On the Non-monotonic Behavior of Event Calculus for Deriving Maximal Time Intervals

Iliano Cervesato, Angelo Montanari, Alessandro Provetti *

Abstract

The Event Calculus was proposed by Kowalski and Sergot as a simple and effective tool for dealing with time and actions in the framework of logic programming [9]. In response to the occurrences of events, the formalism computes maximal and convex intervals of validity of the relationships holding in the modeled world. The case of interest is when the set of events is fixed, but the order of their occurrence times is only partially known. The availability of new (pieces of) information about the relative order of events has a non-monotonic effect, making previous intervals no longer derivable. As a consequence, a meaningful ordering over partially specified event orderings may not be based on inclusion of the corresponding Computed Intervals sets. A monotonic version of the calculus is then proposed and compared to the original. We discuss why it is not immediately viable for AI applications, and show how it can be used to order partially specified orderings. A valuation function is defined that chooses among alternative orderings the one(s) which minimizes the *separation* from the result obtainable by the monotonic version.

1 Introduction

This paper analyses the non-monotonic behavior of the Event Calculus (hereinafter EC), first defined by Sergot and Kowalski [9], when applied to situations where informations about the ordering of events arrive asynchronously with respect to event occurrences recording. These considerations apply directly to fields like narrative understanding and the management of database updates, at which EC was initially aimed.

In the EC framework, the occurrences of events affect the representation of the world by starting or terminating intervals of validity for relationships on

*The second author was supported in part by the Italian National Research Council and by the Italian Department of Education (Grant MURST 60%).

entities of the world. Starting from a description of events that have occurred in the domain of interest (input data in a database context), EC is able to derive intervals where relationships hold or decide whether a relationship holds at a specific date. The computed intervals are maximal and convex and we argue that they are the most informative result that can be extracted from such a description.

Since it has been defined in the framework of logic programming, EC can efficiently derive intervals by running its axiomatic definition as a Prolog program (with slight changes). The computational cost of deriving the validity intervals for a given property has been proven to be $O(n^3)$ in the worst case, where n is the number of initiating and terminating events for that property recorded in the database [4].

When a narrative is such that precise dates are not available and events are therefore only partially ordered, new information *increasing* the ordering may have the effect of making certain previously believed intervals of validity no longer derivable. This kind of behavior seems typical of a human reasoner who is told a narrative where happenings are not described in the order as they occurred; we argue that EC is able to account for it, that is, to produce comparable results.

The paper is structured as follows: Section 2 is dedicated to a systematic presentation of EC. The underlying ideas are explained and a logic programming axiomatization for an EC system is presented. Section 3 focuses on the special case where events are supposed to be completely known but for their relative order. Section 4 analyzes the non-monotonic behavior of EC with respect to the arrival of new ordering pieces that increase (and possibly complete) the ordering information and introduces a monotonic version of the calculus. Section 5 formalizes the notion of partially specified ordering and provides some mathematical tools for evaluating and comparing the strength of different partially specified orderings. In particular, we define a valuation function which chooses among alternative, partially specified orderings the one(s) which minimizes the separation between the results obtainable by the non-monotonic and monotonic versions of EC. Finally, the achieved results are discussed and the research and application perspectives that they open are outlined.

2 The Event Calculus

The Calculus of Events was proposed by Kowalski and Sergot as a system for reasoning about actions and their effects in the world in a logic programming framework, i.e. Horn clauses augmented with negation as failure. Initially aimed at dealing with database updates and narrative understanding [10], EC has later been applied to areas such as continuous processes representation, real-time systems specification and planning [17, 5, 6]. A general description of EC and its applications to temporal deductive databases is given in [18], while a discussion on the evolution of the computational and representational methods of EC can be found in [14, 15].

EC advocates an ontology of *events*, considered more primitive than time. Occurrences of events induce a representation of the world by means of relationships among the entities of interest. They constitute in fact the boundaries of the temporal intervals of validity of the relationships: an event causes a relationship to start holding, while the happening of another event may cease it. Maximal, convex intervals of validity are called periods: given a period p for a relationship r, there is no super-interval of p where r holds continuously. Events are associated with the relationships they affect by means of relations describing *initiation*, i.e. the starting a period of validity, and *termination*, i.e. the ending of a period of validity. Although they may be intended as relations in the logical sense, relationships are represented as first-order terms. In this respect, EC is a first order theory [9].

In this paper an EC system is a database application for computing maximal time intervals for relationships using EC axioms and coherent domain specifications as a logic program. Figure 1 shows the structure of an EC system, where three main components can be distinguished: the axioms of EC, the domain description, and the database of events. The *axioms of EC* are a set of domain-independent rules that express the theory of time underlying EC. They will be described in the sections 2.2. The *domain description* consists of axioms stating which relationships are initiated and/or terminated by certain types of events. They characterize the domain of application. The *database of events* records the specific events that have happened in the world. These last two components of an EC system will be described in section 2.1.

The input of an EC system is a stream of (descriptions of) events, commonly known as the *History* of events. As soon as a new event is notified to the system, it is recorded into the database of events. Assuming that events appear in the history in a chronological order highly simplifies the formalization. This is the case in section 2.2, where the EC axioms are firstly introduced. In general, however, the order of recording may differ from the order events actually happened in the world. The general case is dealt with in section 2.2.1.

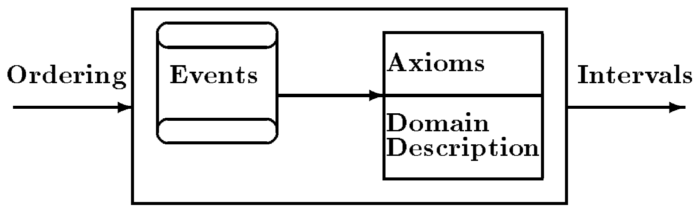

Figure 1

Albeit events are a primary representational issue, in AI the main benefits of an EC system rely in its handling of relationships and its ability to deduce maximal periods of validity. Then, we restrict our attention to the use of EC as a system for:

- deducing periods of validity of relationships, and
- checking the validity of a relationship at a certain time.

2.1 Representation of events and domain knowledge

This section describes the two lowest levels of the EC system: the events database and the domain description. Firstly, the representation of events is discussed; then, we illustrate the representation of the domain knowledge (that will be completed in Subsection 2.2); finally, we focus on the representation of time intervals.

The following example will be used through the rest of this chapter with the intent of illustrating EC. The domain we want to model is a university database which contains records of promotions, retirements and possibly demotions. We assume the following sequence of events has occurred:

(e1) Jim was appointed Professor in October 1990;
(e2) Mary was hired as a Research Assistant from August 1991;
(e3) Jim resigned from Professor in November 1991.

A possible representation of these events by means of logic programming could be[1]:

$Happens(promoted(jim, professor), oct.90)$
$Happens(promoted(mary, researchAssistant), aug.91)$
$Happens(resigned(jim, professor), nov.91)$

This representation is satisfactory as long as every event is fully specified. In order to allow dealing with non-completely specified events, EC uses a representation schema borrowed from semantic networks[9, 10]: each event is described as a set of (instances of) binary predicates; moreover, each single event is labelled with a constant. The first of the previous three events is then recorded as:

$Happens(e1)$
$Actor(e1, jim)$
$Act(e1, promotion)$
$Object(e1, professor)$
$Date(e1, 10.oct.90)$

[1]We follow the syntactic convention according to which variables and predicates symbols begin with a capital letter, while constants and function symbols begin with lowercases. Moreover, '$\leftarrow$' and ',' are to be read as *'if'* and *'and'*, respectively.

The relation *Happens* may seem unnecessary at a first glance. However, it increases readability and permits to represent and to reason about hypothetical events. Such an ability is useful in various applications, e.g., in planning [6]. This aspect will not be discussed further.

The events in the departmental database initiate and terminate periods of time during which a person has a certain rank. The fact that Jim has the rank of professor is an example of a relationship: an assertion describing (part of) the state of the world, whose validity is defined, and may change, over time.

