text{def}}{\equiv} \forall t_2[t_1 \leq t_2 \leq t_3 \rightarrow HoldsAt(f, t_2)] \quad \text{(EC13)}$$

To illustrate the use of *HoldsIn*, consider a simple description of an automated train which can move at a fixed speed S along a track running from West to East, provided its motor is engaged. Using the action term $MoveEast(T)$ to represent the action of moving east for T time units, we can for example write axioms such as

$$Dur(MoveEast(t)) = t \quad \text{(T1)}$$

$$\begin{aligned} &Initiates(MoveEast(t), Location(x_2), t_1) \leftarrow \quad \text{(T2)} \\ &\quad [HoldsAt(Location(x_1), t_1) \\ &\quad \wedge\ x_2 = (x_1 + S \times Dur(MoveEast(t))) \\ &\quad \wedge\ HoldsIn(MotorEngaged, t_1, (t_1 + Dur(MoveEast(t))))] \end{aligned}$$

An alternative way of dealing with actions with duration is to split them into an (instantaneous) "start of action" (e.g. *StartMoveEast*), an "end of action" (e.g. *StopMoveEast*) and introduce an extra fluent representing the fact that the action is taking place (e.g. *MovingEast*). This approach is more easily integrated with the mechanisms described in Section 3.5 for dealing with gradual change, and allows straightforward description of interruptions of partly executed actions.

3.7 Dynamic Management of the Frame

We have already seen in Sections 3.4 and 3.5 how it can sometimes be advantageous to regard some fluents ("frame" fluents) as having an intrinsic (default) persistence, but regard other fluents as liable to change truth values between action occurrences. It can also be useful to be able to express that particular fluents have a default persistence during some intervals of time but not during others. This can, for example, help succinctly describe domains involving non-determinism, continuous change and indirect effects of actions (see [62] for details). In this section we illustrate how this facility for "dynamic management of the frame" can be incorporated into the Event Calculus by use of a new predicate $Releases \subseteq \mathcal{A} \times \mathcal{F} \times \mathcal{T}$. A form of this predicate was first introduced in [28] and it is related to Sandewall's idea of *occlusion* [56].

$Releases(A, F, T)$ expresses that if A occurs at T it will disable the fluent F's innate persistence. The truth value of F will then be free to fluctuate until the next action occurrence which initiates or terminates it. *Releases* is defined in the domain-dependent part of the theory and circumscribed in parallel with *Initiates* and *Terminates*. For example, in the alarm clock example of Section 3.5, we may write

$$Releases(Set, Countdown, t)$$

and if this is the only such statement in our theory, the circumscription will then give

$$Releases(a, f, t) \equiv [a = Set \wedge f = Countdown]$$

Initiates(A, F, T) (respectively *Terminates*(A, F, T)) now expresses that if A occurs at T it will both initiate (respectively terminate) the fluent F and enable F's innate persistence. At any given time-point, therefore, a fluent can be in one of four states – true and persisting, false and persisting, true and released or false and released. To describe these states explicitly, we introduce a predicate *ReleasedAt* $\subseteq \mathcal{F} \times \mathcal{T}$ analogous to *HoldsAt*. Finally we need two new auxiliary predicates *ReleasedBetween* $\subseteq \mathcal{T} \times \mathcal{F} \times \mathcal{T}$ and *PersistsBetween* $\subseteq \mathcal{T} \times \mathcal{F} \times \mathcal{T}$. *ReleasedBetween*$(T_1, F, T_2)$ means "an action releases the fluent F between times T_1 and T_2" and *PersistsBetween*(T_1, F, T_2) means "the fluent is not in a state of release at any time between T_1 and T_2."

The Event Calculus described in (EC1)–(EC6) needs fairly radical modifications to incorporate these extra concepts and predicates. The modified axiomatisation is as follows (for ease of reading (EC1) and (EC2) are listed again, although they are unmodified). The first three axioms are all similar and give definitions for *Clipped*, *Declipped* and *ReleasedBetween*:

$$Clipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[Happens(a, t) \wedge t_1 \leq t < t_2 \wedge Terminates(a, f, t)] \quad \text{(EC1)}$$

$$Declipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[Happens(a, t) \wedge t_1 \leq t < t_2 \wedge Initiates(a, f, t)] \quad \text{(EC2)}$$

$$ReleasedBetween(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[Happens(a, t) \wedge t_1 \leq t < t_2 \wedge Releases(a, f, t)] \quad \text{(EC14h)}$$

The next four axioms indicate how particular actions can put a fluent in one of the four states described above:

$$HoldsAt(f, t_2) \leftarrow [Happens(a, t_1) \wedge Initiates(a, f, t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1, f, t_2) \wedge \neg ReleasedBetween(t_1, f, t_2)] \quad \text{(EC3h)}$$

$$\neg HoldsAt(f, t_2) \leftarrow [Happens(a, t_1) \wedge Terminates(a, f, t_1) \wedge t_1 < t_2 \wedge \neg Declipped(t_1, f, t_2) \wedge \neg ReleasedBetween(t_1, f, t_2)] \quad \text{(EC4h)}$$

$$ReleasedAt(f, t_2) \leftarrow [Happens(a, t_1) \wedge Releases(a, f, t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1, f, t_2) \wedge \neg Declipped(t_1, f, t_2)] \quad \text{(EC15h)}$$

$$\neg ReleasedAt(f, t_2) \leftarrow [Happens(a, t_1) \wedge t_1 < t_2 \wedge [Initiates(a, f, t_1) \vee Terminates(a, f, t_1)] \wedge \neg ReleasedBetween(t_1, f, t_2)] \quad \text{(EC16h)}$$

A weakened version of the "commonsense law of inertia" is captured in the following three axioms:

$$PersistsBetween(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \neg\exists t[ReleasedAt(f, t) \wedge t_1 \leq t \leq t_2] \quad \text{(EC17h)}$$

$$HoldsAt(f, t_2) \leftarrow [HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge PersistsBetween(t_1, f, t_2) \wedge \neg Clipped(t_1, f, t_2)] \quad \text{(EC5h)}$$

$$\neg HoldsAt(f, t_2) \leftarrow [\neg HoldsAt(f, t_1) \wedge t_1 < t_2 \wedge PersistsBetween(t_1, f, t_2) \wedge \neg Declipped(t_1, f, t_2)] \quad \text{(EC6h)}$$

Finally, we need to state that the meta-property of being "released" is itself subject to a form of meta-persistence between action occurrences:

$$ReleasedAt(f, t_2) \leftarrow [ReleasedAt(f, t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1, f, t_2) \wedge \neg Declipped(t_1, f, t_2)] \quad \text{(EC18h)}$$

$$\neg ReleasedAt(f, t_2) \leftarrow [\neg ReleasedAt(f, t_1) \wedge t_1 < t_2 \wedge \neg ReleasedBetween(t_1, f, t_2)] \quad \text{(EC19h)}$$

Individual *ReleasedAt* literals can be included in the domain dependent part of the theory in the same way as *HoldsAt* literals (see for example axiom (R3) in Section 2.1).

