Temporal Reasoning in Logic Programming: A Case for the Situation Calculus.

Javier Pinto
Dept. of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
javier@ai.toronto.edu

Raymond Reiter*
Dept. of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
reiter@ai.toronto.edu

Abstract

We propose, and axiomatize, an extended version of the situation calculus [10] for temporal reasoning in a logic programming framework. This extended language provides for a linear temporal structure, which may be viewed as a path of actual event occurrences within the tree of possible situations of the "classical" situation calculus. The extended language provides for events to *occur* and fluents to *hold* at specific points in time. As a result, it is possible to establish a close correspondence between this extended situation calculus and other linear time formalisms which have been proposed as alternatives to the situation calculus.

In particular, we argue that the functionality of the *event calculus* [6] is subsumed by the extended situation calculus. We present a logic program for temporal reasoning which is provably sound for our axiomatization, relative to the Clark completion semantics of the program. Our logic programming approach has the advantage of being grounded in a pure (without negation as failure) first order axiomatization suitable for reasoning about events and their occurrences. Moreover, efficient algorithms can be obtained for a suitable class of temporal reasoning problems, following the ideas of Kowalski [5].

Keywords: Temporal Reasoning, Knowledge Representation, Situation Calculus, Event Calculus.

1 Introduction and Motivation.

The situation calculus was originally introduced by McCarthy and Hayes [9] as a first order language appropriate for reasoning about actions. The basic ontology of the situation calculus consists of *situations*, which correspond to snapshots of the universe at an instant of time, and *actions* or events, which change the world from one state to another. It is a sorted language with sorts for situations and actions, and it has a distinguished function *do* (or *result*) that, given a situation s and an action a, denotes the situation *result*ing from *do*ing a in s. The situation calculus has been criticized on the grounds that it is not general enough to deal with problems in the real world. However, a recent surge of interest in the situation calculus has shown that most of the criticisms are unjustified. For example, Gelfond, Lifschitz and Rabinov [4] show that by extending the language to deal with a generalized notion of action one can represent concurrency, non-instantaneous actions, conditional actions, etc. Lin and Shoham [7] show how to incorporate concurrent actions and discuss the notion of independence among actions. Levesque, Lin and Reiter[1] propose an extension

*Fellow of the Canadian Institute for Advanced Research.
[1]Forthcoming.

to the situation calculus in which complex actions can be built from primitive ones, providing Algol-like programming constructs as complex actions.

In this paper we consider Kowalski's [5] criticisms of the situation calculus for temporal reasoning. Kowalski claims several *disadvantages* for the situation calculus for representing event occurrences, specifically:

1. High computational cost of the frame axioms.
2. Events need to be totally ordered.
3. Information must be assimilated in chronological order.

In this paper, we argue that these disadvantages are not intrinsic to the situation calculus. We show that it is possible to deal with partially ordered events and that the language does not force us to assimilate knowledge in chronological order. Moreover, we argue that the same algorithm proposed by Kowalski for the *event calculus* can be used in a reasoning system based on the situation calculus.

The rest of the paper is organized as follows: In section 2, we analyze the *event calculus* and show that some ontological choices made in [6] have unintended consequences. In section 3, we show how, with a simple extension to the original situation calculus language, it is possible to express event occurrences and time. In section 4, we describe a logic program for reasoning about time and events based on the declarative specification of section 3; this we do for a class of problems for which complete information may be assumed. Under these circumstances, we prove that the program is sound with respect to its Clark completion semantics. Finally, in section 5, we present our conclusions and discuss extensions of the present research.

2 The Event Calculus Revisited.

The calculus of events was proposed as a general temporal logic framework. It was originally presented as a logic program [6], in which negation as failure plays an essential role. We have found it very difficult to ascribe a semantics to the calculus of events. In this section we point out some of the difficulties we have encountered with this proposal as a formal theory of time.

These difficulties arise from the use of *equality*, the use of *negation as failure*, and the interaction between negation as failure and incomplete knowledge.

The first problem is the treatment of temporal intervals. In the event calculus, two functions, `after` and `before`, are used to deal with time periods. Both are two-place functions that take an action and a fluent as arguments. According to Kowalski, a term of the form $\texttt{after}(\texttt{a}, \texttt{f})$ "names" a time period. Furthermore, according to Kowalski and Sergot [6, p.87], the sentence:

$$\texttt{Holds}(\texttt{p}) \tag{1}$$

"expresses that the relationship associated with `p` holds for the time period `p`." For example, we can write:

$$\texttt{Holds}(\texttt{after}(\texttt{paint}(\texttt{Green}), \texttt{colour}(\texttt{Green})))$$

that can be taken to mean that $\texttt{paint}(\texttt{Green})$ is an event that starts a time period in which the fluent $\texttt{colour}(\texttt{Green})$ is true. Now, if `p1` and `p2` are two time periods which are started and ended by the same events, then they must be different names for the same temporal interval. Otherwise, `p` in (1) could not denote a *temporal interval*. As a simple example, consider figure 1, in which we have three events `E1`, `E2` and `E3` occurring at times `T1`, `T2` and `T3` respectively. Moreover, suppose