The link between events and relationships (what we called so far domain description) is established by the definition of *Initiates* and *Terminates*; these are given as Horn clauses partially instantiated to the relationship names. In our example:

$$\begin{array}{lcl} Initiates(E, rank(Name, NewRank)) & \leftarrow & Act(E, promotion), \\ & & Actor(E, Name), \\ & & Object(E, NewRank) \end{array}$$

$$\begin{array}{lcl} Terminates(E, rank(Name, OldRank)) & \leftarrow & Act(E, demotion), \\ & & Actor(E, Name), \\ & & Object(E, OldRank) \end{array}$$

While these relations are independent from time, it is often the case that the holding of a relationship at the time of the event is a condition for the initiation/termination of another relationship. For istance, consider the statutory regulation:

A faculty shall be appointed chairman if he/she
is full professor at the time of the election.

Given $chairman$ and $fullProfessor$ as relationships, such a rule can be expressed in the following form[2]:

$$\begin{array}{lcl} Initiates(E, chairman(Name)) & \leftarrow & Act(E, election(chair)) \\ & & Actor(E, Name) \\ & & Date(E, T) \\ & & HoldsAt(rank(Name, fullProf), T) \end{array}$$

The intuitive reading of *HoldsAt(r,t)* is that the relationship r is true at time t. It will be explained in detail in the following.

A favourable way to look to *Initiates* and *Terminates* definitions is as of descriptions of the domain where we reason about happenings (events) and their effects. An appropriate set of event-descriptions and *Initiates* and *Terminates* rules will be considered accessible by the general axioms we are going to illustrate.

[2]This way to express conditionals may arise computational and semantics problems which will not be discussed here. [15] and later EC literature can serve as a guide.

Since its first formulation, EC has dealt with the necessity of giving names to periods characterizing the validity of a relationship. The first formulation of EC [9] defined the following mechanism:

$after(ei, r)$ names the period for r initiated by ei;
$before(et, r)$ names the period for r terminated by ei

Clearly, a period for r bounded by ei and et needs to be named as the conjunction of these facts.

Our approach is different in the fact that it derives only those periods which are bounded at both ends, that is, *period(ei,r,et)* represents the period where r holds, bounded by ei on the left and et on the right. Unlike Kowalski and Sergot's EC, we do not want to be able to derive, say, *Holds(after(e,r))*. In fact, *after(e,r)* may account for two different situations:

- Once initiated by e, r holds ad infinitum and we are not expecting to record terminating events in the future. This is the case for a property like $dead(jim)$.
- The terminating event has not been recorded yet, and when complete information is available the database derives a bound period for r started by the same event.

This limitation is issued at the purpose of simplifying definitions and results discussed in the rest of the paper.

2.2 The axioms of Event Calculus

The first type of queries we are interested in is concerned with the calculation of periods. It will be modelled by means of the predicate Holds. For instance, when seeking for Jim's period of professorship, the form of the query is as follows:

?- $Holds(period(StartEvent, rank(jim, professor), EndEvent))$

In case of success, the variables $StartEvent$ and $EndEvent$ will contain the boundaries of Jim's period of professorship. Note that if Jim has been professor more than once, multiple answers will be obtained. In our specific example, the expected answer is:

$StartEvent = e1$,
$EndEvent = e3$

$Holds$ is the topmost predicate of EC and it is defined as follows:

$$\begin{array}{llll}(EC1) & Holds(period(Ei,R,Et)) & \leftarrow & Happens(Ei), \\ & & & Initiates(Ei,R), \\ & & & Happens(Et), \\ & & & Terminates(Et,R), \\ & & & Before(Ei,Et), \\ & & & \text{not } Terminated(Ei,R,Et)\end{array}$$

This formulation is aimed at a simple presentation style: the conditions of *Holds* focus on events while those of *Initiates* and *Terminates* refer to relationships. This is the reason why *Happens* occurs in *Holds* instead than in *Initates* and *Terminates*. Procedurally, the interpretation of *Happens* generates candidate events to be tested against the definitions of *Initiates* and *Terminates*. As the number of events in the database grows, it is convenient to change the order and test initiation and termination first. [4] discusses the computational cost of EC in more detail.

The predicate *Terminated* is used to make sure that during the period $Ei-Et$ no other event affects R:

$$\begin{array}{llll}(EC2) & Terminated(Ei,R,Et) & \leftarrow & Happens(E), \\ & & & Before(Ei,E), \\ & & & Before(E,Et), \\ & & & Terminates(E,R)\end{array}$$

The second type of query is for checking the validity of a property at a certain (specified) date and it is dealt with by the predicate *HoldsAt*:

$$\begin{array}{llll}(EC3) & HoldsAt(R,T) & \leftarrow & Initiates(Ei,R), \\ & & & Date(Ei,Ti), \\ & & & Before(Ti,T), \\ & & & \text{not } TerminatedAt(Ti,R,T)\end{array}$$

The axiom $EC4$ for *TerminatedAt* is a simple transformation of Terminated where dates replace events (it is omitted for brevity).
A sample *HoldsAt* query is:

?- $HoldsAt(rank(jim,professor),aug.91)$

Rules $EC2$ and $EC3$, on the line developed by Sergot in [15], are formulated taking both simplicity and also procedural aspects into account. With respect to the initial formulation of Kowalski and Sergot, they appear to use a smaller number of predicates and to be more efficiently calculated.

2.2.1 Mutually exclusive relationships

Up to now, we have implicitly assumed events to be entered in the database according to their chonological order. In the most general context, however, events may be recorded in an order different from that of their happenings. When relationships which are mutually exclusive in nature are dealt with, this approach may lead to deriving inconsistent results (see [18] for a discussion of these aspects in the deductive database field).

Suppose, for instance, that the positions of research assistant and professor are incompatible, and assume that the following additional event is recorded into our sample database:

(e4) Jim was hired as a Research Assistant in August 1991

Then, the previous answer, viz. $period(e1, rank(jim, professor), e3)$, should not be derivable anymore; it would otherwise result that Jim is holding a professorship and a research assistant position at the same time.

Pictorially, the knowledge we have on the case can be represented as in figure 2, where the boxes and the lines represent events and periods, respectively.

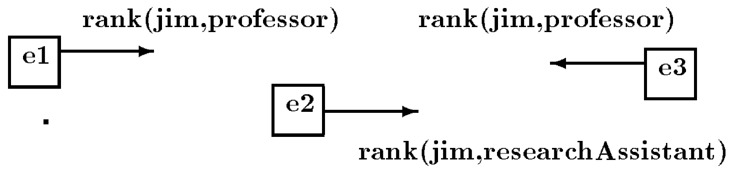

Figure 2

To handle mutually-exclusive relationships correctly, Kowalski and Sergot [9] introduce the predicate $Exclusive(r1, r2)$ as a constraint that forces the derivation of $r1$ to fail when it is possible to conclude that $r2$ holds at the same time, and vice versa. Along with *Initiates* and *Terminates*, *Exclusive* is a predicate for expressing domain knowledge into EC. In our example, the addition of the constraint $Exclusive(professor, researchAssistant)$, together with the new axiom $EC5$, blocks the derivation of incorrect periods for $rank(jim, researchAssistant)$.

Formally, the general version of EC is obtained by replacing axiom $EC1$ with axiom $EC1'$, where $EC1'$ is equal to $EC1$ but the replacement of the predicate *Terminated* with the predicate *Broken*, defined as follows[3]:

$$\begin{array}{llll}(EC5) & Broken(Ei, R, Et) & \leftarrow & Happens(E), \\ & & & Before(Ei, E), \\ & & & Before(E, Et), \\ & & & Initiates(E, R1), \\ & & & Exclusive(R, R1)\end{array}$$

$$\begin{array}{llll}(EC6) & Broken(Ei, R, Et) & \leftarrow & Happens(E), \\ & & & Before(Ei, E), \\ & & & Before(E, Et), \\ & & & Terminates(E, R1), \\ & & & Exclusive(R, R1)\end{array}$$

[3] Also *TerminatedAt* should be transformed accordingly.

The first rule for *Broken* looks for an event that initiates an incompatible relationship. The second rule checks whether there is evidence of an incompatible relationship terminated during the one under consideration. Whenever *Broken* succeeds, *Holds* fails.