The above axiomatisation is fairly complex – it replaces our original six axioms with twelve (longer) ones and introduces four new predicates. However, for practical and computational purposes (e.g. ease of translation into logic programs) and where we are using non-negative time, we can dispense with the predicates *ReleasedAt*, *ReleasedBetween* and *PersistsBetween* and simply incorporate *Releases* in the definitions of *Clipped* and *Declipped*. This gives rise to the following alternative (and complete) set of domain independent axioms:

$$Clipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[Happens(a, t) \wedge t_1 \leq t < t_2 \wedge [Terminates(a, f, t) \vee Releases(a, f, t)]] \quad \text{(EC1i)}$$

$$Declipped(t_1, f, t_2) \stackrel{\text{def}}{\equiv} \exists a, t[Happens(a, t) \wedge t_1 \leq t < t_2 \wedge [Initiates(a, f, t) \vee Releases(a, f, t)]] \quad \text{(EC2i)}$$

$$HoldsAt(f, t_2) \leftarrow [Happens(a, t_1) \wedge Initiates(a, f, t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1, f, t_2)] \quad \text{(EC3)}$$

$$\neg HoldsAt(f, t_2) \leftarrow [Happens(a, t_1) \wedge Terminates(a, f, t_1) \wedge t_1 < t_2 \wedge \neg Declipped(t_1, f, t_2)] \quad \text{(EC4)}$$

$$HoldsAt(f, t) \leftarrow [InitiallyP(f) \wedge \neg Clipped(0, f, t)] \quad \text{(EC5a)}$$

$$\neg HoldsAt(f, t) \leftarrow [InitiallyN(f) \wedge \neg Declipped(0, f, t)] \quad \text{(EC6a)}$$

Note that fluents which are *InitiallyP* or *InitiallyN* are initially in the frame, whereas those which are neither *InitiallyP* nor *InitiallyN* are effectively initially "released". Hence axiom (EC8a) (see Section 3.1) implies that all fluents are initially in the frame, and may or may not be appropriate for a given domain.

3.8 The Event Calculus, Continuous Change and Mathematical Modelling

The techniques using the *Trajectory* and *AntiTrajectory* predicates discussed in Section 3.5 are sufficient for modelling domains with very simple forms of continuous change, in particular where an explicit function of time is known for a particular parameter after a particular fluent has been initiated or terminated. However, this method is in general insufficient for integrating standard mathematical modelling techniques with the Event Calculus, for several reasons. First, the majority of mathematical models are expressed as sets of differential equations, and these cannot in general be solved so as to produce explicit functions of time for each parameter involved. Second, there might only be incomplete knowledge, expressed perhaps using inequalities, about the mathematical relationship between various parameters and/or their derivatives. Third, the circumstances under which various mathematical relationships hold between parameters might not be (easily) expressible in terms of a single fluent. Fourth, trajectories and antitrajectories do not provide mechanisms for describing continuous change in time intervals before any relevant initiating and/or terminating actions have occurred.

A more general approach is to include domain independent axioms which explicitly utilise the mathematical definitions of continuity and differentiability of real-valued functions of time. Under this approach, which is partly inspired by Sandewall's work [54, 55] and described in more detail in [42], continuity of real-valued parameters is regarded as a default analogous to default persistence of fluents, so that discontinuities arise only in particular parameters when specific actions occur. For this section, we will assume that time is represented either

as the real numbers or as the non-negative real numbers. We will assume that some or all terms of sort $\mathcal{P}$ (introduced in Section 3.5) represent real-valued functions of time, and accordingly introduce two new function symbols $Value : \mathcal{P} \times \mathcal{T} \mapsto \mathbb{R}$ and $\delta : \mathcal{P} \mapsto \mathcal{P}$. The term $Value(P, T)$ represents the numerical value of parameter P at time T, and the axiomatisation below ensures that the term $Value(\delta(P), T)$ represents the numerical value at time T of its first derivative (at all time-points where this exists).

To integrate the standard mathematical concepts of continuity and differentiability into the Event Calculus, we need to express them in terms of *Value* and δ. It is also convenient to introduce the predicates $LeftContinuous \subseteq \mathcal{P} \times \mathcal{T}$ and $RightLimit \subseteq \mathcal{P} \times \mathcal{T}$ to capture the corresponding (standard) mathematical concepts[6]:

$$Continuous(p,t) \equiv \forall r \exists t_1 \forall t_2 [[|t - t_2| < t_1 \wedge 0 < r] \rightarrow |Value(p,t) - Value(p,t_2)| < r] \quad \text{(EC20j)}$$

$$Differentiable(p,t) \equiv \forall r \exists t_1 \forall t_2 [[0 < |t - t_2| < t_1 \wedge 0 < r] \rightarrow |(\frac{Value(p,t) - Value(p,t_2)}{t - t_2}) - Value(\delta(p),t)| < r] \quad \text{(EC21j)}$$

$$LeftContinuous(p,t) \equiv \forall r \exists t_1 \forall t_2 [[t_2 < t \wedge (t - t_2) < t_1 \wedge 0 < r] \rightarrow |Value(p,t) - Value(p,t_2)| < r] \quad \text{(EC22j)}$$

$$RightLimit(p,t,r) \equiv \forall r_1 \exists t_1 \forall t_2 [[t < t_2 \wedge (t_2 - t) < t_1 \wedge 0 < r_1] \rightarrow |Value(p,t_2) - r| < r_1] \quad \text{(EC23j)}$$

To respect the convention that actions take effect immediately *after* they occur, it is necessary to axiomatise the mathematical constraint that, at every time-point (including those at which actions occur), the function associated with each parameter is left-hand continuous:

$$LeftContinuous(p,t) \quad \text{(EC24j)}$$

To describe instantaneous changes in the values of parameters at times when actions occur, and discontinuities in their corresponding functions of time, the predicates $BreaksTo \subseteq \mathcal{A} \times \mathcal{P} \times \mathcal{T} \times \mathbb{R}$ and $Breaks \subseteq \mathcal{A} \times \mathcal{P} \times \mathcal{T}$ are introduced.

[6] A function is *left-continuous* if discontinuities occur only between successive intervals where the first is closed on the right and the second is open on the left. For example the function $f(t) = 0$ for all $t \leq 1$, $f(t) = 2$ otherwise, is left-continuous at all time-points, whereas the function $f'(t) = 0$ for all $t < 1$, $f'(t) = 2$ otherwise, is not. The *right-limit* of a function at a particular point is the limit value as the point is approached from the right. So, for example, the right-limit of both f and f' at 1 is 2.

Both are minimised (by circumscribing them in parallel). $BreaksTo(A, P, T, R)$ should be read as 'at time T, an occurrence of action A will cause parameter P to instantaneously take on value R'. More precisely, Axiom (EC27j) below states that if A does indeed occur at time T, then R is the value of the right-hand limit of P at T. $Breaks(A, P, T)$ can be read as 'at time T, action A potentially causes a discontinuity in parameter P'. The following domain-independent axioms make direct use of *BreaksTo* and *Breaks*. Axioms (EC25j) and (EC26j) can be likened to 'frame axioms' for parameters. Axiom (EC28j) states the relationship between *BreaksTo* and *Breaks*, and Axiom (EC29j) states that if an action potentially causes a discontinuity in a given parameter, it also potentially causes discontinuities in its higher derivatives.

