Hypothetical Reasoning From Situation Calculus to Event Calculus

Alessandro Provetti*
CIRFID - Universitá di Bologna
Via Galliera 3/a, Bologna
I-40121 ITALY
provetti@cirfid.unibo.it

Abstract

Pinto and Reiter have argued that the Situation Calculus, improved with time handling axioms, subsumes the features of *linear time* temporal formalisms such as Event Calculus and Interval Logic. In this note we find answers to some of their remarks by showing a modified version of Event Calculus that seems to match Situation Calculus handling of hypothetical reasoning and projection. Further consideration on semantics and expressive power of Event Calculus put forward by Pinto and Reiter are discussed in the light of recent proposal for an unifying semantics for languages for time and actions.

1 Introduction

In their very recent production, Reiter and Pinto[7, 8] have introduced an upgraded version of Situation Calculus (SC) which makes it possible:

- to represent dates and time-stamp actions and situations which *actually* occurred in the world;
- to represent actual situations as a branch of the tree of *possible developments* of things that Situation Calculus handles.

This new features are obtained by adding new predicate definitions and introducing a new sort of constants for representing dates, a convenient ordering, and functions such as $Start(action)$ or $End(action)$, linking actions to their dates.

Pinto and Reiter argue that the improved version matches the so-called linear time formalisms, viz. Allen's Interval Logic and the Calculus of Events(EC) of Kowalski and Sergot[3], on their own ground: representing actions and change over time.

Nonetheless, the resulting Situation Calculus maintains intact its native characteristics(set out in [5]) of dealing with alternative, hypothetical plans/sequences of actions and projecting their effects.

Another point raised by Pinto and Reiter is on semantics: they present a logic programming implementation of a subset of the formalism which enjoys a clear completion-based semantics, in contrast with EC relying on *Negation as Failure*.

In this paper it will be counter-argued that Situation Calculus specific -and indeed desirable- features are easily implementable in a linear-time formalism like Event Calculus.

In chapter 2 a simple version of EC is presented which departs from the original version criticized by Pinto et al. but can be taken as representative of current versions of EC. In chapter 3 new predicates are introduced for allowing reasoning about a fictional sequence of actions and projecting the value of fluents. This simulation can be either performed *in the future*, for exploring the result of alternative plans or starting from a date in the past, which allows for counterfactual reasoning.

In chapter 4 the declarative semantics aspect is discussed; if an EC axiomatization is seen as a logic program, then the most common declarative semantics agree, yielding what is believed a clear semantics. Indeed, a new semantics is proposed by translating EC axiomatizations to the language $\mathcal{A}$ of Gelfond and Lifschitz [1], which enjoys a semantics conceived for actions and change. A translation from a domain description EC-style to one in $\mathcal{A}$ is proposed which maps also the closed-world assumption into the target axiomatization. This technical result is kept for a full version of the paper, while it would be necessary to define a similar translation from Pinto and Reiter's formalisms to $\mathcal{A}$ itself; it will then result very interesting to compare the two axiomatizations and their models *within the same language*. This approach is specular to that of Kartha in [2] on translating $\mathcal{A}$ to chosen nonmonotonic formalisms

In the end, the author argues for a substantial equivalence of the two (improved) formalisms.

In the rest of the paper acquaintance with Situation Calculus and the semantics of Logic Programming is assumed.

*Work done during author's stay at Computer Science Department of University of Texas at El Paso, which is gratefully acknowledged.

2 The Event Calculus of the 90s

The Event Calculus has been proposed by Kowalski and Sergot [3] as a system for reasoning about time and actions in the framework of Logic Programming.

Event Calculus is based on an ontology of *events*, assumed as primitive. These events are represented by means of constants that uniquely identify them. The second ontology is that of *fluents*[1], which represents descriptions of the reality being modeled.

A fluent *holds* over time from the moment when an event *initiates* it, i.e. the event makes it true in the world. Events may also *terminate*, i.e. make false in the world, fluents. The Event Calculus is based on forward default persistence: a fluent holds over time until a terminating event is recorded.