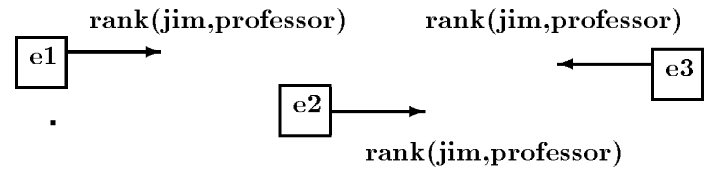

Figure 3

Exclusivity is a convenient means to constrain interferences due to incomplete sequences of events relating to the same relationship. This situation includes also the case when events are not recorded in chronological order. In our example, suppose that e4 was mistyped and that the correct update to be recorded is:

(e4′) Jim was hired as a Professor in August 1991

The answer of the previos section, viz. $period(e1, rank(jim, professor), e3)$, is again incorrect, since a still unknown event ending one period of professorship for Jim must have happened between e1 and e4′.

This problem can be easily solved by exploiting the previously introduced notion of exclusivity. It is in fact sufficient to specify that any relation is exclusive with itself (axiom $EC7$) to prevent the absence of recorded starting/ending points for a relationship to allow deriving incorrect intervals:

$(EC7)\quad Exclusive(R, R)$

3 Managing the temporal ordering of events

Even though an History may represent only partially the actual succession of happenings, the description of each event has been assumed so far complete. Several components of an event description (Subsection 2.1) may be missing: the actor, the act, the object, or the date, and even combinations. In the extreme case, we only know that a completely unspecified event has occurred. The complementary case, a fully specified event without a *Happens* statement, is used in [6] for hypothetical reasoning (see Fig. 4).

The rest of the paper will focus on the special case where the only source of incompleteness concerns the *ordering of events*: events are fully specified with the possible exception of the date they happened. An event may be recorded without a date of happening. Moreover, temporal relationships among events enter the database at any time after their occurrence. Any fragment of information concerning the temporal relationship of events will be called an *ordering piece*.

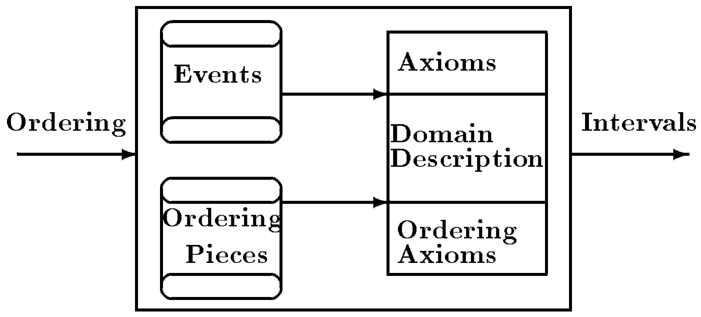

Figure 4

To simplify the description, we separate the History, i.e. the set of events recorded into the database, from the ordering of events. Furthermore, we assume the set of events to be fixed once and for all so that the only information available as input is a stream of ordering pieces. Such an assumption leads to the specialized EC system described in Figure 4. There are different types of ordering pieces, including absolute dates, relative ordering, temporal displacements. Relying on its intuitive meaning, the $Before$ predicate expressing relative ordering has been already used in the previous axioms. Subsection 3.2 will provide EC with a set of ordering axioms formally defining $Before$ in terms of ordering pieces at hand.

3.1 Types of ordering pieces

So far, the only temporal information about events consisted in $Date$ instancies accompanying their descriptions. While uncoupling $Date$ instancies from events, temporal information will be allowed to vary in format and type, that is, the already mentioned different types of *ordering pieces.*

Definition 1.
An ordering piece is an instance of the following type:

(a) dates, seen as absolute references to the time-line, e.g., *e1 occurred on October 12th*;

(b) relative pair ordering, e.g., *e1 before e2*;

(c) temporal displacement, e.g., *e1 and e2 happened at 20 units of time from each other.*

These basic temporal relationships can be used to derive several others, including:

(d) relative set ordering, e.g., *e1 before* {*e2, e3*}, which is equivalent to: *e1 before e2 and e1 before e3*;

(e) ordered temporal displacement, i.e. temporal displacement coupled with relative ordering: *e1 happened 20 units before/after e2* which is equivalent to: *e1 before/after e2 and e1 and e2 happened at 20 units of time from each other*.

It is worth noting that the ordering rests defined by the time-line. In particular, in admitting these forms of temporal displacements we commit to a specific metric of time. Albeit EC has been upgraded for dealing with changing time granularities, the results, as those in [7, 12], are not discussed here.

An examination of the axioms of EC from Section 2 shows that *Date* and *Before* only are needed for computing intervals. Therefore, the axioms of ordering, that will be given in the next section, will be tailored for deriving ordering information in the form of *Before* statements (and *Date* when possible). Notice also that in Subsection 2.2 a simple comparison between dates sufficed to implement *Before*. In the present case, the arrival of a new ordering piece may trigger the axioms of ordering and may entail, by transitivity, the validity of new (implicit) temporal relationships among events.

3.2 The axioms of ordering

The axioms of ordering will now be given for each form of ordering piece considered in the previous section. In the case of time-stamped events (a), dates can be reduced to numbers and then compared by standard means:

$$(B1) \quad Before(E1, E2) \leftarrow \begin{array}{l} Date(E1, T1), \\ Date(E2, T2), \\ TPrecedes(T1, T2) \end{array}$$

$TPrecedes(T1, T2)$ checks if the date $T1$ precedes $T2$. Notice that in Subsection 2.2 we implicity refer to this axiom defining $Before$ in terms of $Date$.

In the case of relative pair ordering (b), the calculus of relations is based on transitivity. In order to avoid loops and make *Before* effectively computable, the ordering pieces need to be instantiated with the *BeforeFact* predicate:

$$(B2) \quad Before(E1, E2) \leftarrow BeforeFact(E1, E2)$$

$$(B3) \quad Before(E1, E2) \leftarrow \begin{array}{l} BeforeFact(E1, E3), \\ Before(E3, E2) \end{array}$$

Temporal displacement (c) per se does not allow one to derive any instance of the $Before$ relation. Relative set ordering (d) requires the following additional axiom and the new predicate $BeforeSet(element, set)$:

$$(B4) \quad Before(E1, E2) \leftarrow BeforeSet(E1, S), Member(E2, S)$$

Finally, for ordered temporal displacements (e), it appears convenient to instantiate the predicate $Apart(E1, E2, n)$ to express that $E1$ happened n *time units after E2*. Clearly, if the date of one of the event is known, it is possible to calculate that of the other:

$$(B5) \quad Before(E1, E2) \leftarrow Apart(E2, E1, N)$$

$$(B6) \quad Date(E1, T1) \leftarrow Apart(E1, E2, N), Date(E2, T2), TPlus(T2, N, T1)$$

$$(B7) \quad Date(E2, T2) \leftarrow Apart(E1, E2, N), Date(E1, T1), TMinus(T1, N, T2)$$

4 The non-monotonic behavior of Event Calculus

In this chapter we discuss in detail the non-monotonic behavior of EC when dealing with incomplete information and how to possibly 'cure' this aspect of EC. As an example, when events are recorded in an order different from the order they happened in the world, the deductions made by EC at a certain point may be invalidated by the arrival of new pieces of information.

Consider the example introduced in Subsection 2.1 and suppose that only $e1$, $e2$ and $e3$ are recorded. Then, the period:

$$Holds(period(e1, rank(jim, professor), e3))$$

is derivable. However, as soon as the event $e4$ or $e4'$ is recorded into the database of events, the very same sentence is not derivable anymore. This kind of reasoning actually corresponds to the way a human would reason, assuming the same information available.

When answering such kind of queries, EC shows a non-monotonic behaviour: any time a new event is recorded, earlier results may become no longer deducible; the new event can indeed *break* the period of validity of a relationship. This behaviour is due to the use of negation as failure that allows EC to automatically withdraw conclusions that the addition of new information renders inconsistent. It is worth noting that in EC no belief revision mechanism is needed to restore consistency [9]. While a bare integrity constraints approach would refuse contrasting updates to preserve the database consistency, EC takes contrasting information not as 'wrong' or 'mistaken', but simply as incomplete. We claim that this approach is more fitted for AI applications and reasoning under uncertainty.

Restricting updates to the addition of new pieces of information about the relative ordering of events does not make EC monotonic as shown in the next section. Again, any time a new ordering piece is recorded, previous intervals may be no longer derivable. To make EC monotonic with respect to the addition of ordering pieces, EC axioms have to be revised so that they derive validity periods only when the knowledge on event ordering at hand rules out the possibility of contrasting future information. Such a monotonic EC is needed, for instance, in situations where a *don't know* answer is better than a defeasible instantiation.