$$\neg[Happens(a, t) \wedge Breaks(a, p, t)] \rightarrow Continuous(p, t) \tag{EC25j}$$

$$\neg[Happens(a, t) \wedge Breaks(a, \delta(p), t)] \rightarrow Differentiable(p, t) \tag{EC26j}$$

$$[BreaksTo(a, p, t, r) \wedge Happens(a, t)] \rightarrow RightLimit(p, t, r) \tag{EC27j}$$

$$BreaksTo(a, p, t, r) \rightarrow Breaks(a, p, t) \tag{EC28j}$$

$$Breaks(a, p, t) \rightarrow Breaks(a, \delta(p), t) \tag{EC29j}$$

To make useful derivations using this axiomatisation, for any given time point T it is useful to be able to refer to the next point after T at which an action occurs, if there is such a point. Axioms (EC30j), (EC31j) and (EC32j) state that if any action occurs at any time point after T, then the term $Next(T)$ refers to the least such time point. (Such points are somewhat analogous to the "least natural time points" discussed in [51].)

$$t < Next(t) \tag{EC30j}$$

$$[t < t_1 \ \wedge \ t_1 < Next(t)] \ \rightarrow \ \neg Happens(a, t_1) \tag{EC31j}$$

$$[Happens(a_1, t_1) \ \wedge \ t < t_1] \ \rightarrow \ \exists a.Happens(a, Next(t)) \tag{EC32j}$$

The above axiomatisation leaves us free to include (unsolved) sets of simultaneous differential equations in domain descriptions. As a simple illustration, suppose we wish to represent that the rate of change of the level of liquid in a tank is negatively proportional to the flow through a valve in its bottom, and that when the valve is open the flow is in turn proportional to the level (i.e. pressure). We need a single fluent *ValveOpen*, two parameters *Level* and *Flow*, and actions *OpenValve* and *CloseValve*. As well as *Happens*, *Initiates* and *Terminates* facts such as

$$Initiates(OpenValve, ValveOpen, t) \tag{V1}$$

$$Terminates(CloseValve, ValveOpen, t) \tag{V2}$$

we can represent information about the instantaneous effects of actions on parameters using *Breaks*,

$$Breaks(OpenValve, Flow, t) \tag{V3}$$

$$Breaks(OpenValve, \delta(Level), t) \tag{V4}$$

and include mathematical constraints (differential equations) which hold in different circumstances, e.g.

$$Value(\delta(Level), t) = -Value(Flow, t) \tag{V5}$$

$$HoldsAt(ValveOpen, t) \rightarrow \exists r[Value(Flow, t) = r.Value(Level, t)] \tag{V6}$$

In this case the full theory will include the circumscription $CIRC[(\text{EC28j}) \wedge (\text{EC29j}) \wedge (\text{V3}) \wedge (\text{V4}) \ ; \ Breaks, BreaksTo]$. The Event Calculus now allows us to infer new boundary conditions for sets of differential equations which become applicable when actions such as *OpenValve* and *CloseValve* occur. A variation of this example is discussed in more detail in [42].

The above axiomatisation lays a foundation for integrating the Event Calculus with representational and computational techniques from the field of Qualitative Reasoning [9] [34]. An Event Calculus based axiomatisation of some of the basic concepts in [34] is given in [42].

3.9 Other Issues and Extensions

Space limitations forbid a detailed summary of all work done on extending the classical logic Event Calculus in this article. In particular, three important topics we have not covered are *hierarchical actions*, *ramifications*, and *knowledge producing actions*.

Hierarchical or *compound actions* are non-instantaneous actions whose occurrence consists of the occurrence of a set of shorter actions. (For example, the "go to work" action might comprise a "walk to the station" action, a "get the train" action and a "walk to the office" action.) These can be formalised in the Event Calculus using "happens if happens" formulae. For more details, see [63] or [68]. Davila [10] has done related work on formulating programming constructs within an Event Calculus framework.

The ramification problem is the problem of representing permanent constraints between collections of fluents, and indirect effects of actions propagated via such constraints, whilst preserving a succinct and elaboration tolerant solution to the frame problem. Shanahan [66] has shown that a straightforward extension of the Event Calculus can handle many canonical examples of the ramification problem, including those in which concurrent events simultaneously affect the same fluent. In Section 4 we show an equivalence between the Event

Calculus and the Language $\mathcal{E}$ [21], and $\mathcal{E}$ has been extended to deal with ramifications in [22] by using fixed point definitions to express how actions indirectly initiate and terminate fluents. It seems likely that this same technique can be described in the classical Event Calculus using inductive definitions similar to those in [69] and [70].

To our knowledge, little work has been done in the Event Calculus on representing the effects of knowledge producing actions. These are important, for example, in the context of planning. To catch a flight, an agent may plan to go to the airport and then look at the departures board to find out which gate the flight is boarding from. The action of looking at the board doesn't change the state of the external world but rather the agent's knowledge of it. To reason about such actions, the agent has to have a model about its own future knowledge state and how this will relate to the external world. Work on addressing these issues in the context of other action formalisms can be found for example in [36], [38], [46], [47] and [57].

4 A Correspondence Result

The focus of the previous sections has been on the development of Event Calculus axiomatisations written in standard predicate calculus to represent knowledge about the effects of actions. In this sense it follows the tradition established by McCarthy and others in developing the Situation Calculus [40]. Implicit in such work is the idea that such classical logic theories can act as specifications for computer programs that simulate various forms of reasoning about the domains represented. However, more recently there has been a trend towards the use of more specialised logics for representing and reasoning about the effects of actions, and in particular a growing body of work on the development and implementation of "action description languages" [16, 17]. It is not our intention here to argue the merits and demerits of specialised as opposed to general purpose logics. (We do not for example subscribe to the view that formulations in classical or other general purpose logics require formulations in specialised logics to act as their "specification" or "semantics", or that specialised logics are at a "higher level" because they lack a proof theory.) However, it is clearly advantageous to explore correspondences between various types of representation, so that results and implementations for one approach can be more readily adapted to others.

While the majority of action description languages bear a resemblance to the Situation Calculus, the Language $\mathcal{E}$ [21, 22] is inspired by, and inherits its ontology from, the Event Calculus. In this section we describe the circumstances under which Event Calculus theories correspond to Language $\mathcal{E}$ domain descriptions and may thus take advantage of the provably correct automated proof procedures that have been developed for $\mathcal{E}$ (see e.g. [23], [24], [26]).

4.1 The Language $\mathcal{E}$

The definition of the Language $\mathcal{E}$ given here corresponds to that in [21]. (This definition has subsequently been extended in various ways, in particular to deal with ramifications and the ramification problem [22, 23].)

The Language $\mathcal{E}$ is really a collection of languages. The particular vocabulary of each language depends on the domain being represented, but always includes a set of *fluent constants*, a set of *action constants*, and a partially ordered set of *time-points*. A *fluent literal* may either be a fluent constant or its negation, as shown in the following definitions.