Since the first proposal, a number of improved formalization have steamed, in order to adapt the calculus to different tasks. Hence, the reduced version of Shanahan in[11] in presented, since it can be taken as a common-core definition embedded in the latest applications[2].
Events are represented by sets of instantiations like the following:

$$\begin{array}{l} Happens(E_1) \\ Date(E_1, T_1) \\ Act(E_1, Unstack(B)) \end{array}$$

Notice that there are both *event-tokens*, labeled with the constants $E_1, E_2 \ldots$ and *events-types* named by *Unstack*, *Stack* etc. The effect of an action-type(its meaning) is understood by looking at the *Initiates/Terminates* axioms where it appears.

The definitions of *Initiates* and *Terminates* are for expressing domain knowledge. A convenient example is the Block World, as both Shanahan and Pinto et al. use it:

$$\begin{array}{lll} Initiates(e, On(x,y)) & \leftarrow & \\ & & Act(e, Move(x,y)) \end{array}$$

$$\begin{array}{lll} Initiates(e, Clear(z)) & \leftarrow & \\ & & Act(e, Move(x,y)), \\ & & Date(e,t), \\ & & HoldsAt(On(x,z),t), \\ & & z \neq y \end{array}$$

$$\begin{array}{lll} Terminates(e, Clear(y)) & \leftarrow & \\ & & Act(e, Move(x,y)) \end{array}$$

$$\begin{array}{lll} Terminates(e, On(x,y)) & \leftarrow & \\ & & Act(e, Move(x,z)), \\ & & z \neq y \end{array}$$

[1] Elsewhere called *properties* or *relationships*.

[2] This version is even more simplified, as it assumes events are recorded in the database in the same order as they happened in reality. For discussing a fuller formalization, the reader is invited to consult late works of Sergot[10] and Sripada[13].

Starting from a database of events and a domain description by *Initiates/Terminates* the axioms of EC makes it possible to derive atoms:

$$Holds(F,T)$$

which are understood as *"fluent F is true at time T"*. Axiom *ECI* means that a fluent holds at a certain time if an event happened earlier initiated the fluent itself and there is no evidence in the database of the fluent stopping to hold in the meantime. In other words, in the interval between the initiation of the fluent and the time the query is about, no terminating events must happen. This is made sure by axiom *ECII*. The forward default persistence rule is implemented by using Negation as Failure on *Clipped* in ECI.

$$\begin{array}{lll} (ECI) \quad HoldsAt(f,t) & \leftarrow & \\ & & Happens(e), \\ & & Initiates(e,f), \\ & & Date(e,t_s), \\ & & t_s < t, \\ & & not\ \ Clipped(t_s,p,t) \end{array}$$

$$\begin{array}{lll} (ECII) \quad Clipped(t_s,f,t) & \leftarrow & \\ & & Happens(e^*), \\ & & Terminates(e^*,f), \\ & & Date(e^*,t^*), \\ & & t_s < t^*, \\ & & t^* \leq t \end{array}$$

The predicates $<$ and $\leq$ establish an ordering on events. We stipulate that temporal constants $T_1, T_2, T_3 \ldots$ are mapped on naturals, and that the ordering relations are also mapped on the same relations on naturals, thus inheriting their properties.

In chapter 3, an improved version of the axioms will be presented in order to deal with hypothetic events. The hypothetic events have no time-stamping, so that the problem of integrating the linear order of actual events and the order on those hypothetical is not addressed directly.

2.1 The Assumption Underlying Event Calculus

EC is a formalism based on negation-as-failure. This device implements the implicit assumptions on the knowledge of the domain that are used by EC. Techniques are available, viz. explicit negation, for making these closure assumptions explicit. Let us list these assumptions, taking advantage of the discussions in [9, 11]:

- *It is assumed that no events occur other than those which are known to occur.*
- *It is assumed that all the events are time-stamped.*

  These two assumptions seems too strong for *real* applications such as database updates; in fact, they are lifted in enriched versions of EC.

- *It is assumed that no types of events can affect a given fluent other than those which are known to do so*

  This assumption can be made explicit by resorting to classical negation with these axioms:

  $$\neg Initiates(e,f) \quad \leftarrow \quad not \;\; Initiates(e,f)$$

  $$\neg Terminates(e,f) \quad \leftarrow \quad not \;\; Terminates(e,f)$$

  This approach is semantically founded on the Answer Sets semantics of Gelfond and Lifschitz and, for matter or generality, won't be used in the rest of the paper.

- *It is assumed that fluents persist until an event happen that influence them.*
- Conversely, *It is assumed that every fluent has an explanation in terms of events.*

That is, at least one initiating event is necessary for making a fluent true. This is particularly interesting for generating explanations of fluents by abducing events[11].