4.1 Dealing with ordering pieces in Event Calculus

Restricting the dynamic component of the database to the management of ordering pieces, instead of completely-new events, allows us to focus on the effects of ordering updates in isolation.

Consider the following example. Suppose that our fixed set of events consists of *e1, e2 and e3*, all concerning a relationship r. Suppose also that the domain description is as follows:

$Initiates(e1, r)$
$Initiates(e2, r)$
$Terminates(e3, r)$

Now, consider two possible input streams. In the first case, the following ordering pieces $t1$ and $t2$ are initially given in input to the EC system:

$(t1)$ $BeforeFact(e1, e2)$
$(t2)$ $BeforeFact(e1, e3)$

EC computes the interval:

$Holds(period(e1, r, e3))$

As soon as the ordering piece $t3 = BeforeFact(e2, e3)$ appears in the ordering stream, however, the previous conclusion is no longer valid and it must be replaced with:

$Holds(period(e2, r, e3))$

So, increasing the initial information may yield a different set of computed intervals which is not a superset of the previous one, a clear manifestation of non-monotonicity.

Suppose now that the following ordering pieces *t1* and *t2* are initially given in input to the EC system:

$(t1)$ $BeforeFact(e1, e3)$
$(t2)$ $BeforeFact(e2, e3)$

EC derives the following intervals:

$Holds(period(e1, r, e3))$
$Holds(period(e2, r, e3))$

The two computed interval answers are clearly incompatible (see Subsection 2.2.1), and, in principle, the calculus could be refined to produce the answer:

$Holds(period(e1, r, e3))$ *exclusive_or* $Holds(period(e2, r, e3))$

Information at hand, however, does not suffice to discriminate between them. As soon as the ordering piece $t3 = BeforeFact(e2, e1)$ appears in the input stream, the system is obliged to retract the second of these intervals.

Notice that the previous answer is still a logical consequence of the deductive closure of the database given that:

$Holds(period(e1, r, e3)))$ & *not* $Holds(period(e2, r, e3)) \rightarrow$
$Holds(period(e1, r, e3))$ *exclusive_or* $Holds(period(e2, r, e3))$

In this second case, the behaviour of the system is monotonic, wrt the *exclusive_or* computed answers.

4.2 A monotonic version of Event Calculus

Let us now introduce a weaker variant of EC that forces the monotonicity of the calculus. It implements a sort of absolute persistence so as to exclude the possibility of deriving information that could be later retracted, provided that the event history does not change (such an assumption plays an essential role). Although its deductive power is rather limited, it will be used in later sections as a *tool for comparing partially specified orderings.* The idea is to transform the definition of *Holds* so that *Holds(period(ei,r,et))* succeeds if and only if it is possible to conclude that no event affecting *r* may have occurred after *ei* and before *et*. We call this version *Monotonic Event Calculus* (MEC) since the periods derived in this case are indefeasible wrt augmenting the ordering of events (deletion of ordering pieces is forbidden): new ordering pieces coming in may result in new periods being derived, but every old result is still available.

MEC is made up of the previous conventions and predicate definitions except for the definition of *Broken*, which turns to be the following:

$$\begin{array}{llcl}
(EC5m) & Broken(Ei, R, Et) & \leftarrow & Happens(E), \\
 & & & \text{not } Before(E, Ei), \\
 & & & \text{not } Before(Et, E), \\
 & & & Initiates(E, R1), \\
 & & & Exclusive(R, R1)
\end{array}$$

$$\begin{array}{llcl}
(EC6m) & Broken(Ei, R, Et) & \leftarrow & Happens(E), \\
 & & & \text{not } Before(E, Ei), \\
 & & & \text{not } Before(Et, E), \\
 & & & Terminates(E, R1), \\
 & & & Exclusive(R, R1)
\end{array}$$

Let us apply MEC to the example of the last section. In both the proposed cases, nothing is derivable until only $t1$ and $t2$ are known to the system. However, as soon as $t3$ becomes available, MEC is able to derive the intervals:

$Holds(period(e2, r, e3))$ and $Holds(period(e1, r, e3))$

respectively.

It is quite straightforward to prove that every interval derived by MEC is also derived by EC. This result can be formally stated as follows.

Proposition 1.
Let Hi be an history and $\prec_i$ a set of mutually consistent ordering pieces. Let $EC(MEC) \cup Hi \cup \prec_i$ the logic progam obtained by the union of the axioms contained in $EC(MEC)$, Hi and $\prec_i$.
For any pair of events ei, ej belonging to Hi and any relationship r:
if $MEC \cup Hi \cup \prec_i \vdash Holds(period(ei, r, ej))$
then $EC \cup Hi \cup \prec_i \vdash Holds(period(ei, r, ej))$.
Proof

MEC only differs from EC in the definition of the predicate $Broken$ occurring in the body of $EC1'$. Let us denote the predicate $Broken$ in EC and MEC by $BrokenEC$, defined respectively by axioms $EC5$ and $EC6$, and $BrokenMEC$, defined by axioms $EC5m$ and $EC6m$. Furthermore, let us define $ORD = \{B2, B3\} \cup \prec_i$ (notice that the fact that $\{B2, B3\}$ is a subset (subprogram) of both EC and MEC is used in the proof).

The basic steps of the proof are:

if $MEC \cup Hi \cup \prec_i \vdash Holds(period(ei, r, ej))$

then $MEC \cup Hi \cup \prec_i \vdash$ not $BrokenMEC(period(ei, r, ej))$

then $MEC \cup Hi \cup \prec_i \not\vdash BrokenMEC(period(ei, r, ej))$

then for each event e and each relationship $r1$ such that:
$Happens(e), Initiates(e, r)$ (or $Terminates(e, r)$) and $Exclusive(r, r1)$ succeed,
either $ORD \not\vdash$ not $Before(e, ei)$ or $ORD \not\vdash$ not $Before(et, e)$.

then either $ORD \vdash Before(e, ei)$ or $ORD \vdash Before(et, e)$

then either $ORD \not\vdash Before(ei, e)$ or $ORD \not\vdash Before(e, et)$,
given that the ordering is antisymmetric (antisymmetry can be expressed by means of the integrity constraint: $\leftarrow Before(X, Y), Before(Y, X)$)

then $EC \cup Hi \cup \prec_i \not\vdash BrokenEC(period(ei, r, ej))$

then $EC \cup Hi \cup \prec_i \vdash$ not $BrokenEC(period(ei, r, ej))$

then $EC \cup Hi \cup \prec_i \vdash Holds(period(ei, r, ej))$.
Q.E.D.

It is easy to see that the opposite implication does not hold in general.

5 Relating orderings to answer sets

In this Section we shall apply the definitions given in the previous sections for studying the effect of ordering updates. The initial idea is to relate partially-specified orderings to the sets of intervals computed by EC given the corresponding ordering pieces as input. The goal is to devise a mechanism which can choose (among two or more partially specified, alternative orderings) the one which better 'fits' the constraints in the domain description (including the exclusive instancies).

5.1 Partially specified orderings

In this section we focus on finite histories of lenght n events and such that, for each pair of distinct events ei and ej, the date of ei differs from the date of ej. Under such an assumption, the total orderings over Hi are $n!$.

We look at the input of ordering pieces as at the process of putting new constraints that *narrow* the specification of the actual event ordering and, then, reduce the number of admissible orderings that can be consistently foreseen. Let us call each of these admissible orderings an *extension* of the given partial specification. As the ordering specification becomes complete, the "surviving extensions" reduce to just one.

Now the notion of *partially specified ordering* has to be formalized. First of all, given a finite set E of n events, a total ordering $\prec$ over E is a antireflexive, antisymmetric and transitive binary relation such that:

$$\forall x, y \in E,\ \ x \neq y \rightarrow x \prec y \ or \ y \prec x$$

It univocally identifies a subset of $E \times E$. Further, let $Ord(E)$ be the set of all possible total orderings over E:

$$Ord(E) = \{\prec_i \mid \prec_i \subset E \times E \ and \ \prec_i \ is \ total \ and \ \forall x, y, z \in E \ \neg(x \prec_i x), \\ x \prec y \rightarrow \neg(y \prec x), x \prec y \ and \ y \prec z \rightarrow x \prec z\}$$

It consists of $n!$ elements, each one consisting of $n \cdot (n-1)/2$ mutually consistent ordering pieces. Each element of $Ord(E)$ is then completely specified by $n \cdot (n-1)/2$ ordering pieces. On the contrary, any proper subset of ordering pieces provides a partial specification of the actual ordering only and it univocally identifies the subset of $Ord(E)$ consisting of all its admissible extensions.