Definition 1 (Domain Language). *A domain language is a tuple* $\langle \Pi, \preceq, \Delta, \Phi \rangle$*, where* $\preceq$ *is a partial (possibly total) ordering defined over the non-empty set* Π *of time points,* Δ *is a non-empty set of action constants, and* Φ *is a non-empty set of fluent constants.*

Definition 2 (Fluent literal). *A fluent literal of* $\mathcal{E}$ *is an expression either of the form* F *or of the form* $\neg F$*, where* $F \in \Phi$*.*

Three types of statements are used to describe domains; *h-propositions* ("h" for "happens"), *t-propositions* ("t" for "time point") and *c-propositions* ("c" for "causes"). Their intended meanings are clear from their definitions:

Definition 3 (h-proposition). *An h-proposition in* $\mathcal{E}$ *is an expression of the form*

$$A \textbf{ happens-at } T$$

where $A \in \Delta$ *and* $T \in \Pi$*.*

Definition 4 (t-proposition). *A t-proposition in* $\mathcal{E}$ *is an expression of the form*

$$L \textbf{ holds-at } T$$

where L *is a fluent literal of* $\mathcal{E}$ *and* $T \in \Pi$*.*

Definition 5 (c-proposition). *A c-proposition in* $\mathcal{E}$ *is an expression either of the form*

$$A \textbf{ initiates } F \textbf{ when } C$$

or of the form

$$A \textbf{ terminates } F \textbf{ when } C$$

where $F \in \Phi$*,* $A \in \Delta$*, and* C *is a set of fluent literals of* $\mathcal{E}$*.*

C-propositions of the form "A **initiates** F **when** $\emptyset$" and "A **terminates** F **when** $\emptyset$" can be written more simply as "A **initiates** F" and "A **terminates** F" respectively. A *domain description* in $\mathcal{E}$ is a triple $\langle \gamma, \eta, \tau \rangle$, where γ is a set of c-propositions, η is a set of h-propositions and τ is a set of t-propositions.

The Event Calculus domain described in Section 2.1 might be described as an $\mathcal{E}$ domain description D_R as follows. For action and fluent constants we would have $\Delta = \{Insert, GoThrough, Pickup\}$ and $\Phi = \{Inside, HasKey, Locked\}$ respectively. For Π and $\preceq$ we would use the real numbers with the usual ordering relation. Axioms (R1)–(R4) would be expressed in D_R as:

$$
\begin{gathered}
Pickup \textbf{ initiates } HasKey \\
Insert \textbf{ initiates } Locked \textbf{ when } \{\neg Locked, HasKey\} \\
GoThrough \textbf{ initiates } Inside \textbf{ when } \{\neg Locked, \neg Inside\} \\
GoThrough \textbf{ terminates } Inside \textbf{ when } \{\neg Locked, Inside\} \\
Insert \textbf{ terminates } Locked \textbf{ when } \{Locked, HasKey\} \\
Locked \textbf{ holds-at } 0 \\
Inside \textbf{ holds-at } 0 \\
Pickup \textbf{ happens-at } 2 \\
Insert \textbf{ happens-at } 4 \\
GoThrough \textbf{ happens-at } 6
\end{gathered}
$$

(The reader may also find it useful to compare this collection of propositions with axioms (R5)–(R12) in Section 2.2.)

The semantics of $\mathcal{E}$ is based on simple definitions of interpretations and models. Since the primary interest is in inferences about what holds at particular time-points in Π, it is sufficient to define an interpretation as a mapping of fluent/time-point pairs to *true* or *false* (i.e. a "*holds*" relation). An interpretation *satisfies* a fluent literal or set of fluent literals at a particular time-point if it assigns the relevant truth values to each of the corresponding fluent constants:

Definition 6 (Interpretation). *An interpretation of $\mathcal{E}$ is a mapping*

$$H : \Phi \times \Pi \mapsto \{true, false\}$$

Definition 7 (Point satisfaction). *Given a set of fluent literals C of $\mathcal{E}$ and a time point $T \in \Pi$, an interpretation H satisfies C at T iff for each fluent constant $F \in C$, $H(F,T) = true$, and for each fluent constant F' such that $\neg F' \in C$, $H(F',T) = false$.*

The definition of a model in $\mathcal{E}$ is parametric on the definitions of an *initiation point* and a *termination point*. Initiation and termination points are simply time-points where a c-proposition and an h-proposition combine to describe a direct effect on a particular fluent:

Definition 8 (Initiation/termination point). *Let H be an interpretation of $\mathcal{E}$, let $D = \langle \gamma, \eta, \tau \rangle$ be a domain description, let $F \in \Phi$ and let $T \in \Pi$. T is an initiation-point (respectively termination-point) for F in H relative to D iff there is an $A \in \Delta$ such that (i) there is both an h-proposition in η of the form "A* **happens-at** *T" and a c-proposition in γ of the form "A* **initiates** *F* **when** *C" (respectively "A* **terminates** *F* **when** *C") and (ii) H satisfies C at T.*

For an interpretation to qualify as a model, three basic properties need to be satisfied; (1) fluents change their truth values only via occurrences of initiating or terminating actions, (2) initiating a fluent establishes its truth value as *true*, and (3) terminating a fluent establishes its truth value as *false*. In addition, (4) every model must match with each of the t-propositions in the domain description:

Definition 9 (Model). *Given a domain description $D = \langle \gamma, \eta, \tau \rangle$ in $\mathcal{E}$, an interpretation H of $\mathcal{E}$ is a model of D iff, for every $F \in \Phi$ and $T, T', T_1, T_3 \in \Pi$ such that $T_1 \prec T_3$, the following properties hold:*

1. *If there is no initiation-point or termination-point T_2 for F in H relative to D such that $T_1 \preceq T_2 \prec T_3$, then $H(F, T_1) = H(F, T_3)$.*
2. *If T_1 is an initiation-point for F in H relative to D, and there is no termination-point T_2 for F in H relative to D such that $T_1 \prec T_2 \prec T_3$, then $H(F, T_3) = true$.*
3. *If T_1 is a termination-point for F in H relative to D, and there is no initiation-point T_2 for F in H relative to D such that $T_1 \prec T_2 \prec T_3$, then $H(F, T_3) = false$.*
4. *For all t-propositions in τ of the form "F **holds-at** T", $H(F, T) = true$, and for all t-propositions of the form "$\neg F$ **holds-at** T'", $H(F, T') = false$.*

Definition 10 (Consistency). *A domain description is consistent iff it has a model.*

Definition 11 (Entailment). *A domain description D entails the t-proposition "F **holds-at** T", written[7] "$D \models_{\mathcal{E}} F$ **holds-at** T", iff for every model H of D, $H(F, T) = true$. D entails the t-proposition "$\neg F$ **holds-at** T" iff for every model H of D, $H(F, T) = false$.*

As regards the robot example, using the above definitions it is easy to see that

$$D_R \models_{\mathcal{E}} \neg \mathit{Inside} \textbf{ holds-at } 8$$

More generally, if time is taken as the integers or reals, Definitions 8 and 9 indicate that the Language $\mathcal{E}$ corresponds to the "deterministic" Event Calculus described in Section 3.2, i.e. with domain independent axioms (EC1), (EC2), (EC3b), (EC4b), (EC5), (EC6), (EC9b) and (EC10b). Specifically condition 1 of Definition 9 mirrors axioms (EC1), (EC2), (EC5) and (EC6), condition 2 mirrors (EC3b) and (EC9b), and condition 3 mirrors (EC4b) and (EC10b). This correspondence is established more formally in the next section.

[7] The symbol $\models_{\mathcal{E}}$ is used here to distinguish Language $\mathcal{E}$ entailment from entailment in classical logic. It is identical in meaning to the symbol $\models$ used in other publications concerning the Language $\mathcal{E}$.

4.2 Translating Between the Event Calculus and $\mathcal{E}$

Clearly, for some domains (such as the robot example) translation from the Event Calculus to $\mathcal{E}$ (and vice versa) is straightforward. Equally clearly, for some other Event Calculus theories, perhaps with disjunctive or existentially quantified sentences partially defining *Initiates*, *Terminates*, *Happens* or *HoldsAt* (e.g. the robot example extended with (R13)), a translation into the restricted syntax of $\mathcal{E}$ is not possible. But it is difficult and cumbersome in general to describe necessary and sufficient syntactic conditions whereby an Event Calculus theory can be translated into an equivalent Language $\mathcal{E}$ domain description.