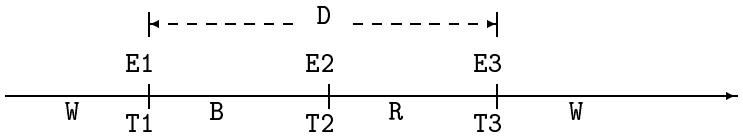

Figure 1: Simple Example

we have four fluents: `W` which holds only before `T1` and after `T3`; `B` which holds only between `T1` and `T2`; `R` which holds only between `T2` and `T3`; and `D` which holds only between `T1` and `T3`. Among others, the time period `after(E1,D)` holds; this is the same as the time periods $\langle$`T1` $-$ `T3`$\rangle$ and `before(E3,D)`. Now, from rule G3 in Kowalski and Sergot's program [6, p.88] we know that:

$$\texttt{start}(\texttt{after}(\texttt{e},\texttt{u}),\texttt{e}). \tag{2}$$

Therefore, we may infer that the time period `after(E1,R)` is started by `E1` at time `T1`. Now, we can easily provide axioms that make event `E2` terminate only `B` and nothing else. Therefore, the interval `after(E1,R)` would not be terminated by `E2`. However, we know that `E3` must terminate `R`; therefore we conclude that the time period denoted by `after(E1,R)` is terminated by `E3`. Hence, `after(E1,R)` is the time period $\langle$`T1` $-$ `T3`$\rangle$, which we know holds. We conclude that `Holds(after(E1,R))` must be true, which is clearly unintended (and contradictory).

This problem stems from the choice of ontology, and seems to be caused by the `Holds` predicate, which states that time periods hold (?!). In [5, p. 142], Kowalski suggests that for a very specific class of problems "time periods can be eliminated altogether." This is suggested for computational, rather than representational reasons. However, it seems that either time periods, or the `Holds` predicate should be eliminated from the ontology for the proposal to work.

Another problem arises from the use of negation as failure. Assume that we query a temporal reasoner based on the calculus of events with `holdsAt(F,T)`, i.e. we want to know whether some fluent `F` holds at time `T`. Furthermore, assume that we obtain a negative answer. How are we to interpret this answer? In general, we want a clear characterization of what the system's answers mean. In this case, there are two possible interpretations: (i) Assuming complete information about the events that have occurred, and assuming complete knowledge about the initial conditions and the effects of events, it is the case that `F` does not hold in `T`, (ii) We have been unable to ascertain whether `F` holds or not because of lack of information. It is simple to imagine examples of the first case. The second possibility arises, for example, with case 3 in [6, p.90], where two events `e` and `e`$'$ are known, the first one initiates a fluent `u` and the second one terminates a fluent `u`$'$, with `u` and `u`$'$ mutually exclusive. It turns out that some intervening event (or events) must have occurred between `e` and `e`$'$ which terminates `u` and initiates `u`$'$. The problem is that for any particular time point `T` after `e` and before `e`$'$, the query `holdsAt(u,T)` will fail. This is a problem inherent to the use of negation as failure, which is, in this case, too strong. Similar difficulties arise with concurrent events which are not precluded by the event calculus.

In the event calculus, problems also arise with negation as failure in the presence of incomplete information in settings having nothing to do with incompleteness about the temporal relations between events. The event calculus appeals to *case semantics*. The basic idea is that events are treated as first order objects which may be predicated by so-called *cases*. For example, events may have *agents*, *objects*, etc. An apparent advantage of *case semantics* is that event cases need not always be known in order to reason about events. Unfortunately, under negation as failure, this advantage is not always realized. For example, assume that John gave a book away, but it is not known to whom. Negation as failure will conclude that for each individual in the universe, it is not

the case that that individual owns the book. Therefore, nobody owns the book after John gave it to somebody.

It is well known that logic programming is inappropriate for reasoning with non-categorical theories. In fact, the extension of the logic programming paradigm to work with such theories is a very active area of research [3]. Therefore, it should not be surprising that the examples of overcommitment mentioned above arise. Our concern with the event calculus, formulated as it is as a logic program with negation as failure, is that it provides no clear guidelines to the programmer about when overcommitments can arise, which is to say, it does not characterize a class of *sound* programs. Of course, soundness is possible only relative to some prior, universally accepted specification of the task. This paper attempts to provide such a specification of the kinds of temporal reasoning tasks for which the event calculus was designed. In doing so, we shall appeal to an extension of the situation calculus, enriched with time and event occurrences. With this specification in hand, we shall be in a position to derive a logic program which is sound with respect to it.

It is important to point out that Murray Shanahan [14, 15, 16] has extended the calculus of events in several interesting ways. However, he bases his research on an event calculus which is loosely based on the original calculus of events. In fact, Shanahan's event calculus avoids the problems we describe in this section by eliminating the notion of temporal interval from the logical language altogether. His departure from the original calculus is more extreme in his later work [16] in which he does not rely on negation-as-failure as the *non-monotonic* approach to deal with the frame problem. It would be interesting to study the logic programming implementation of Shanahan's event calculus and how it relates to our proposal.

3 The Extended Situation Calculus.

3.1 The Basic Language and Axioms.

In this subsection we summarize the axiomatization and some results from [10]. We have sorts $\mathcal{A}, \mathcal{S}, \mathcal{F}, \mathcal{T}, \mathcal{D}$ for action types, situations, propositional fluents, time, and other domain objects respectively. Variables are denoted by lower case letters (with or without subscripts), and constants are denoted by upper case letters (with or without subscripts). Unless otherwise stated, letters a, s, f, t (A, S, F, T) are used for variables (constants) of sorts $\mathcal{A}, \mathcal{S}, \mathcal{F}, \mathcal{T}$ respectively. We will have a single 1-place predicate variable φ over situations. Other variables and constants will be assumed to be of sort $\mathcal{D}$. Also, free variables are assumed to be universally quantified. The sort T corresponds to the non-negative reals, extended to include ∞, for which we assume the standard interpretation.