On the basis of the correspondence between ordering pieces and $Before$ statements, we can equivalently define as *partial specification* the transitive closure of a set of $Before$ statements. The input process can then be seen as a sequence of partial specifications $\prec_0 = \{\}, \prec_1, \prec_2, \prec_3, \ldots, \prec_m, \ldots$, such that:

$$\prec_{i+1} = (\prec_i \cup \{Before(ej, ek)\})^* \text{ and } \forall i \prec_i \text{ is consistent}$$

where $()^*$ denotes the transitive closure and $\prec_i$ is consistent if and only if the ordering pieces belonging to it are mutually consistent.

Of course, for each i, the set of admissible extensions of $\prec_{i+1}$ is a subset of those of $\prec_i$. Moreover, it is easy to prove that, for all i, the admissible extensions are less than or equal to $\lceil n!/2^i \rceil$. Finally, it is worth noting that in the worst case

$$(\prec_i \cup \{Before(ei, ej)\})^* = \prec_i \cup \{Before(ei, ej)\}$$

This is the case in the following example. Let $Hi = \{e1, e2, e3\}$. In the case of the input stream:

$$\begin{aligned}
&\prec_1 = (\prec_0 \cup \{(e1, e3)\})^* = \{(e1, e3)\} \\
&\prec_2 = (\prec_1 \cup \{(e1, e2)\})^* = \prec_1 \cup \{(e1, e2)\} = \{(e1, e3), (e1, e2)\} \\
&\prec_3 = (\prec_2 \cup \{(e2, e3)\})^* = \prec_2 \cup \{(e2, e3)\} = \{(e1, e3), (e1, e2), (e2, e3)\}
\end{aligned}$$

the number of inputs required to completely specify a total ordering is equal to the number of pieces of information which univocally characterize it, i.e. $n \cdot (n-1)/2$.

5.2 An ordering over partially specified orderings

In this section, we outline a way for studying formally the relations between partially specified orderings on events $\prec_i$ and *interval answer sets*, i.e. the sets of intervals that EC and MEC derive from a given $\prec_i$.

Let H be the deductive closure of the set of event descriptions Hi and of the domain-dependent predicates present in the system. The sets $Ans(H, \prec_i)$ and $Ans_m(H, \prec_i)$ of intervals derivable respectively from EC and MEC are defined as follows:

$$\begin{aligned}
&\forall \prec_i, Ans(H, \prec_i) = \{period(ei, r, ej)/EC, H, \prec_i \vdash Holds(period(ei, r, ej))\} \\
&\forall \prec_i, Ans_m(H, \prec_i) = \{period(ei, r, ej)/MEC, H, \prec_i \vdash Holds(period(ei, r, ej))\}
\end{aligned}$$

¿From Proposition 1, it follows that, for any history H and any ordering $\prec_i$, the following holds:

$$Ans_m(H, \prec_i) \subseteq Ans(H, \prec_i)$$

It is easy to prove that $Ans(H, \prec_i)$ and $Ans_m(H, \prec_i)$ coincide when $\prec_i$ is the empty set of ordering pieces as well as when it is a total ordering.

5.3 Results from Logic Programming

The sets $Ans(H, <_i)$ and $Ans_m(H, <_i)$ above prove to be well-defined with respect to the declarative semantics of the following logic programs:

$$P_{EC} = \{EC1', EC5, EC6, EC7\} \cup \{B1, B2, B3, B4, B5, B6, B7\} \cup \{Initiates, Terminates, Exclusive\}$$

$$P_{MEC} = \{EC1', EC5m, EC6m, EC7\} \cup \{B1, B2, B3, B4, B5, B6, B7\} \cup \{Initiates, Terminates, Exclusive\}$$

where $\{Initiates, Terminates, Exclusive\}$ denotes a given set of axioms expressing domain knowledge.

In fact, it is possible to prove that P_{EC} is a stratified logic program (no recursion through negation) [1]. The following results also holds for P_{MEC} with minor modifications in the stratification policy.

Proposition 2.
P_{EC} is stratified, in particular with respect to the following partition:

$$P_{EC} = \{EC1'\} \cup \{EC5, EC6\} \cup \{Initiates, Terminates, B2, B3\} \cup \{Happens, EC7, Act, Actor, Date, BeforeFact\}$$

Proof:
It follows directly from Definition 3 of [1].
Q.E.D.

Proposition 3.
P_{EC} has a unique minimal model.
Proof
Taking the partition of P_{EC} as an ordering over predicates, i.e. as an ordering $<_p$ such that $Holds <_p Broken <_p ...$, it satisfies definition 4 (stratification) of [13]. Hence, P_{EC} has a a unique perfect model which is minimal as defined by Przymusinski (definition 2) [13].
Q.E.D.

Notice that this model coincides with the Iterated Fixpoint of [1] (theorem 4) and that the perfect model of P_{EC} is also a model of Prioritized Circumscription $CIRC(P_{EC}, <_p)$. Furthermore, other semantics defined for broader classes of logic programs, viz. Stable Models, agree in the case of stratified programs. Finally, as for the proof-theory, we stress that subsequent results, like Conjecture 1 of [1], ascribe to P_{EC} the property of completeness of SLDNF resolution.

Unfortunately, stratification may disappear when $HoldsAt$ atoms are included in the body of $Terminates$ and $Initiates$ definitions. In [16] Shanahan sketches how to get round this problem by proving a weaker property of Local Stratification [13].

As long as the $HoldsAt$ predicate is excluded, however, it is possible to guarantee that $Ans(H, <_i)$ and $Ans_m(H, <_i)$ are actually computable. Transitive closure of the *Before* predicate over any finite history can indeed be proved to be finite. Then, it is possible to remove *B2* and *B3* from P_{EC} as well as from P_{MEC} and to replace them by the finite set of facts constituting the transitive closure of *Before*, i.e., transitive closure can be pre-processed and then added to P_{EC} and P_{MEC}. The resulting versions of P_{EC} and P_{MEC} are hierarchical and then terminating. The thesis immediately follows from the boundedness of *Holds* goals [3].

5.3.1 Termination

The valutation of ground queries to P_{EC} with the usual Prolog interpretation rule is always terminating.

Proposition 4.
P_{EC} is (left) terminating.

The proof, given in Appendix A, is obtained by applying the compositional methodology defined in [3] that combines termination proofs of separate programs to obtain proofs of larger programs.

In this case P_{EC} is proven terminating by separately proving that $P_B = \{B2, B3, BeforeFact\}$ and $P_{EC} \setminus P_B$ are terminating.

5.4 Defining the persistence Degree

Relying on the formalization that has been introduced so far, the results on non-monotonicity and monotonicity of P_{EC} and P_{MEC} are summarized by the following relations:

$$\prec_i \subseteq \prec_j \not\rightarrow Ans(H, \prec_i) \subseteq Ans(H, \prec_j)$$
$$\prec_i \subseteq \prec_j \rightarrow Ans_m(H, \prec_i) \subseteq Ans_m(H, \prec_j)$$

The non-monotonicity of P_{EC} makes absolute persistence of computed intervals unviable. Nevertheless, it is possible to associate a *persistence degree* with the set of intervals computed from a history Hi and an ordering $\prec_i$ that gives us an idea of the number of intervals that will remain valid independently of the arrival of new ordering pieces.

A plausible definition of persistence degree is:

$$||Ans(H, \prec_i)|| = (|Ans_m(H, \prec_i)| + 1)/(|Ans(H, \prec_i)| + 1)$$

where $|Ans_m(H, \prec_i)|$ and $|Ans(H, \prec_i)|$ denote the number of validity intervals computed by P_{EC} and P_{MEC}, respectively. This definition is devised to make $||Ans(H, \prec_i)||$ equal to 1 if and only if $Ans(H, \prec_i)$ and $Ans_m(H, <_i)$ coincide; otherwise, it is a rational number in the open interval $(0, 1)$. Clearly, when $\prec_i$ is a total ordering on H its measure is equal to 1.