To illustrate, consider the following Event Calculus description of a "millennium counter" – a display of the minutes passed since 12 midnight on 31 December 2000. Time is taken as the integers, where each integer represents one second and 0 represents 12 midnight, 31 December 2000. An action *Tick* happens once every 60 seconds and increments the display by 1:

$$\begin{aligned} &\mathit{Initiates}(a,f,t) \equiv [a=\mathit{Tick} \wedge \exists n.[f=\mathit{Display}(n) \\ &\qquad \wedge \mathit{HoldsAt}(\mathit{Display}(n-1),t)]] \end{aligned}$$

$$\begin{aligned} &\mathit{Terminates}(a,f,t) \equiv [a=\mathit{Tick} \wedge \exists n.[f=\mathit{Display}(n) \\ &\qquad \wedge \mathit{HoldsAt}(\mathit{Display}(n),t)]] \end{aligned}$$

$$\mathit{Happens}(a,t) \equiv [a=\mathit{Tick} \wedge \exists t'.[t=(t'*60)]]$$

$$\mathit{HoldsAt}(\mathit{Display}(0),0) \wedge \forall n.[n\neq 0 \rightarrow \neg \mathit{HoldsAt}(\mathit{Display}(n),0)]$$

This axiomatisation might at first seem problematic as regards translation into $\mathcal{E}$; it entails an infinite number of positive ground *Initiates*, *Terminates* and *Happens* literals and (even without augmentation with domain independent Event Calculus axioms) an infinite number of negative ground *HoldsAt* literals (at $t = 0$). All of these need explicit representation in $\mathcal{E}$. But the following (infinite) Language $\mathcal{E}$ domain description $\langle \gamma, \eta, \tau \rangle$ is well defined and clearly entails the same collection of "holds at" facts along the time line:

$$\begin{aligned} \gamma = \; &\{\mathit{Tick} \textbf{ terminates } \mathit{Display}(n) \textbf{ when } \{\mathit{Display}(n)\} \mid n \in \mathbb{Z}\} \\ &\cup \\ &\{\mathit{Tick} \textbf{ initiates } \mathit{Display}(n) \textbf{ when } \{\mathit{Display}(m)\} \mid \\ &\qquad\qquad n,m \in \mathbb{Z} \text{ and } n=m+1\} \end{aligned}$$

$$\eta = \{\mathit{Tick} \textbf{ happens-at } (t*60) \mid t \in \mathit{\Pi}\}$$

$$\begin{aligned} \tau = \; &\{\mathit{Display}(0) \textbf{ holds-at } 0\} \\ &\cup \\ &\{\neg \mathit{Display}(n) \textbf{ holds-at } 0 \mid n \in \mathbb{Z} \text{ and } n\neq 0\} \end{aligned}$$

This example illustrates that any general syntactic constraints that we place on Event Calculus theories in order to ensure that they are translatable into $\mathcal{E}$ are likely to be over-restrictive. In what follows, we therefore instead concentrate on establishing a collection of sufficient (and intuitive) "semantic" constraints for a correct translation to be possible. Each of these will in most cases be straightforward to check from the form of the axiomatisation in question. Precisely what we mean by a "correct translation" is established in Proposition 1.

In Definitions 12 to 20 and Proposition 1 that follow, we will assume that $D = \langle \gamma, \eta, \tau \rangle$ is a Language $\mathcal{E}$ domain description written in the language $\langle \Pi, \leq, \Delta, \Phi \rangle$ (where Π is either $\mathbb{Z}$ or $\mathbb{R}$). We will also assume that T_{EC} is a collection of (domain dependent) axioms written in a sorted predicate calculus language of the type described in Section 2 that constrains the interpretation of the sort $\mathcal{T}$ to be Π, and that T_{EC} does not mention the predicates *Clipped*, *Declipped*, *StoppedIn* and *StartedIn*. Furthermore we will assume that the language of T_{EC} includes all symbols in Δ as ground terms of sort $\mathcal{A}$ and all symbols in Φ as ground terms of sort $\mathcal{F}$. **Notation:** We will denote as $\Phi^{\pm}$ the set of all (positive and negative) fluent literals that can be formed from the fluent constants in Φ. Given a model M of T_{EC}, $\|G\|_M$ will denote the interpretation (i.e. the denotation) of the ground term or symbol G in M. We will refer to the set of domain independent Event Calculus axioms {(EC1), (EC2), (EC3b), (EC4b), (EC5), (EC6), (EC9b), (EC10b)} (see Sections 2 and 3.2) as Det_{EC}.

The first condition to express is that (in all its models) T_{EC} establishes uniqueness of names for the fluents and actions referred to in D:

Definition 12 (Name-matches). *D name-matches T_{EC} iff for every model M of T_{EC}, for every $F, F' \in \Phi$ and for every $A, A' \in \Delta$,*

- *if $F \neq F'$ then $\|F\|_M \neq \|F'\|_M$, and*
- *if $A \neq A'$ then $\|A\|_M \neq \|A'\|_M$.*

Typically this name-matches property might be established by a collection of inequality statements in T_{EC} between ground fluent and action literals (e.g. $Inside \neq HasKey$, etc. in the Robot example) or by universally quantified implications such as $\forall m, n.[Display(m) = Display(n) \rightarrow m = n]$.

The next condition to establish (Definitions 13 to 16 below) is that all interpretations of *Initiates*, *Terminates* and *Happens* licensed by T_{EC} are isomorphic to the unique interpretation (relative to the interpretation of *HoldsAt*) explicitly indicated by the c- and h-propositions in D:

Definition 13 (h-satisfies). *Given a model M of T_{EC}, a time-point $T \in \Pi$ and a set $C \subseteq \Phi^{\pm}$ of Language $\mathcal{E}$ fluent literals, M h-satisfies C at T iff for all $F \in \Phi$, if $F \in C$ then $\langle \|F\|_M, T \rangle \in \|HoldsAt\|_M$, and if $\neg F \in C$ then $\langle \|F\|_M, T \rangle \notin \|HoldsAt\|_M$.*

Definition 14 (Initiates-matches). *D initiates-matches T_{EC} iff for every model M of T_{EC}, every time-point T and every action α and fluent ϕ in the domain of discourse of M the following holds. $\langle \alpha, \phi, T \rangle \in \|Initiates\|_M$ if and only if there exist $F \in \Phi$, $A \in \Delta$ and $C \subseteq \Phi^{\pm}$ such that $\alpha = \|A\|_M$, $\phi = \|F\|_M$, M h-satisfies C at T, and "A **initiates** F **when** C" $\in \gamma$.*

Definition 15 (Terminates-matches). *D terminates-matches T_{EC} iff for every model M of T_{EC}, every time-point T and every action α and fluent ϕ in the domain of discourse of M the following holds. $\langle \alpha, \phi, T \rangle \in \|Terminates\|_M$ if and only if there exist $F \in \Phi$, $A \in \Delta$ and $C \subseteq \Phi^{\pm}$ such that $\alpha = \|A\|_M$, $\phi = \|F\|_M$, M h-satisfies C at T, and "A **terminates** F **when** C" $\in \gamma$.*