Strictly speaking, our language is second order. However, the only second order sentence we use is the induction axiom (3). Therefore, we restrict ourselves to a standard sorted first order sublanguage (with equality) to express all other axioms of our theories.

We include the special constant S_0, denoting the initial state, of sort $\mathcal{S}$, the function $do : \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{S}$, the relation $< \subseteq \mathcal{S} \times \mathcal{S}$, along with other functions and relations introduced later.[2]

Basic Axioms:

$$(\forall \varphi).[\varphi(S_0) \wedge (\forall s, a).(\varphi(s) \supset \varphi(do(a, s)))] \supset (\forall s).\varphi(s), \tag{3}$$

[2]We use the abbreviation $s_1 \leq s_2$ to mean $s_1 < s_2 \vee s_1 = s_2$. Also we use the notation $a < b < c$ to mean $a < b \wedge b < c$, and similarly when the expression combines $<$ and $\leq$.

$$(\forall s_1, s_2, a).s_1 < do(a, s_2) \equiv s_1 \leq s_2, \tag{4}$$
$$(\forall a_1, a_2, s_1, s_2).do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \tag{5}$$
$$(\forall s_1, s_2).s_1 < s_2 \supset \neg s_2 < s_1. \tag{6}$$

Axiom (3) is an induction axiom, necessary to prove properties true in all situations [13]. Notice that axiom (5) precludes concurrent actions.

Proposition 3.1 *Some consequences of these axioms are:*

$$(\forall s).S_0 \leq s, \tag{7}$$
$$(\forall s).s \neq S_0 \supset (\exists a, s').s = do(a, s'),$$
$$(\forall s_1, s_2, s_3).s_1 < s_2 \wedge s_2 < s_3 \supset s_1 < s_3, \tag{8}$$
$$(\forall a, s).s < do(a, s), \tag{9}$$
$$(\forall a, s).\neg s < s,$$
$$(\forall a_1, a_2, s_1, s_2).do(a_1, s_1) < do(a_2, s_2) \supset s_1 < s_2.$$

Proposition 3.2 *Unique names axioms for states are consequences of the basic axioms:*

$$S_0 \neq do(a, s),$$

$$do(a, s) = do(a', s') \supset a = a' \wedge s = s'.$$

To fully understand the meaning of the uniqueness of names axioms we must point out that two situations are the same if and only if they are the result of applying the same sequence of actions from the initial situation. Therefore, two situations S_1 and S_2 may be different and yet assign the same truth value to all the fluents. That is, they may be such that:

$$holds(F, S_1) \equiv holds(F, S_2) \tag{10}$$

In fact, the notion of situation must not be confused with the notion of *state* in *Strips*-like languages, in which a state corresponds to the set of fluents that are true in that state. In fact, some researchers make a distinction between a state and a situation, defining the state of a situation as the set of fluents that hold in that situation. In this case, two distinct situations may be in the same state. In fact, we can define a *sameState* relation for pairs of situations such that a pair of situations belong to the relation if they satisfy (10). Then, this relation can be used to write axioms that state that some sequence of actions (e.g., load, unload) leave the world in the same state (but, naturally, in a different situation). In this paper we do not need to make such a distinction.

The proofs of propositions 3.1 and 3.2 rely on the second order induction axiom (3). In order to illustrate how this induction axiom is utilized, we sketch a proof of transitivity of $<$, that is we want to prove that:

$$(\forall s_1, s_2, s_3).s_1 < s_2 \wedge s_2 < s_3 \supset s_1 < s_3.$$

To do this, we use the induction axiom with:

$$\varphi(s) = (\forall s_1, s_2).s_1 < s_2 \wedge s_2 < s \supset s_1 < s,$$

$\varphi(S_0)$ follows from (7)[3]. We then have to show that $\varphi(s) \supset \varphi(do(a,s))$ for an arbitrary a. In fact, from (9) it is trivial to see that $s < do(a,s)$ follows, taking this into account, we go on to infer that indeed $\varphi(s) \supset \varphi(do(a,s))$. Having shown that, (3) allows us to conclude $(\forall s)\varphi(s)$.

Later on, we will use the function $not : \mathcal{F} \to \mathcal{F}$, whose meaning is constrained by the following axioms:

$$not(not(f)) = f, \tag{11}$$
$$holds(not(f), s) \equiv \neg holds(f, s). \tag{12}$$

We introduce the primitive predicate $precond \subseteq \mathcal{F} \times \mathcal{A}$, which is meant to say that a property is a precondition for the successful execution of an action. Also, we define the predicate $possible \subseteq \mathcal{A} \times \mathcal{S}$ to mean that it is possible to do an action in a situation:

$$possible(a, s) \equiv (\forall f).precond(f, a) \supset holds(f, s). \tag{13}$$

To introduce change, *effect axioms* have to be specified; these have the following form:

$$possible(A, s) \wedge \phi_{A,F}(s) \supset holds(F, do(A, s)).$$

Here, $\phi_{A,F}(s)$ is a first order formula providing qualifications for the effect of the action A on fluent F. Effect axioms of this form provide sufficient conditions for literals of the form $holds(F, do(A, s))$. In order to obtain necessary and sufficient conditions we need to provide some completion mechanism (e.g., closed world assumption, predicate completion, etc.). As discussed later (section 3.2), we utilize a monotonic approach, using so-called *successor state axioms*.