When trying to compare the inferential strength of different partially specified orderings, a measure based on the persistence degree is ar least more general than the obvious criteria relying on partial specification inclusion (which is not always applicable), and more appropriate than comparing derivable interval sets by means of set inclusion.

An example should illustrate the concept. Suppose we have the history Hi:

$$Hi = \{Happens(e1), Happens(e2), Happens(e3),$$
$$Happens(e4), Happens(e5), Happens(e6)\}$$

where the actual ordering is such that, for $i = 1, ..5$, $Before(ei, e(i+1))$. Moreover, assume the following domain description:

$Initiates(e1, r), Terminates(e2, r),$
$Initiates(e3, p), Terminates(e4, p),$
$Initiates(e5, r), Terminates(e6, r), Exclusive(r, p)$

The result is pictorially represented in Figure 5:

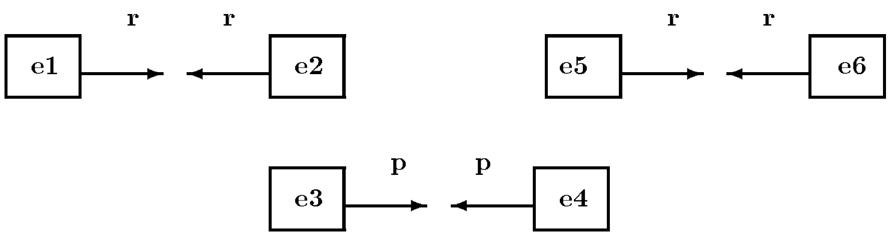

Figure 5

Consider now the sequence of partial ordering specifications:

$$\Sigma_0 = \{\}, \Sigma_1 = \Sigma_0 \cup \sigma_1, \ldots$$

where σs are ordering pieces. In the following example, based on a random sequence of σs, Ans, Ans_m and the persistence degree evolve as follows(B will stay for $Before$):

$\sigma_1 = \{B(e1, e4)\}$	$\vert Ans(H, \Sigma_1)\vert = 0$	$\vert Ans_m(H, \Sigma_1)\vert = 0$	$\vert\vert Ans(H, \Sigma_1)\vert\vert = 1$
$\sigma_2 = \{B(e1, e6)\}$	$\vert Ans(H, \Sigma_2)\vert = 1$	$\vert Ans_m(H, \Sigma_2)\vert = 0$	$\vert\vert Ans(H, \Sigma_2)\vert\vert = 0.5$
$\sigma_3 = \{B(e2, e4)\}$	$\vert Ans(H, \Sigma_3)\vert = 1$	$\vert Ans_m(H, \Sigma_3)\vert = 0$	$\vert\vert Ans(H, \Sigma_3)\vert\vert = 0.5$
$\sigma_4 = \{B(e1, e2)\}$	$\vert Ans(H, \Sigma_4)\vert = 2$	$\vert Ans_m(H, \Sigma_4)\vert = 0$	$\vert\vert Ans(H, \Sigma_4)\vert\vert = 0.33$
$\sigma_5 = \{B(e3, e4)\}$	$\vert Ans(H, \Sigma_5)\vert = 3$	$\vert Ans_m(H, \Sigma_5)\vert = 0$	$\vert\vert Ans(H, \Sigma_5)\vert\vert = 0.25$
$\sigma_6 = \{B(e4, e5)\}$	$\vert Ans(H, \Sigma_6)\vert = 3$	$\vert Ans_m(H, \Sigma_6)\vert = 0$	$\vert\vert Ans(H, \Sigma_6)\vert\vert = 0.25$
$\sigma_7 = \{B(e2, e3)\}$	$\vert Ans(H, \Sigma_7)\vert = 3$	$\vert Ans_m(H, \Sigma_7)\vert = 0$	$\vert\vert Ans(H, \Sigma_7)\vert\vert = 0.25$
$\sigma_8 = \{B(e2, e6)\}$	$\vert Ans(H, \Sigma_8)\vert = 3$	$\vert Ans_m(H, \Sigma_8)\vert = 1$	$\vert\vert Ans(H, \Sigma_8)\vert\vert = 0.5$
$\sigma_9 = \{B(e5, e6)\}$	$\vert Ans(H, \Sigma_9)\vert = 3$	$\vert Ans_m(H, \Sigma_9)\vert = 3$	$\vert\vert Ans(H, \Sigma_9)\vert\vert = 1$

5.5 Choosing between alternative orderings

Let us show now how the concept of persistence degree of computed interval sets can be actually used to discriminate between alternative, partially specified orderings. Suppose we have the simple history Hi:

$Hi = \{Happens(e1), Happens(e2), Happens(e3)\}$

Moreover, assume the following domain description:

$Initiates(e1, r), Initiates(e2, r), Terminates(e3, r)$

Let $\prec_1$ be equal to $\{Before(e1, e3)\}$. The interval sets $Ans(H, \prec_1)$ and $Ans_m(H, \prec_1)$ are equal to $period(e1, r, e3)$ and to the empty set, respectively. The degree of persistence $||Ans(H, \prec_1)||$ is then equal to 0.5.

In the case there are contrasting pieces of information about, for instance, the relative ordering of $e1$ and $e2$ and there are no reason to prefer one piece of information to the other (i.e. sources of contrasting information are equally reliable), the persistence degree of the corresponding sets of computed intervals can be used to choose between the two alternatives:

1. If we assume that $BeforeFact(e1, e2)$ holds, then:

$$\prec_{2a} = \{Before(e1, e3), Before(e1, e2)\}$$

   In such a case, the interval sets $Ans(H, \prec_{2a})$ and $Ans_m(H, \prec_{2a})$ are equal to $Ans(H, \prec_1)$ and $Ans_m(H, \prec_1)$, respectively. As a consequence, the degree of persistence also remains unchanged:

$$||Ans(H, \prec_{2a})|| = ||Ans(H, \prec_1)|| = 0.5$$

2. Differently, if $BeforeFact(e2, e1)$ is assumed, then:

$$\prec_{2b} = \{Before(e1, e3), Before(e2, e1), Before(e2, e3)\}$$

   The transitive closure of the $Before$ relationship univocally determines the total ordering of events. We know that in such a case $Ans(H, \prec_{2b})$ and $Ans_m(H, \prec_{2b})$ coincide (both of them are equal to $\{period(e1, r, e3)\}$). Therefore:

$$||Ans(H, <_{2b})|| \ = \ 1$$

While in the first case the persistence of $\{period(e1, r, e3)\}$ depends on the relative ordering of $e2$ and $e3$, in the second case it is definitively valid. The different degree of persistence of the answer sets leads us to prefer the second configuration of events to the first one. It is worth noting that the EC computed answer sets by themselves are not able to discriminate between the two configurations, given that $Ans(H, \prec_{2a}) = Ans(H, <_{2b})$.

It is quite straightforward to generalize the proposed example with respect to both the number of events and the number of contrasting pieces of information about their ordering. It should be clear, however, that we are not simply claiming that a total order is better than a partial one with respect to the inferential strenght (it is true, but obvious). We dealt with the problem to guess the correct order of a set of events, or, at least, to find out a partial order that is closed as much as possible to it, when only incomplete information is available and recognized that contrasting ordering pieces are not equally informative. The notion of persistence degree is then introduced as a formal tool to identify and select more informative ordering pieces.

6 Conclusions

EC is a well-suited formalism for computing maximal and convex validity intervals for relationships represented in databases that are updated by recording further events (database expansion by append-only updates [8]). Such an ability makes EC a candidate for supporting AI applications involving temporal intervals calculation with time-line adjustments[9].

The paper has shown that the interesting case of a partially known order of events happenings, dealt with by the original EC, has a non-monotonic behavior with respect to the availability of new pieces of information about the ordering of events. In fact, when precise dates are not available and events are only partially ordered, new pieces of information further specifying temporal ordering may have the effect of making certain previously believed intervals of validity no longer assumable.