Definition 16 (Happens-matches). *D happens-matches T_{EC} iff for every model M of T_{EC}, every time-point T and every action α in the domain of discourse of M the following holds. $\langle \alpha, T \rangle \in \|Happens\|_M$ if and only if there exists $A \in \Delta$ such that $\alpha = \|A\|_M$ and "A **happens-at** T" $\in \eta$.*

Finally, it is necessary to establish that (without the domain independent Event Calculus axioms in Det_{EC}), T_{EC} imposes exactly the same collection of pointwise constraints on the interpretation of *HoldsAt* that are indicated by the t-propositions in D. To do this it is necessary to impose a domain closure property on fluent names (the first condition in Definition 19). It is also necessary to ensure that T_{EC} does not entail any extra "global dependencies" not captured by the t-propositions of D, either between two or more fluents (e.g. $\forall t.[HoldsAt(HasKey, t) \rightarrow HoldsAt(Inside, t)]$), or between fluents and other facts represented in T_{EC} (e.g. $\forall t.[HoldsAt(HasKey, t) \rightarrow SmallEnoughToHold(Key)]$). This is guaranteed by the third condition in Definition 19.

Definition 17 (t-model). *An interpretation H of $\mathcal{E}$ is a t-model of D iff, for every $F \in \Phi$ and $T, T' \in \Pi$, for all t-propositions in τ of the form "F **holds-at** T", $H(F, T) = true$, and for all t-propositions of the form "$\neg F$ **holds-at** T'", $H(F, T') = false$.*

Definition 18 ($\mathcal{E}$-projection). *The $\mathcal{E}$-projection of a model M of T_{EC} is defined as the following (Language $\mathcal{E}$) interpretation H_M:*

$$H_M(F,T) = \begin{cases} true & if \ \ \langle \|F\|_M, T \rangle \in \|HoldsAt\|_M \\ false & otherwise \end{cases}$$

Definition 19 (Holds-matches). *D holds-matches T_{EC} iff for every model M of T_{EC} the following conditions are satisfied:*

- *for every fluent ϕ in the domain of discourse of M there exists $F \in \Phi$ such that $\phi = \|F\|_M$,*
- *the $\mathcal{E}$-projection of M is a t-model of D,*
- *For every t-model H^t of D there is a model M^{H^t} of T_{EC} which differs from M only in the interpretation of HoldsAt and is such that H^t is the $\mathcal{E}$-projection of M^{H^t}.*

Definition 20 (matches). *D matches T_{EC} iff D name-matches, initiates-matches, terminates-matches, happens-matches and holds-matches T_{EC}.*

Proposition 1. *Let $F \in \Phi$ and let $T \in \Pi$. If T_{EC} is consistent and D matches T_{EC} then:*

– $D \models_{\mathcal{E}} F$ **holds-at** T *iff* $T_{EC} \cup Det_{EC} \models HoldsAt(F,T)$
– $D \models_{\mathcal{E}} \neg F$ **holds-at** T *iff* $T_{EC} \cup Det_{EC} \models \neg HoldsAt(F,T)$

Proof. It is sufficient to prove the following:

1. If there exists a model H of D such that $H(F,T) = true$ then there exists a model M^H of $T_{EC} \cup Det_{EC}$ such that $M^H \| \sim HoldsAt(F,T)$.
2. If there exists a model M of $T_{EC} \cup Det_{EC}$ such that $M \| \sim HoldsAt(F,T)$ then there exists a model H_M of D such that $H_M(F,T) = true$.
3. If there exists a model H of D such that $H(F,T) = false$ then there exists a model M^H of $T_{EC} \cup Det_{EC}$ such that $M^H \| \sim \neg HoldsAt(F,T)$.
4. If there exists a model M of $T_{EC} \cup Det_{EC}$ such that $M \| \sim \neg HoldsAt(F,T)$ then there exists a model H_M of D such that $H_M(F,T) = false$.

Proof of (1):

If there exists a model H of D such that $H(F,T) = true$ then by Definitions 9 and 17 H is a t-model of D. Hence, since T_{EC} is consistent, by Definition 19 there exists a model M^H of T_{EC} such that H is the $\mathcal{E}$-projection of M^H. Therefore $M^H \| \sim HoldsAt(F,T)$. Since T_{EC} does not mention the predicates *Clipped*, *Declipped*, *StoppedIn* and *StartedIn* then clearly we can assume that M^H is such that it satisfies (EC1), (EC2), (EC9b) and (EC10b). Since D name-matches, initiates-matches, terminates-matches and happens-matches T_{EC} then by condition 1 of Definition 9 M^H satisfies (EC5) and (EC6), by condition 2 of Definition 9 M^H satisfies (EC3b), and by condition 3 of Definition 9 M^H satisfies (EC4b). Therefore M^H is a model of $T_{EC} \cup Det_{EC}$.

Proof of (2):

If there exists a model M of $T_{EC} \cup Det_{EC}$ such that $M \| \sim HoldsAt(F,T)$, then by Definition 19 the $\mathcal{E}$-projection H_M of M is a t-model of D and $H_M(F,T) = true$. It remains to show that H_M satisfies conditions 1, 2 and 3 of Definition 9. Since D name-matches, initiates-matches, terminates-matches and happens-matches T_{EC}, it follows directly from the fact that $M \| \sim [(\text{EC5}) \wedge (\text{EC6})]$ that H_M satisfies condition 1 of Definition 9, it follows directly from the fact that $M \| \sim (\text{EC3b})$ that H_M satisfies condition 2 of Definition 9, and it follows directly from the fact that $M \| \sim (\text{EC4b})$ that H_M satisfies condition 1 of Definition 9.

Proof of (3):

This is identical to the proof of (1), but substituting "$H(F,T) = false$" for "$H(F,T) = true$" and substituting "$M^H \| \sim \neg HoldsAt(F,T)$" for "$M^H \| \sim HoldsAt(F,T)$".

Proof of (4):

This is identical to the proof of (2), but substituting "$M \| \sim \neg HoldsAt(F,T)$" for "$M \| \sim HoldsAt(F,T)$" and substituting "$H_M(F,T) = false$" for "$H_M(F,T) = true$".

(end of proof of Proposition 1)

Proposition 1 is analogous in some respects to the results in [27], which show the equivalence of various classical logic formulations of the Situation Calculus to the Language $\mathcal{A}$. But whereas the conditions for the results in [27] are syntactic, those for Proposition 1 are semantic and so less restrictive. Although checking through all the conditions for Proposition 1 to hold might at first sight seem tedious, in many cases the fact that a collection of domain dependent axioms "matches" a Language $\mathcal{E}$ domain description will be obvious. In particular, it is clear that any Language $\mathcal{E}$ domain description written using only a finite number of action and fluent constants can be straightforwardly translated into an Event Calculus axiomatisation by formulating sentences analogous to (R1) – (R4) (see Section 2.1).

As stated earlier, Proposition 1 is useful because it allows the (deterministic) classical logic Event Calculus to take advantage of the provably correct automated reasoning procedures developed for $\mathcal{E}$ (see [21],[22],[23],[24]). Of these implementations, the most flexible is that described in [23, 24], which is based on a sound and complete translation of $\mathcal{E}$ into an argumentation framework. The resulting implementation E-RES [24] [26] allows reasoning backwards and forwards along the time line even in cases where information about what holds in the "initial state" (i.e. before any action occurrences) is incomplete. E-RES has been further extended into an abductive planning system [25] able to produce plans and conditional plans even with incomplete information about the status of fluents along the time line.