So far, we have introduced axioms that allow for the specification of what is true and what truths change along different paths[4] that start in the initial situation S_0. Each one of these paths is interpreted as a different way in which the world could evolve. We incorporate a predicate *actual* for situations. The intended meaning is that a situation is *actual* if it lies on the path that describes the world's real evolution. The axioms for *actual* are:

$$actual(S_0), \tag{14}$$
$$(\forall a, s).actual(do(a, s)) \supset actual(s) \wedge possible(a, s), \tag{15}$$
$$(\forall a_1, a_2, s).actual(do(a_1, s)) \wedge actual(do(a_2, s)) \supset a_1 = a_2. \tag{16}$$

Axioms (14-15) express that the initial situation is always actual, and if a situation is actual, then its immediate predecessor must also be. Axiom (16) says that an actual situation has at most one actual successor situation. An important characteristic of actual situations is that they all lie on the same path (i.e., they constitute the time line), as the following shows:

Proposition 3.3

$$(\forall s_1, s_2).s_1 \leq s_2 \supset [actual(s_2) \supset actual(s_1)],$$
$$(\forall s_1, s_2).actual(s_1) \wedge actual(s_2) \supset s_1 < s_2 \vee s_2 < s_1 \vee s_1 = s_2.$$

We also introduce a notion of time, which will allow us to gain the same representational features as some linear temporal logics (e.g., the calculus of events [6]). Intuitively, each actual situation has a starting time. The time span of of an actual situation s extends from its starting point upto the

[3] It is easy to prove (7) by induction, simply take $\varphi(s) = S_0 \leq s$ and use axiom (4).

[4] A path is a sequence of situations that result from performing a sequence of actions starting in S_0.

starting point of its successor situation (i.e., the ending time of s) in the actual line. If an actual situation has no actual successor situation, then it spans all time points after its starting point. During the time span of a situation no fluents change truth values[5]. We incorporate the sort $\mathcal{T}$, interpreted as a continuous time line, into our language and introduce the relation $start \subseteq \mathcal{S} \times \mathcal{T}$. Events/actions are considered to occur at the ending time of situations[6]. This is captured by the following axioms[7]:

$$actual(s) \equiv (\exists\, t).start(s,t), \tag{17}$$
$$start(s,t) \wedge start(s,t') \supset t = t', \tag{18}$$
$$start(s,t) \wedge start(do(a,s),t') \supset t < t', \tag{19}$$
$$start(S_0, 0). \tag{20}$$

These have the immediate consequence:

Proposition 3.4

$$start(s,t) \wedge start(s',t') \supset [s < s' \equiv t < t'].$$

Occurrences are introduced as a relation between *event types* and situations. For example, $occurs(pickup(x), s)$ says that a pickup event occurred in situation s. Occurrences are defined in terms of the actual path as follows:

$$occurs(a,s) \equiv actual(do(a,s)). \tag{21}$$

Proposition 3.5 *The following can be easily established:*

$$actual(s) \equiv s = S_0 \vee (\exists a, s').s = do(a,s') \wedge occurs(a,s').$$

In some cases, it might be convenient to establish a relationship between events/actions that occur and the time at which they occur (rather than the situation). For this purpose, we introduce a predicate $occurs_{\mathcal{T}} \subseteq \mathcal{A} \times \mathcal{T}$, defined as:

$$occurs_{\mathcal{T}}(a,t) \equiv (\exists\, s).occurs(a,s) \wedge start(do(a,s),t). \tag{22}$$

Also, define a relation $holds_{\mathcal{T}}$ between fluents and time points and a relation *during* between time points and situations:

$$holds_{\mathcal{T}}(f,t) \equiv (\exists\, s).during(t,s) \wedge holds(f,s), \tag{23}$$
$$during(t,s) \equiv start(s,t_1) \wedge t_1 < t \wedge$$
$$[((\exists\, a, t_2).start(do(a,s),t_2) \wedge t \leq t_2) \vee (\neg(\exists\, a) actual(do(a,s)))]. \tag{24}$$

Intra-state persistence is derivable:

[5] This restricts the properties of the world that can be represented as fluents, e.g. the position of a moving ball may not be conveniently represented by a fluent. This topic is left for future research.

[6] Notice that we do not make a commitment as to whether or not actions have durations. In fact, our axiomatization is compatible with either view.

[7] In what follows, we use $<$ and $\leq$ as relations between situations (as defined earlier), as well as for the standard ordering relation between the elements of $\mathcal{T}$.