The paper focuses on updates of an EC-style database that increase the knowledge about event ordering and analyzes their non-monotonic effects. It formally relates partially-specified orderings to the corresponding sets of computed intervals Ans. The inadequacy of an ordering over partially specified orderings based on inclusion of the Ans sets was discussed and a taxonomy based on the 'distance' between Ans and Ans_m was defined to overcome this limitation. To measure such a distance the notion of persistence degree of an ordering is introduced. Finally, an example of how to use persistence degrees to choose among alternative, partially specified orderings is provided.

It is not claimed that the definition of persistence degree that has been proposed is always the better. According to the application domain, the answer of the monotonic and of the non-monotonic versions may indeed acquire a different relevance. Rather than trying to provide a 'universal' definition of persistence degree, the paper aimed at contributing to a better understanding of the possibilities of using EC for calculating temporal relations and to push further its range of application by introducing the notion of persistence degree. Along this direction, future work will be devoted to extend the achieved results and to take into account different types of ordering pieces.

Acknowledgements

The authors thank Vladik Kreinovich for having first pointed to EC as a viable system for interval calculation. Johan van Benthem provided useful discussions on the properties of partially-ordered sets of events. Paulo Azevedo and Alberto Paccanaro kindly proof-read and commented early versions of this paper.

Appendix A. Proof of Proposition 4

In order to understand the proof, some preliminary notions must be briefly introduced. A detailed definition of them can be found in [3].

First of all, we need the notion of *LD-derivation*, which only differs from the standard notion of *SLD-derivation* in the use of the Prolog first-left selection rule. A program is said *left terminating* if all its LD-derivations starting with a ground goal are finite. Secondly, we need the notion of *level mapping* $|\ |$, which is a function from program atoms to natural numbers. A program is said *acceptable* if it is acceptable with respect to some level mapping $|\ |$ and a model I of it, and it is acceptable wrt $|\ |$ and I if all its clauses are. A clause of the program is acceptable wrt $|\ |$ and I if I is a model of it and, for every ground instance $A \leftarrow A_1, ..., A_{i-1}, B, A_{i+1}, ..., A_n$ of it such that $I \models A_1, ..., A_{i-1}$, $|A| > |B|$. A program is left terminating if and only if it is acceptable (Corollary 3.11 of [3]).

To deal with modular programs the notion of acceptability must be replaced by the notion of *semi-acceptability*. Let *rel(A)* the relation symbol occurring in atom A. For any pair of atoms A and B of a given program, we say that *rel(A) refers to rel(B)* if there is a clause with A in its head and B in its body. Furthermore, we say that *rel(A) depends on rel(B)* ($rel(A) \sqsupseteq rel(B)$) if the pair *(rel(A),rel(B))* is in the reflexive and transitive closure of the relation *refers to*. On the basis of $\sqsupseteq$, we can define two meaningful relations, viz.

$$rel(A) \simeq rel(B) \equiv (rel(A) \sqsupseteq rel(B) \wedge rel(B) \sqsupseteq rel(A))$$

$$rel(A) \sqsupset rel(B) \equiv (rel(A) \sqsupseteq rel(B) \wedge rel(B) \not\sqsupseteq rel(A))$$

The definition of semi-acceptable program is equal to the one of acceptable program but the replacement of the condition $|A| > |B|$ with

(i) $|A| > |B|$ if $rel(A) \simeq rel(B)$;

(ii) $|A| \geq |B|$ if $rel(A) \sqsupset rel(B)$

[3] proved that a program is acceptable if and only it is semi-acceptable (Corollary 5.5).

In case of *terminating* programs, i.e. programs such that all their SLD-derivations starting with a ground goal are finite, the notions of acceptable and semi-acceptable programs can be replaced by the simpler notions of *recurrent* and *semi-recurrent* programs. Such notions can be obtained from the notions of acceptability and semi-acceptability by removing the model parameter [3].

Finally, given two programs P and Q, we say that a relation *rel(A)* is *defined in P* if *rel(A)* occurs in the head of a clause from P and that P *extends* Q if no relation defined in P occurs in Q.

[3] proved that if P and Q are two programs such that P extends Q and there exists a model I of $P \cup Q$ such that:

(i) Q is semi-acceptable wrt a level mapping $| \, |_Q$ and $I \cap B_Q$, where B_Q is the Herbrand Base of Q;

(ii) P is semi-acceptable wrt a level mapping $| \, |_P$ and I;

(iii) there exists a level mapping $|| \, ||_P$ such that for every ground instance $A \leftarrow A_1, .., A_{i-1}, B, A_{i+1}, .., A_n$ of a clause from P such that $I \models A_1, .., A_{i-1}$

(a) $||A||_P \geq ||B||_P$ if *rel(B)* is defined in P;

(b) $||A||_P \geq |B|_Q$ if *rel(B)* is defined in Q

then $P \cup Q$ is semi-acceptable wrt $| \, |$ and I, where $| \, |$ is defined as follows:

1. if *rel(A)* is defined in P, then $|A| = |A|_P + ||A||_P$;
2. if *rel(A)* is defined in Q, then $|A| = |A|_Q$

Let us now prove *Proposition 4*. First of all, let P and Q be equal to $P_{EC} \setminus \{B2, B3, BeforeFact\}$ and $\{B2, B3, BeforeFact\}$, respectively.

It is immediate to prove that P is hierarchical wrt the partition of *Proposition 2*, provided that $B2, B3$ have been removed and *BeforeFact* have been replaced by *Before*. A hierarchical program P is recurrent wrt the level mapping $| \, |_P$ that associates the index of the stratum within which *rel(A)* is defined with each atom A such that *rel(A)* is defined in P (indices of strata are formally defined in [2]) and 0 with each atom A such that *rel(A)* is not defined in P (this is the case of *Before*, which is defined in Q).

[3] proved that if P is recurrent wrt $| \, |_P$, then it is semi-recurrenct wrt $| \, |_P$ (Lemma 4.3), and then P is semi-recurrent wrt $| \, |_P$ if and only if it is semi-acceptable wrt $| \, |_P$ and B_P (Lemma 5.2).

Les us introduce the interpretation $I_{P \cup Q} = B_P \cup$ *BeforeFact* of P. It is immediate to prove that $I_{P \cup Q}$ is a model of P (the relation *BeforeFact* does not occur in P) and that P is semi-acceptable wrt $| \, |_P$ and $I_{P \cup Q}$. Now, we are able to prove now that Q is acceptable wrt a level mapping $| \, |_Q$ and a model I_Q such that

$$I_Q = I_{P \cup Q} \cap B_Q$$

that is, I_Q includes $[Before(X, Y)]$ (the set of all ground instances of the atom $Before(X, Y)$) and all and only the ground instances of the atom *BeforeFact(X, Y)* recorded in the database (the set $\{BeforeFact\}$), and $| \, |_Q$ is defined as follows.

Let E and $TotOrd(E)$ be the set of events $\{e1, .., en\}$, with $n > 1$, and a total ordering over E extending $\{BeforeFact\}$, respectively. For the sake of simplicity, suppose also that, for $i = 1, .., n$, $Before(ei, e(i+1)) \in TotOrd(E)$. We define $| \, |_Q$ so that:

(i) $|Before(ei, ej)|_Q = j - i + 1 + n$ if $Before(ei, ej) \in TotOrd(E)$;

(ii) $|Before(ei, ej)|_Q = (1/(i - j + 1)) + n$ otherwise

$| \, |_Q$ is equal to the number of events occurring between ei and ej (ei and ej included) plus n (total number of events) in case (i) and the inverse of the number of events occurring between ej and ei (ej and ei included) plus n in case (ii). It is immediate to prove that:

if $|Before(ei, ek)|_Q = k - i + 1 + n$ and $|Before(ek, ej)|_Q = j - k + 1 + n$, then $|Before(ei, ej)|_Q = |Before(ei, ek)|_Q + |Before(ek, ej)|_Q - (n+1) = m + 1 + n$ with $m > 2$.

This simple lemma will be used in the rest of the proof.

(iii) $|BeforeFact(ei, ej)|_Q = j - i + 1$ if $Before(ei, ej) \in TotOrd(E)$;

(iv) $|BeforeFact(ei, ej)|_Q = 1/(i - j + 1)$ otherwise.