5 Summary

In this article, we have described a basic, classical logic variation of the Event Calculus, and then summarised previous work on how this axiomatisation may be adapted and/or extended in various ways to represent various features of particular domains. In particular, we have described versions of the Event Calculus able to incorporate non-deterministic actions, concurrent actions, action preconditions and qualifications, delayed actions and effects, actions with duration, gradual and continuous change, and mathematical models using sets of simultaneous differential equations. We have also shown how one particular version of the basic Event Calculus may be given a sound and complete translation into the Language $\mathcal{E}$ and thus inherit $\mathcal{E}$'s provably correct automated reasoning procedures.

References

1. A. Baker, *Nonmonotonic Reasoning in the Framework of the Situation Calculus*, Artificial Intelligence, Vol 49(5-23), 1991.
2. I. Cervesato, L. Chittaro and A. Montanari, *A Modal Calculus of Partially Ordered Events in a Logic Programming Framework*, in Proceedings ICLP'95, MIT Press, pages 299-313, 1995.

3. I. Cervesato, L. Chittaro and A. Montanari, *A General Modal Framework for the Event Calculus and its Skeptical and Credulous Variants*, in in W. Wahlster, editor, Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96), pp. 33-37, John Wiley and Sons, 1996.
4. I. Cervesato, M. Franceschet and A. Montanari, *A Hierarchy of Modal Event Calculi: Expressiveness and Complexity*, in H.Barringer *et al*, Proceedings of the 2nd International Conference on Temporal Logic (ICTL'97, pp. 1-17, Kluwer Applied Logic Series, 1997.
5. I. Cervesato, M. Franceschet and A. Montanari, *Modal Event Calculi with Preconditions*, in R. Morris and L. Khatib, Proceedings of the Fourth International Workshop on Temporal Reasoning (TIME'97), pp. 38-45, IEEE Computer Society Press, 1997.
6. I. Cervesato, M. Franceschet and A. Montanari, *The Complexity of Model Checking in Modal Event Calculi with Quantifiers*, Journal of Electronic Transactions on Artificial Intelligence, Linköping University Electronic Press, `http://www.ida.liu.se/ext/etai/`, 1998.
7. L. Chittaro, A. Montanari and A. Provetti, *Skeptical and Credulous Event Calculi for Supporting Modal Queries*, in A. Cohn, Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94), pp. 361-365, John Wiley and Sons, 1994.
8. N. Chleq, *Constrained Resolution and Abductive Temporal Reasoning*, Computational Intelligence, vol. 12, no. 3, pp. 383406, 1996.
9. J. M. Crawford and D. W. Etherington, *Formalizing Reasoning about Change: A Qualitative Reasoning Approach*, Proceedings AAAI'92, pp. 577-583, 1992.
10. J. Davila, *Reactive Pascal and the Event Calculus*, Proceedings FAPR'96 Workshop on Reasoning about Actions and Planning in Complex Environments, eds. U. Siegmund and M. Thielscher, vol. 11 of Technical Report AIDA, 1996.
11. M. Denecker, L. Missiaen and M. Bruynooghe, *Temporal Reasoning with Abductive Event Calculus*, in Proceedings ECAI 92, Vienna, 1992.
12. M. Denecker, K. Van Belleghem, G. Duchatelet, F. Piessens and D. De Schreye *A Realistic Experiment in Knowledge Representation in Open Event Calculus : Protocol Specification*, in Proceedings of the Joint International Conference and Symposium on Logic Programming, 1996.
13. M. Denecker, D. Theseider Dupré, and K. Van Belleghem, *An Inductive Definition Approach to Ramifications*, in Electronic Transactions on Artificial Intelligence, vol 2, 1998.
14. P. Doherty, *Reasoning about Action and Change Using Occlusion*, Proceedings ECAI'94, pp. 401-405, 1994.
15. K. Eshghi, *Abductive Planning with Event Calculus*, Proceedings of the 5th International Conference and Symposium on Logic Programming, ed.s Robert Kowalski and Kenneth Bowen, MIT Press, pp. 562-579, 1988.
16. M. Gelfond and V. Lifschitz, *Representing Actions in Extended Logic Programming*, JICSLP'92, ed. Krzysztof Apt, 560, MIT Press, 1992.
17. M. Gelfond and V. Lifschitz, *Representing Action and Change by Logic Programs*, JLP, 17 (2,3,4) 301–322, 1993.
18. R. C. Jeffrey, *Formal Logic: Its Scope and Limits*, McGraw-Hill, 1967.
19. C. G. Jung, K. Fischer and A. Burt, *Multi-Agent Planning Using an Abductive Event Calculus*, DFKI Report RR-96-04 (1996), DFKI, Germany, 1996.
20. C. G. Jung, *Situated Abstraction Planning by Abductive Temporal Reasoning*, Proceedings ECAI'98, pp. 383387, 1998.

21. A. Kakas and R. Miller, *A Simple Declarative Language for Describing Narratives with Actions*, JLP 31(1–3) (Special Issue on Reasoning about Action and Change) 157–200, 1997.
22. A. Kakas and R. Miller, *Reasoning about Actions, Narratives and Ramifications*, Journal of Electronic Transactions on Artificial Intelligence 1(4), Linköping University Electronic Press, `http://www.ida.liu.se/ext/etai/`, 1998.
23. A. Kakas, R. Miller and F. Toni, *An Argumentation Framework for Reasoning about Actions and Change*, Proceedings of LPNMR'99, 1999.
24. A. Kakas, R. Miller and F. Toni, *E-RES – A System for Reasoning about Actions, Events and Observations*, Proceedings of NMR 2000, Special Session on System Demonstrations and Descriptions, `http://xxx.lanl.gov/abs/cs.AI/0003034`, 2000.
25. A. Kakas, R. Miller and F. Toni, *Planning with Incomplete Information*, Proceedings of NMR 2000, Special Session on Representing Actions and Planning, `http://xxx.lanl.gov/abs/cs.AI/0003049`, 2000.
26. A. Kakas, R. Miller and F. Toni, *E-RES - Reasoning about Actions, Events and Observations*, Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'2001), September 17-19, 2001, Vienna, Austria, ed. T. Eiter, M. Truszczynski and W. Faber, pub. Springer-Verlag (LNCS/LNAI series), 2001.
27. G. N. Kartha, *Soundness and Completeness Theorems for Three Formalizations of Action*, Proceedings IJCAI'93, page 724, 1993.
28. G. N. Kartha and V. Lifschitz, *A Simple Formalization of Actions Using Circumscription*, Proceedings IJCAI'95, pp. 1970-1975, 1995.
29. R. A. Kowalski, *Database Updates in the Event Calculus*, Journal of Logic Programming, vol. 12, pp. 121-146, 1992.
30. R. A. Kowalski, *Legislation as Logic Programs*, Informatics and the Foundations of Legal Reasoning, Kluwer Academic Publishers, ed.s Z. Bankowski et al., pp. 325-356, 1995.
31. R. A. Kowalski and F. Sadri, *The Situation Calculus and Event Calculus Compared*, in Proceedings of the International Logic Programming Symposium (ILPS'94), 1994.
32. R. A. Kowalski and F. Sadri, *Reconciling the Event Calculus with the Situation Calculus*, Journal of Logic Programming, Special Issue on Reasoning about Action and Change, vol. 31, pp. 39-58, 1997.
33. R. A. Kowalski and M. J. Sergot, *A Logic-Based Calculus of Events*, New Generation Computing, vol. 4, pp. 67-95, 1986.
34. B. Kuipers, *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*, MIT Press, 1994.
35. F. Lévy and Joachim Quantz, *Representing Beliefs in a Situated Event Calculus*, Proceedings ECAI'98, pp. 547551, 1998.
36. H. Levesque, *What is Planning in the Presence of Sensing?*, in Proceedings of AAAI'96, 1996.
37. 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, pp. 297-352, 1994.
38. J. Lobo, G. Mendez and S. Taylor, *Adding Knowledge to the Action Description Language A*, in Proceedings of AAAI'97, 1997.
39. J. McCarthy, *Circumscription A Form of Non-Monotonic Reasoning*, Artificial Intelligence, vol. 13, pp. 27-39, 1980.