Proposition 3.6

$$during(t,s) \wedge during(t',s) \supset holds_{\mathcal{T}}(f,t) \equiv holds_{\mathcal{T}}(f,t').$$

Also, a time point belongs to exactly one situation:

Proposition 3.7

$$during(t,s) \wedge during(t,s') \supset s = s'.$$

Proposition 3.8

$$occurs(a,s) \equiv (\exists\, t).occurs_{\mathcal{T}}(a,t) \wedge start(do(a,s),t).$$

We can also introduce the notion of events occurring between situations or between times as follows:

$$occursBet(e,s_1,s_2) \equiv (\exists\, s).s_1 < s < s_2 \wedge occurs(e,s). \tag{25}$$
$$occursBet_{\mathcal{T}}(e,t_1,t_2) \equiv (\exists\, t).t_1 < t < t_2 \wedge occurs_{\mathcal{T}}(e,t). \tag{26}$$

We can infer that given two actual situations s_1 and s_2, if nothing occurs between them, then one situation must immediately follow the other:

Proposition 3.9

$$actual(s_2) \wedge s_1 < s_2 \wedge \neg(\exists\, e).occursBet(e,s_1,s_2) \supset (\exists\, a).s_2 = do(a,s_1).$$

From these we can show that:

Proposition 3.10

$$start(s,t) \wedge start(do(a,s),t') \supset \neg(\exists\, e).occursBet_{\mathcal{T}}(e,t,t').$$

and

Proposition 3.11

$$\begin{aligned} start(s,t) \equiv\ & \\ & [s = S_0 \wedge t = 0] \vee \\ & [(\exists\, a, s_p, t_p).s = do(a,s_p) \wedge occurs_{\mathcal{T}}(a,t) \wedge \\ & \quad start(s_p,t_p) \wedge \neg(\exists\, a').occursBet_{\mathcal{T}}(a',t_p,t)]. \end{aligned}$$

To facilitate the implementation described later, specifically, to eliminate a potential source of loops in the logic program corresponding to these axioms, we need:

Proposition 3.12

$$\begin{aligned} start(s,t) \equiv\ & \\ & [s = S_0 \wedge t = 0] \vee \\ & [(\exists\, a, s_p, t_p).s = do(a,s_p) \wedge occurs_{\mathcal{T}}(a,t) \wedge \\ & \quad start(s_p,t_p) \wedge \neg(\exists\, a').occursBet_{\mathcal{T}}(a',t_p,t)\ \wedge \\ & \quad t_p < t \wedge (t_p = 0 \vee (\exists\, a_p)occurs_{\mathcal{T}}(a_p,t_p))]. \end{aligned}$$

3.2 The Frame Problem.

Given two situations s_1 and s_2, with $s_1 < s_2$ we want to infer that anything true in s_1 persists to s_2, unless there is reason to believe otherwise. In the extended situation calculus, this problem has two components:

1. We need to infer persistence of fluents from one situation s to its immediate successor $do(a, s)$ based on the effects of the action a. We call this *successor state persistence.*

2. We need to assume that between two situations s_1 and s_2, no event occurs, whenever this assumption is consistent. We call this the assumption of *no intervening events.* From this assumption, it will follow that $s_2 = do(a, s_1)$ for some action a (see prop. 3.9), in which case successor state persistence will yield persistence of fluents from s_1 to s_2.

3.2.1 Successor State Persistence

To address this problem, we appeal to a monotonic solution to the frame problem which relies on *successor state axioms* [11]. In this approach a *successor state axiom* is provided for every fluent in the language. Each such axiom provides a complete characterization of a fluent's truth value in state $do(a, s)$ in terms of what is true of the state s. Syntactically, for each given fluent F, these axioms have the form:

$$possible(a, s) \supset holds(F, do(a, s)) \equiv \Phi_F(a, s),$$

where $\Phi_F(a, s)$ is a first order formula with free variables a and s, and where s is the only term of sort *situation* mentioned by Φ_F. For example, in the blocks world, the following might be a suitable successor state axiom for the fluent *holding*:

$$\begin{aligned} possible(a, s) \supset [&holds(holding(x), do(a, s)) \equiv \\ &a = pickup(x) \vee holds(holding(x), s) \wedge a \neq drop(x) \wedge \neg(\exists y) a = put(x, y)]. \end{aligned}$$

In a database setting, say for an education application, the following might be a successor state axiom for the relation $enrolled(st, c)$, meaning that student st is enrolled in course c:

$$\begin{aligned} possible(a, s) \supset [&holds(enrolled(st, c), do(a, s)) \equiv \\ &a = register(st, c) \vee holds(enrolled(st, c), s) \wedge a \neq drop(st, c)]. \end{aligned}$$

Notice that for these axioms to have their intended effects, we need unique names axioms for actions. See [11] for a discussion.

Reiter ([11]) shows how such axioms may be obtained from the effect, or "causal" axioms of the domain, and how it is that they solve the frame problem. In [12] Reiter shows how successor state axioms can be used to specify transactions in database update applications.

To summarize, our solution to the successor state persistence assumption consists of a set of successor state axioms, one for each fluent, together with a set of unique names axioms for the action terms.

3.2.2 No Intervening Events

Formalizing the assumption that no event occurs between states s_1 and s_2, unless it must occur, reduces to a suitable predicate circumscription [8] to minimize the extension of the *occurs* predicate. Our appeal to circumscription for the purposes of providing a logic program for a fragment of the extended situation calculus (Section 4) will be unproblematic, since we will be appealing to a simple completeness assumption about event occurrences. See Section 4 below for details.