If observations on the value of fluents can be introduced in the formalization, i.e. $HoldsAt$ updates are allowed, a transformation of the axioms is necessary for giving consistent answers, at cost of a loss of elegance; Sripada[13] presents a version of the calculus for accommodating such updates.

3 Hypothetical Reasoning in EC

In this section we define new predicates (on top of those already existing) for performing projection of hypothetical sequences of actions. The purpose is that effectively illustrated by Pinto and Reiter[7]:

> By preserving the branching state property of the Situation Calculus, we can express and answer a variety of hypothetical queries, although counterfactuals cannot be expressed. For example "At time T_p in the past, when you put A on B, could A have been put on C instead?" can be simply expressed as:
>
> $$during(T_p, s) \wedge actual(s) \supset possible(put(A,C), s).$$
>
> "If I had performed $put(A,C)$, would F have been true?"
>
> $$holds(F, do(put(A,C), S_p)) \quad \bigwedge \quad possible(put(A,C), S_p).$$
>
> None of these features is possible in linear temporal logics. We need the branching structure of the situation calculus, coupled with a linear time line in that branching structure.

In the following, the new axioms and a modified and enriched version of the old ones will be illustrated, so that to deal with the sample queries proposed.

3.1 The new predicates

The ideas motivating the new predicates definition are the following:

- to rewrite situation calculus axioms within EC, in order to carry out projection;
- to provide a link between the point in time t where the simulation begins and the value of fluents in the simulation. That is, fluents that are true at t are still true during the simulation as long as an event does not terminate them. To this extent, the effect of the simulation depends from the time it starts;
- to make it possible both to project in the future and to reason hypothetically about a sequence of actions; to this extent, the effect of a simulation does not depend from the time it starts.

HypHolds

The new predicate $HypHolds$ is the counterpart of Situation Calculus $Holds$ and it is understood as follows:

a) $HypHolds(F, E\_type, T)$ is true if -has E_type been performed at time T- F would be true thereafter.

$(EC1)$
$HypHolds(f, Res(e\_type, t)) \leftarrow$
$\quad MayHappen(e\_type, t),$
$\quad Initiates(e\_type, f, t)$

$(EC2)$
$HypHolds(f, Res(e\_type, t)) \leftarrow$
$\quad MayHappen(e\_type, t),$
$\quad not \;\; Terminates(e\_type, f, t),$
$\quad HoldsAt(f, t)$

Now the predicate is defined for an arbitrary sequence of actions performed starting from T:

b) $HypHolds(F, Res(A_n, Res(\ldots, Res(A_1, T)\ldots)))$ is true if -has the sequence of actions $A_1 \ldots A_n$ been performed starting from T- then F would be true thereafter. In practice T replaces S_0, thus linking the chain of actions to the starting point of the simulation.

$(EC3)$
$HypHolds(f, Res(e\_type, s)) \leftarrow$
$\quad HypMayHappen(e\_type, s),$
$\quad HypInitiates(e\_type, f, s)$

$(EC4)$
$HypHolds(f, Res(e\_type, s)) \leftarrow$
$\quad HypMayHappen(e\_type, s),$
$\quad HypHolds(f, s)$
$\quad not \;\; HypTerminates(e\_type, f, s)$

Starting the simulation with $t = 0$, where each fluent is false (by NAF) is a way to study *in insulation* the net effect of a plan.

MayHappen

In order to ensure that an action(i.e. a type of event) can be performed at a certain time or in a certain state of affairs, the predicate $MayHappen$ and $HypMayHappen$ are introduced:

$$\begin{aligned} MayHappen(E\_type, t) \quad & \leftarrow \\ & HoldsAt(C_1, t), \\ & \ldots \\ & HoldsAt(C_n, t) \end{aligned}$$

For instance:

$$\begin{aligned} MayHappen(Move(a,b), t) \quad & \leftarrow \\ & HoldsAt(Clear(b), t) \end{aligned}$$

For each $MayHappen$ instantiation, a relative instantiation of $HypMayHappen$ is made; For instance:

$$HypMayHappen(Move(a,b), s) \leftarrow HypHolds(Clear(b), s)$$

HoldsAt and Clipped

The modifications to these predicates are not substantial, some folding operation has been carried out and the arity of $Initiates$ and $Terminates$ has been increased to accommodate the parameter time. As far as it goes, this version is expected to give the same results as Shanahan's in terms of success of $HoldsAt$ queries.