40. 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, pp. 463-502, 1969.
41. R. Miller, *Situation Calculus Specifications for Event Calculus Logic Programs*, in Proceedings of the Third International Conference on Logic Programming and Non-monotonic Reasoning, Lexington, KY, USA, Springer Verlag, 1995.
42. R. S. Miller and M. P. Shanahan, *Reasoning about Discontinuities in the Event Calculus*, Proceedings 1996 Knowledge Representation Conference (KR'96), pp. 6374, 1996.
43. R. S. Miller and M. P. Shanahan, *The Event Calculus in Classical Logic - Alternative Axiomatisations*, Journal of Electronic Transactions on Artificial Intelligence, Vol. 3 (1999), Section A, pages 77-105, `http://www.ep.liu.se/ej/etai/1999/016/`, 1999.
44. L. R. Missiaen, *Localized Abductive Planning for Robot Assembly*, Proceedings 1991 IEEE Conference on Robotics and Automation, pub. IEEE Robotics and Automation Society, pages 605-610, 1991.
45. L. R. Missiaen, M. Denecker and M. Bruynooghe, *An Abductive Planning System Based on Event Calculus*, Journal of Logic and Computation, volume 5, number 5, pages 579–602, 1995.
46. R. C. Moore, *A Formal Theory of Knowledge and Action*, In Hobbs and Moore, ed.s, Formal Theories of the Commonsense World, Ablex, Norwood, USA, 1985.
47. L. Morgenstern, *Knowledge Preconditions for Actions and Plans*, in Proceedings of the International Joint Conference in Artificial Intelligence 1987 (IJCAI'97), Morgan Kaufmann, 1987.
48. J. Pinto and R. Reiter, *Temporal Reasoning in Logic Programming: A Case for the Situation Calculus*, Proceedings ICLP 93, page 203, 1993.
49. A. Provetti, *Hypothetical Reasoning about Actions: From Situation Calculus to Event Calculus*, Computational Intelligence, volume 12, number 2, 1995.
50. 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, pp. 359-380, 1991.
51. R. Reiter, *Natural actions, concurrency and continuous time in the situation calculus*, in Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96), Cambridge, Massachusetts, U.S.A, November 5-8, 1996.
52. A. Russo, R. Miller, B. Nuseibeh and J. Kramer, *An Abductive Approach for Handling Inconsistencies in SCR Specifications*, in proceedings of the 3rd International Workshop on Intelligent Software Engineering (WISE3), Limerick, Ireland, June, 2000.
53. F. Sadri and R. Kowalski, *Variants of the Event Calculus*, Proceedings of the International Conference on Logic Programming, Kanagawa, Japan, Stirling L. (Ed), The MIT Press, pp. 67-81, 1995.
54. E. Sandewall, *Combining Logic and Differential Equations for Describing Real World Systems*, Proceedings KR'89, Morgan Kaufman, 1989.
55. E. Sandewall, *Filter Preferential Entailment for the Logic of Action in Almost Continuous Worlds*, Proceedings IJCAI'89, pages 894-899, 1989.
56. E. Sandewall, *The Representation of Knowledge about Dynamical Systems, Volume 1*, Oxford University Press, 1994.
57. R. Scherl and H. Levesque, *The Frame Problem and Knowledge-Producing Actions*, in Proceedings of AAAI'93, 1993.

58. M. P. Shanahan, *Representing Continuous Change in the Event Calculus*, Proceedings ECAI'90, pp. 598-603, 1990.
59. M. P. Shanahan, *A Circumscriptive Calculus of Events*, Artificial Intelligence, vol 77 (1995), pages 249-284, 1995.
60. M. P. Shanahan, *Robotics and the Common Sense Informatic Situation*, Proceedings ECAI'96, pp. 684-688, 1996.
61. M. P. Shanahan, *Noise and the Common Sense Informatic Situation for a Mobile Robot*, Proceedings AAAI'96, pp. 1098-1103, 1996.
62. M. P. Shanahan, *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*, MIT Press, 1997.
63. M. P. Shanahan, *Event Calculus Planning Revisited*, Proceedings 4th European Conference on Planning (ECP'97), Springer Lecture Notes in Artificial Intelligence no. 1348, pp. 390-402,1997.
64. M. P. Shanahan, *Noise, Non-Determinism and Spatial Uncertainty*, Proceedings AAAI'97, pp. 153-158, 1997.
65. M. P. Shanahan, *Reinventing Shakey*, Working Notes of the 1998 AAAI Fall Symposium on Cognitive Robotics, pp. 125-135, 1998.
66. M. P. Shanahan, *The Ramification Problem in the Event Calculus*, Proceedings IJCAI'99, 1999.
67. M. P. Shanahan, *A Logical Account of the Common Sense Informatic Situation for a Mobile Robot*, Electronic Transactions on Artificial Intelligence, 1999.
68. M. P. Shanahan, *The Event Calculus Explained*, in Artificial Intelligence Today, eds. M. J. Wooldridge and M. Veloso, Springer-Verlag Lecture Notes in Artificial Intelligence no. 1600, Springer-Verlag, pages 409-430, 1999.
69. E. Ternovskaia, *Inductive Definability and the Situation Calculus*, in "Transactions and Change in Logic Databases", Lecture Notes in Computer Science, volume 1472, Ed. Freitag B., Decker H., Kifer M. (Eds.), pub. Springer Verlag, 1997.
70. E. Ternovskaia, *Causality via Inductive Definitions*, in Working Notes of "Prospects for a Commonsense Theory of Causation", pages 94-100, AAAI Spring Symposium Series, March 23-28, 1998.
71. K. Van Belleghem, M. Denecker and D. De Schreye, *Representing Continuous Change in the Abductive Event Calculus*, in Proceedings 1994 International Conference on Logic Programming, ed. P. Van Hentenrijck, pages 225-240, 1994.
72. K. Van Belleghem, M. Denecker and D. De Schreye, *The Abductive Event Calculus as a General Framework for Temporal Databases*, Proceedings of the International Conference on Temporal Logic, 1994.
73. K. Van Belleghem, M. Denecker and D. De Schreye, *Combining Situation Calculus and Event Calculus*, in Proceedings of the International Conference on Logic Programming, 1995.
74. K. Van Belleghem, M. Denecker and D. De Schreye, *On the Relation Between Situation Calculus and Event Calculus*, Journal of Logic Programming, 31(1–3) (Special Issue on Reasoning about Action and Change), 1996.