3.3 Defining an Interval Based Ontology.

The version of the situation calculus we have introduced has an ontology based on time points. On the other hand, the calculus of events, as well as other prominent temporal logics (e.g. Allen's [1]), have ontologies in which the primitive temporal objects are intervals of time. In this section we show how the expressiveness of the interval based language can be realized within the situation calculus. Therefore, when axiomatizing a given domain, one has the freedom to choose whatever view of time that seems most appropriate for the application.

Before introducing predicates on intervals, we extend the language of the situation calculus with two functions *pred* and *act* which identify the unique predecessor of a situation and the action that led to a situation. From (3)-(6) it follows that:

$$s \neq S_0 \supset (\exists\, !a, s')\, s = do(a, s'). \tag{27}$$

That is, every situation, except for S_0, has a unique predecessor. Also, there is a unique action connecting a situation different from S_0 to its predecessor. Thus, we use the function $pred : \mathcal{S} \to \mathcal{S}$ to name the predecessor of a situation (we leave it undefined for S_0), and the function $act : \mathcal{S} \to \mathcal{A}$ to name the action that leads to the situation (also left undefined for S_0). Therefore:

$$s \neq S_0 \supset do(act(s), pred(s)) = s. \tag{28}$$

$$term(f, s) \equiv \neg holds(f, s) \wedge [s \neq S_0 \supset holds(f, pred(s))], \tag{29}$$
$$init(f, s) \equiv holds(f, s) \wedge [s \neq S_0 \supset \neg holds(f, pred(s))], \tag{30}$$
$$broken(f, s_1, s_2) \equiv (\exists\, s)(s_1 \leq s < s_2) \wedge \neg holds(f, s), \tag{31}$$
$$maximal(s_1, s_2, f) \equiv \tag{32}$$
$$\quad (s_1 < s_2) \wedge \neg broken(f, s_1, s_2) \wedge$$
$$\quad init(f, s_1) \wedge term(f, s_2),$$
$$incompatible(f, f') \equiv [holds(f, s) \equiv \neg holds(f', s)]. \tag{33}$$

These have the following consequences:

$$term(f, s) \wedge s \neq S_0 \supset (\exists\, s')maximal(s', s, f), \tag{34}$$
$$[term(f, s) \wedge incompatible(f, f') \wedge term(f', s') \wedge s < s'] \supset$$
$$\quad (\exists\, s_1)maximal(s_1, s', f') \wedge s < s_1, \tag{35}$$
$$[init(f, s) \wedge incompatible(f, f') \wedge init(f', s') \wedge s < s'] \supset$$
$$\quad (\exists\, s_2)maximal(s, s_2, f') \wedge s_2 < s', \tag{36}$$
$$[init(f, s) \wedge term(f', s') \wedge incompatible(f, f')] \supset$$
$$\quad (\exists\, s_1, s_2)(s < s_1 \leq s_2 < s') \wedge$$
$$\quad maximal(s, s_1, f) \wedge maximal(s_2, s', f'). \tag{37}$$

It is important to note that the predicates introduced in this subsection do not depend on the definition of *actual*. However, by integrating the notion of an *interval* with the notion of an *actual* time line we gain the same representational features claimed for the event calculus. For example,

the definition of *maximal* (axiom (32)) and theorems (35)-(37) capture the intuitions behind the cases of start and end points of intervals as discussed in [6, pp. 90-93].

Without a precise axiomatization for the calculus of events a formal argument that our language subsumes that of the calculus of events cannot be made. The alternative, which we have followed in the definitions above, is to show how, within the extended situation calculus, an interval-based ontology can be defined analogous to that of the event calculus. In other words, we have argued that the extended situation calculus is suitable for those applications where something like the event calculus provides the right ontology.

It is also worth noting that the introduction of terminations and initiations (*term*, *init* and *maximal*) provides an efficient algorithm for dealing with the frame problem in the case that complete information is at hand. Following Kowalski's proposal [5, pp.138-142], we can define an algorithm to answer queries of the form $holds(F, S)$, in which S is a completely determined situation. Such an algorithm would deal with a unique sequence of actions, which is required to be totally ordered, and the effects of each action have to be completely known. The algorithm uses a simple table of terminations and initiations. For every action that is given in the sequence, the algorithm keeps track of all the fluents that are affected by the action[8]. If a query is given with regards to a fluent F, the algorithm looks in its data structure for the last action that affected that fluent and answers whether the fluent holds based on the information encoded.

4 A Logic Programming Implementation and a Soundness Argument.

In this section we describe a logic programming implementation of a fragment of our extended situation calculus axiomatization, and show its soundness with respect to Clark's completion semantics ([2]), under suitable circumstances. The program exhibits much of the functionality of the event calculus, but in view of its soundness, is on firmer ground. Many of the assumptions we make in what follows are clearly too strong, and can be relaxed; exactly how to do this will be the subject of future research.