Initiates and Terminates

Also for these predicates duplication is necessary in order to handle both dates and situations. The new definition of $Initiates$ and $HypInitiates$ are like in this example:

$$\begin{aligned} Initiates(e, On(x,y), t) \quad & \leftarrow \\ & Act(e, Move(x,y)), \\ & Date(e, t) \end{aligned}$$

$$\begin{aligned} Initiates(e, Clear(z), t) \quad & \leftarrow \\ & Act(e, Move(x,y)), \\ & HoldsAt(On(x,z), t), \\ & Date(e, t), \\ & z \neq y \end{aligned}$$

$$HypInitiates(Move(x,y), On(x,y), s)$$

$$HypInitiates(Move(x,y), Clear(z), s) \leftarrow HypHolds(On(x,z), s), \; z \neq y$$

A similar transformation must be applied to the definition of $Terminates$.

3.1.1 The new predicates at work

The first question addressed by Pinto and Reiter:

> "At time T_p in the past, when you put A on B, could A have been put on C instead?"

translates into the following:

$$? - MayHappen(Put(A,C), T_p)$$

Conversely, the second example:

> "At time T_p in the past, when you put A on B, could A have been put on C instead?"

translates into:

$$? - HypHolds(On(A,C), Res(Put(A,C), T_p))$$

4 Comparing the Semantics

Pinto and Reiter[7] have compared the standard "first-order + circumscription" semantics with that of EC:

> One advantage of this is the clean semantics provided by our axiomatization, in contrasts to the event calculus reliance on the *Negation as failure* feature of logic programming, whose semantics is not well understood.

The argument is rather appropriate, EC has been natively defined within Logic Programming [3, 10, 4] and the use of *negation as failure* for implementing default persistence is somehow intrinsic to EC.

It is nonetheless the case to notice that the set of axioms described in this paper (P_{EC}) form together a stratified logic program in the sense of Apt et al.[6], under the following stratification [3]:

$$\begin{aligned} <_p = \quad & \{HoldsAt, HypHolds, MayHappen, \\ & HypMayHappen, Initiates, HypInitiates\} \\ & < \{Clipped, Terminates, HypTerminates\} < \\ & \{<, \leq\} < \\ & \{Happens, Act, Date\} \end{aligned}$$

On stratified programs the semantics common in literature hold a unique minimal model. This is the case for Przymusinki's perfects models semantics[6] by taking the partition as an ordering over predicates; the same goes for Apt et al. [6] iterated Fixpoint technique and for Gelfond and Lifschitz's Stable Models semantics. The resulting, minimal and unique model of these semantics should carry an unambiguous meaning for EC[4].

Taking $<_p$ as a circumscribing policy, the perfect model results in a model of prioritized circumscription $CIRC(P_{EC}, <_p)$ for the theory P_{EC}; it may be rewarding to compare the respective circumscriptive

[3] This stratification is in fact redundant, but fits better intuition on *layers* of predicates. To the extent of defining the declarative semantics predicates $<$ and $\leq$ can be defined as a set of ground instances on time constants.

[4] Notice in passing that Conjecture 1 of Apt *et al.* in [6] ascribes to stratified programs the completeness of SLDNF resolution.

models of two intuitively equivalent theories in EC and SC. This has not yet been carried out to author's knowledge.

4.1 Alternative Semantics

Beside the stratification-based semantics discussed above, there have been efforts to provide alternative semantics for event calculi; a first attempt is probably that of Shanahan[12], who discussed a characterization in terms of circumscription. In this section it is proposed an alternative approach by translation of Event Calculus formalizations to the language $\mathcal{A}$ of Gelfond and Lifschitz[1], which enjoys a declarative semantics purported to actions and fluents.. The translation τ transforms a set of event descriptions in terms of $Happens$, $Date$ etc. into a correspondent set of $\mathcal{A}$ axioms. The result sought after is soundness and completeness of the translation of an EC domain description D *and* of a query $? - HoldsAt(F,T)$ into an domain description $\tau(D)$ *and* a v-proposition F **after** $\mathcal{C}_D(T)$ such that:

$$D \vdash_{EC} HoldsAt(F,T) \Longleftrightarrow \tau(D) \models_{\mathcal{A}} F \textbf{ after } \mathcal{C}_D(T)$$

where the *chronicle* $\mathcal{C}_D(T)$ is the list of actions happened before T in D and ordered by means of their dates. The proof of this proposition will be included in the full version of paper.