4.1 Problem Independent Clauses

```
holds(n(F),S):- not holds(F,S).        /* If half of axiom (12). */

holdsT(F,T):- during(T,S), holds(F,S). /* If half of axiom (23). */

actual(s_0).                    /* Part of if half of Proposition 3.5. */
actual(do(A,S)):- occurs(A,S). /* Rest of if half of Proposition 3.5. */

occurs(A,S):- occursT(A,T), start(do(A,S),T).
                        /* If half of Proposition 3.8. */

start(s_0,0).           /* Part of if half of Proposition 3.12. */
start(do(A,S),T):- occursT(A,T),           /* Rest of if half of */
                   (occursT(Ap,Ts);Ts=0),  /* Proposition  3.12. */
                   Ts<T,
                   start(S,Ts),
                   not occursBetT(E,Ts,T).

occursBetT(E,Tp,T):- occursT(E,Tpp), Tp<Tpp, Tpp<T.
```

[8] The primitive fluents or their negations.

```
                        /* If half of Axiom 26. */
during(T,S):- start(S,T1), /* Part of if half of Axiom 24. */
              start(do(A,S),T2),
              T1<T, T=<T2.
during(T,S):- start(S,T1), T1<T,     /* Rest of if half of Axiom 24. */
              not actual(do(A,S)).
```

Notice that, in the above, we use the if half of Proposition 3.12 instead of Proposition 3.11. This is to eliminate a source of non-termination in the program, which the use of Proposition 3.11 would cause.

4.2 Problem Specific Clauses

4.2.1 Initial State

We assume that the specification of the initial state consists of a sentence of the form:

$$holds(f, S_0) \equiv f = F_1 \vee \ldots \vee f = F_k.$$

In other words, we assume complete initial information about the *holds* predicate. The if half of such a specification yields the corresponding logic programming clauses:

```
holds(f1,s_0).   holds(f2,s_0). ..., holds(fk,s_0).
```

4.2.2 Event Occurrences

We assume that the specification of event occurrences consists of a sentence of the form:

$$occurs_{\mathcal{T}}(e, t) \equiv [e = E_1 \wedge t = T_1] \vee \ldots \vee [e = E_p \wedge t = T_p].$$

In other words, we assume complete information about the $occurs_{\mathcal{T}}$ predicate. The if half of such a specification yields the corresponding logic programming clauses:

```
occursT(e1,t1).  occursT(e2,t2). ..., occursT(ep,tp).
```

Notice that there are two possible ways to interpret a declaration that at time T an event E has occurred:

1. The assertion $occurs_{\mathcal{T}}(E, T)$ is an *implicit* assertion that, in addition, $possible(E, S)$ is true, where S is the state which includes time T (see axiom (15). This perspective is problematic in the logic programming setting. (See Kowalski [6] for an example.)

2. The preconditions of the event E are known to be true at time T. In this case, event preconditions are being treated as *integrity constraints*; before an assertion of the form $occurs_{\mathcal{T}}(E, T)$ can be accepted by the database, the precondition (integrity constraint) $possible(E, S)$ must be proved true, where S is the state which includes time T. If the precondition is false (or unknown), the update $occurs_{\mathcal{T}}(E, T)$ is rejected.

In this paper, we adopt the latter interpretation. This allows us to assume that all event preconditions are identically true, i.e.

$$possible(a, s) \equiv true. \tag{38}$$

Of course, this makes sense only when integrity constraint enforcement has been incorporated into the temporal database, and is invoked whenever event occurrences are declared to the database. We do not further discuss integrity maintenance in this paper.

4.2.3 Successor State Axioms

As discussed in section 3.2.1, these are sentences of the following form, one for each fluent F:

$$possible(a, s) \supset holds(F, do(a, s)) \equiv \Phi_F(a, s).$$

In view of our assumption about integrity maintenance (axiom (38)), successor state axioms will have the particularly simple form:

$$holds(F, do(a, s)) \equiv \Phi_F(a, s).$$

For example, the following is a successor state axiom for the fluent *rank* in an education database:

$$\begin{aligned} holds(rank(x, y), do(e, s)) \equiv \\ [(\exists z)e = promote(x, z, y) \vee e = hire(x, y) \vee \\ \neg[(\exists z)e = promote(x, y, z) \vee e = leave(x, y)] \wedge holds(rank(x, y), s)]]. \end{aligned}$$

This has, as its if half, the logic programming clauses:

```
holds(rank(X,Y),do(E,S)):- E = hire(X,Y) ; E = promote(X,Yp,Y).
holds(rank(X,Y),do(E,S)):- not E=promote(X,Y,Z), not E=leave(X,Y),
                           holds(rank(X,Y),S).
```

We shall assume that all logic programming clauses corresponding to successor state axioms are if halves of such axioms.

4.3 An Example

As a simple example, in figure 2 we show a very simple set of problem specific rules to implement Kowalski and Sergot's promotion example. Of course, these clauses are in addition to the problem independent clauses of Section 4.1 above.

```
holds(rank(X,Y),do(E,S)):- E = hire(X,Y) ; E = promote(X,Yp,Y).
holds(rank(X,Y),do(E,S)):- not E=promote(X,Y,Z), not E=leave(X,Y),
                           holds(rank(X,Y),S).
occursT(promote(mary,lecturer,assisProf),1).
occursT(promote(john,lecturer,assisProf),2).
occursT(leave(john,assisProf),3).
holds(rank(mary,lecturer),s_0).   holds(rank(john,lecturer),s_0).
```

Figure 2: The Promotion Example

4.4 A Soundness Argument

We are now in a position to argue the soundness of the above description of a logic programming implementation for temporal reasoning. Soundness will be with respect to the Clark completion semantics of the program.

1. **iff:** All the program clauses are if halves of corresponding iff axioms in our extended situation calculus.

2. **Equality Theory:** Unique names for states holds (Proposition 3.2), as they do for events (Section 3.2.1). We assume they hold for all other domain functions, for example, that $AssisProf \neq AssocProf$, etc. In other words, the axioms satisfy Cark's equality theory.

3. **The Frame Problem:** Our specification of a solution to the frame problem had two components (Section 3.2): successor state persistence and the assumption of no intervening events. The former is handled by successor state axioms, which we have already incorporated into our axiomatization. The latter involves minimizing event occurrences. It is easy to see that under the assumption about event occurrences of Section 4.2.2, minimizing the predicate *occurs*, leaving all other predicates fixed, does not change the theory, i.e. the models of the theory are all already minimal with respect to *occurs*. So the axioms satisfy the assumption of no intervening events. It follows that the axioms satisfy our specification of a solution to the frame problem.

The axioms satisfy all the conditions of the Clark completion semantics of the program. We conclude that the program is sound with respect to Clark's semantics.

4.5 Completeness

While we do not give it here, there is a simple inductive proof that the above logic program is complete for ground queries of the form `holds(f,t)`, where `f` is some fluent constant and `t` is a numeric constant representing a time point.

5 Conclusions and Future Research.

We have shown how the advantages of linear temporal logics can be realized within the situation calculus, a theory with a branching temporal structure. In our first order axiomatization it is not necessary that actions be totally ordered, nor that information be provided in chronological order. Furthermore, a lower bound on the computational cost is determined by the computational complexity of the task at hand. Whether or not this lower bound is realized by a given implementation is dependent upon the algorithms or reasoning framework utilized. The algorithm proposed by Kowalski for the calculus of events [5, pp.138-142] can be used in conjunction with our logical specification to efficiently address the inefficiencies associated with our solution to the frame problem.

We have also provided a logic programming implementation of our axiomatization for a very specific class of problems, namely, those for which we have a complete specification of the occurrences, along with a complete specification of the initial state. This implementation is provably sound with respect to the Clark completion semantics, and is complete with respect to ground queries of the form `holdsT(F,T)`.

This research can be extended in several ways. Our logic programming implementation can certainly be improved. In particular, the requirement of complete knowledge of events and effect of actions can be relaxed if we use a logic programming framework that allows for uncertainty (e.g. logic programming with both classical negation and negation as failure [3]).

Furthermore, our version of the situation calculus has certain representational limitations. For example, it is not possible to describe continuously changing properties as fluents (e.g., the position

of a falling object). To extend our framework to deal with these limitations, ideas of Gelfond, Lifschitz and Rabinov [4] will prove valuable.

A natural extension of this framework is the inclusion of axioms from which occurrences can be derived, as opposed to axioms in which occurrences are explicitly stated. For example, we can write a sentence stating that whenever E_1 occurs at time T, then some other event E_2 must occur at time $T + \delta$, for some fixed δ. These axioms arise in domains in which causality between events can be expressed.

Levesque, Lin and Reiter have introduced a notion of complex events, which are defined in terms of a set of operators for sequencing, non-deterministic choice, *while*-loops, etc. These complex events have been defined within the standard situation calculus, and the integration of these events in a framework in which occurrences take place has yet to be done. Such an extension will translate into a more expressive situation calculus with complex event occurrences. In turn, this will allow us to expand the classes of problems we can solve in a logic programming setting, by incorporating these complex occurrences.

Finally, a clear advantage of the situation calculus over other temporal frameworks derives from the branching structure of time. When incorporating a *time line*, we have not lost the remainder of the branching structure. The non-actual branching structure can be interpreted as alternative ways in which the world could evolve (as opposed to the way in which it *actual*ly evolved). Therefore, the temporal structure of the situation calculus can be used as the basis for hypothetical reasoning in theories in which occurrences are stated. We intend to explore this form of hypothetical reasoning in future research.

References

[1] Allen, J. F. Towards a general theory of action and time. *Artificial Intelligence 23* (1984), 123–154.

[2] Clark, K. *Negation as Failure.* Logic and Databases. Plenum Press, New York, 1978.

[3] Gelfond, M., and Lifschitz, V. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9* (1991), 365–385.

[4] Gelfond, M., Lifschitz, V., and Rabinov, A. What are the Limitations of the Situation Calculus? In *Working Notes, AAAI Spring Symposium Series. Symposium:Logical Formalization of Commonsense Reasoning.* (Mar. 1991), pp. 59–69.

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

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

[7] Lin, F., and Shoham, Y. Provably correct theories of action. Tech. rep., University of Toronto, 1992.

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

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

[10] Pinto, J., and Reiter, R. Adding a time line to the situation calculus. In *The Second Symposium on Logical Formalizations of Commonsense Reasoning* (Jan. 1993), pp. 172–177.

[11] Reiter, R. *The Frame Problem in the Situation Calculus: A Simple Solution (sometimes) and a completeness result for goal regression.* Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy. Academic Press, San Diego, CA, 1991, pp. 359–380.

[12] Reiter, R. On specifying database updates. Tech. Rep. KRR-TR-92-3, University of Toronto, July 1992.

[13] Reiter, R. Proving Properties of States in the Situation Calculus. Submitted for publication, Feb. 1992.

[14] Shanahan, M. Representing continuous change in the event calculus. In *Proceedings ECAI* (1990).

[15] Shanahan, M. Towards a calculus for temporal and qualitative reasoning. 1990.

[16] Shanahan, M. A circumscriptive calculus of events. Unpublished, 1992.