The advantages of the translation are twofold: EC is given a new semantics and, in principle, at least a significant class of $\mathcal{A}$ axiomatizations might be effectively computed in Prolog by defining a reverse translation to EC programs. As soon as a similar translation from extended SC to $\mathcal{A}$ will become available, it will be possible to compare the two languages within the same semantical framework.

5 Conclusion

Similarities and differences between Event Calculus and Situation Calculus have been subject of much attention in the latest literature[4, 7, 8].

On the one hand, Pinto and Reiter have successfully implemented the treatment of time into SC thus matching the results obtainable with EC. This work, on the other hand, has shown an improved version of EC which performs hypothetical reasoning on the effect of actions, one of the features that motivated Situation Calculus at its birth[5].

Far this undertake from being finished, the author argues for a substantial equivalence of the two formalisms on the ground of expressive power, clear semantics and computational properties.

As for flexibility, extended versions of Event Calculus existing in the literature for dealing with compound events, temporal granularities and continuous processes are quite encouraging, as well as applications to abductive planning, deductive databases and process modeling in areas such as engineering and Law.

As for elegance, tastes probably matter. The present author feels easier at Event Calculus because of a more intuitive ontology of events and dates rater than actions, situations and dates[5], because of a plain computational value of the axiomatization and because the *closed-world based* semantics need not careful metatheoretical specifications(circumscription) to yield the expected results.

This is not to say that all the flaws of EC Pinto and Reiter point to can be easily fixed. As an instance, the aim to provide names for intervals of time bounded by events partially known has resulted in the first formalization of EC allowing unintended models, as shown in [7]. The quest for improving EC is helped by such criticisms, as long as they recognize the long way EC has gone since 1986.

Acknowledgments

My thanks to Michael Gelfond, Chitta Baral, Stefania Costantini, Gaetano Lanzarone, Paulo Azevedo and Angelo Montanari.

References

[1] Michael Gelfond and Vladimir Lifschitz. Representing Actions and Change by Logic Programs. In *The Journal of Logic Programming.*, Vol. 17(2,3,4),november 1993. pages 301-355.

[2] G. Neelakantan Kartha. Soundness and Completeness Theorems for Three Formalizations of Action. *Proc. of IJCAI'93 Conference*, 1993. pages 724–729.

[3] Robert Kowalski and Marek Sergot. A Logic-based Calculus of Events. *New Generation Computing*, volume 4 pages 67–95. Ohmsha Ltd and Springer Verlag, 1986

[4] Robert Kowalski. *Database Updates in the Event Calculus. Journal of Logic Programming*, volume 12, June 1992, pages 121–146.

[5] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[6] Jack Minker, editor. *Foundations of Deductive databases and Logic Programming.* Morgan Kaufmann Publ., 1988.

[7] Javier Pinto and Raymond Reiter. *Adding a Time Line to the Situation Calculus.* Working Papers of *Common Sense '93*, The second AAAI symposium on logical formalizations of common sense reasoning. Austin(Tx), January 1993.

[5] See [9] for a discussion on the ontologies of such formalisms

[8] Javier Pinto and Raymond Reiter. *Temporal Reasoning in Logic Programming: A Case for the Situation Calculus.* Proceedings of *ICLP'93 Conference.* Budapest, June 1993.

[9] Alessandro Provetti. *Action and Change in Logic Programming: Event Calculus, Situation Calculus and* $\mathcal{A}$. Manuscript. Spring 1993.

[10] Marek J. Sergot. (Some topics in) *Logic Programming in AI.* Lecture notes of the GULP advanced school on Logic Programming. Alghero, Italy, 1990.

[11] Murray P. Shanahan. *Prediction is Deduction but Explanation is Abduction.* Proc. of *IJCAI'89 Conference.* Detroit, 1989. pages 1055–1050.

[12] Murray P. Shanahan. *A Circumscriptive Calculus of Events.* Imperial College Dept. of Computing Technical Report.London, 1992.

[13] Sury Sripada. *Temporal Reasoning in Deductive Databases.* PhD Thesis in Computing. Imperial College, London, 1991. *A presentation of this work can be found in the Proc. of IJCAI'93, pages. 860–865.*