Let us prove now that each clause of Q is acceptable wrt $| \, |_Q$ and I_Q:

1. $\{BeforeFact\}$: immediate, given that they have an empty body;
2. $B2$: immediate, given that, for all $ei, ej \in E$, $|Before(ei, ej)|_Q = |BeforeFact(ei, ej)|_Q + n$, with $n > 1$;
3. $B3$: for all $ei, ej, ek \in E$, we must consider separately the first and the second atom in the body.

   With respect to the first atom,

   $|Before(ei, ej)|_Q > |BeforeFact(ei, ek)|_Q$

   immediately follows from

   $|BeforeFact(ei, ek)|_Q = m \leq n < |Before(ei, ej)|_Q = m' + n$
   with $m' > 0$.

   With respect to second atom, there are two cases:

   3.1 $I_Q \not\models BeforeFact(ei, ek)$, i.e. $BeforeFact(ei, ek) \notin \{BeforeFact\}$: immediate;

3.2 $I_Q \models BeforeFact(ei, ek)$, i.e. $BeforeFact(ei, ek) \in \{BeforeFact\}$: three different cases must be considered:

3.2.1 $Before(ei, ej) \in TotOrd(E)$ and
$Before(ek, ej) \in TotOrd(E)$: the proof is by contradiction.
Let $|Before(ei, ej)|_Q = m + n$, with $m > 1$ and suppose that $|Before(ek, ej)|_Q = m' + n$, with $m' \geq m$. Given that $TotOrd(E)$ is an extension of $\{BeforeFact\}$, $Before(ei, ek) \in TotOrd(E)$ and
$|Before(ei, ek)|_Q = m'' + n$, with $m'' > 1$. From the previous lemma, it follows that:
$|Before(ei, ej)|_Q =$
$|Before(ei, ek)|_Q + |Before(ek, ej)|_Q - (n + 1) =$
$= (m''+n)+(m'+n)-(n+1) > (1+n)+(m'+n)-(n+1) \geq m+n$
(contradiction).

3.2.2 $Before(ei, ej) \in TotOrd(E)$ and
$Before(ek, ej) \notin TotOrd(E)$: immediate, given that from $|Before(ei, ej)|_Q = m + n$ with $m > 1$ and $|Before(ek, ej)|_Q = m' + n$, with $m' \leq 1$,
it follows that $|Before(ei, ej)|_Q > |Before(ek, ej)|_Q$

3.2.3 $Before(ei, ej) \notin TotOrd(E)$ and
$Before(ek, ej) \notin TotOrd(E)$: the proof is by contradiction.
Let $|Before(ei, ej)|_Q = m+n$, with $m = 1/(i-j+1)$ and suppose that $|Before(ek, ej)|_Q = m' + n$, with $m' = 1/(k - j + 1)$ and $m' \geq m$. It is immediate to prove that $i - k \geq 0$. On the other hand, given that $TotOrd(E)$ is an extension of $\{BeforeFact\}$, $Before(ei, ek) \in TotOrd(E)$ and $|Before(ei, ek)|_Q = m'' + n$, with $m'' = k - i + 1$ and $m'' > 1$. It is immediate to prove that $i - k < 0$ (contradiction).

Notice that the fourth case $Before(ei, ej) \notin TotOrd(E)$ and $Before(ek, ej) \in TotOrd(E)$ is inherently contradictory. From $BeforeFact(ei, ek)$ we can derive that $Before(ei, ek) \in TotOrd(E)$; then, from $BeforeFact(ei, ek) \in TotOrd(E)$ and
$Before(ek, ej) \notin TotOrd(E)$, the previous lemma allows us to conclude that $|Before(ei, ej)|_Q = m + n$, with $m > 1$, while, from $Before(ei, ej) \notin TotOrd(E)$, it follows that $|Before(ei, ej)|_Q = m' + n$, with $m' < 1$ (contradiction).

From Lemma 5.3 [3], it follows that Q is semi-acceptable wrt the level mapping $| \; |_Q$ and the model I_Q.

To conclude the proof, we need to identify a suitable level mapping $|| \; ||_P$. Let $||A||_P$ be equal to $|A|_P + 2n$. For each ground instance
$A \leftarrow A_1, .., A_{i-1}, B, A_{i+1}, .., A_n$ of a clause from P such that $I \models A_1, .., A_{i-1}$

(a) if *rel(B)* is defined in P, then $||A||_P \geq ||B||_P$ follows from $|A|_P > |B|_P$;

(b) if *rel(B)* is defined in Q, then $||A||_P \geq |B|_Q$ follows from $|A|_P > 2n$ and $|B|_Q \leq n - 1 + 1 + n = 2n$

This allows us to conclude that $P_{EC} = P \cup Q$ is semi-acceptable, and then left terminating, wrt $|\ |_{P \cup Q}$ and $I_{P \cup Q}$.
Recall that $|\ |_{P \cup Q}$ is defined as follows:

1. if *rel(A)* is defined in P, then $|A|_{P \cup Q} = |A|_P + ||A||_P$;
2. if *rel(A)* is defined in Q, then $|A|_{P \cup Q} = |A|_Q$

Q.E.D.

References

[1] K. A. Apt, H. A. Blair, A. Walker, *Towards a Theory of Declarative Knowledge.* in Foundations of Deductive Databases and Logic Programming, J. Minker (ed.), Morgan Kaufman publ., Los Altos CA, 1988.

[2] K. A. Apt, *Logic Programming.* in Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics, Jan van Leeuwen (ed.), Elsevier and The MIT Press, 1990, pp. 493–574.

[3] K. A. Apt, D. Pedreschi, *Modular Termination Proofs for Logic and Pure Prolog Programs.* Technical Report TR-6/93, Università di Pisa, Dipartimento di Informatica, 1993 (to appear also in Proceedings of Fourth International School for Computer Science Researchers, Acireale, G. Levi (ed.), Oxford University Press).

[4] L. Chittaro, A. Montanari, *Reasoning about Discrete Processes in a Logic Programming Framework.* Proceedings of GULP'93 - Eight Italian Conference on Logic Programming, Gizzeria Lido (CZ), Italy, 1993, pp. 407–421.

[5] L. Chittaro, A. Montanari, *Experimenting a Temporal Logic for Executable Specifications in an Engeneering Domain.* in Artificial Intelligence in Engineering VIII, Vol. 1: Design, Methods and Techniques, G. Rzevski, J. Pastor, R.A. Adey (eds), Computational Mechanics Publications & Elsevier Applied Science, Boston and London, 1993, pp. 185–202.

[6] K. Eshghi, *Abductive Planning with Event Calculus.* Proceedings of the 5th International Conference on Logic Programming, Seattle, 1988, pp. 562–579.

[7] C. Evans, *The Macro-Event Calculus: Representing Temporal Granularity.* in Artificial Intelligence in the Pacific Rim. Proceedings of PRICAI'90 Conference, Nagoya, Japan, IOS Press, 1991.

[8] P. Gardenfors, H. Rott *Belief Revision.* to appear in Handbook of Logic in AI and Logic Programming, Vol. IV: Epistemic and Temporal Reasoning, 1993.

[9] R. Kowalski, M. J. Sergot, *A Logic-Based Calculus of Events.* New Generation Computing, Vol. 4, Springer Verlag, 1986, pp. 67–95.

[10] R. Kowalski, *Database Updates in the Event Calculus.* Journal of Logic Programming, Vol. 12, June 1992, pp. 121–146.

[11] J. W. Lloyd, *Foundations of Logic Programming, 2nd Extended Edition.* Springer-Verlag, 1987.

[12] A. Montanari, E. Maim, E. Ciapessoni, E. Ratto, *Dealing with time granularity in the Event Calculus.* Proceedings of FGCS'92, Fifth Generation Computer Systems, Tokyo, Japan, 1992, pp. 702–712.

[13] T. Przymusinski, *On the Declarative Semantics of Deductive Databases and Logic Programs.* in Foundations of Deductive Databases and Logic Programming, J. Minker (ed.), Morgan Kaufman publ., Los Altos CA, 1988, pp. 193–216.

[14] F. Sadri, *Three Recent Approaches to Temporal Reasoning.* in Temporal Logics and their Application, A. Galton (ed.), Academic Press, 1987, pp. 121–168.

[15] M. J. Sergot, *(Some Topics in) Logic Programming in AI.* Lecture Notes of the GULP, Advanced School on Logic Programming, Alghero, Italy, 1990 (*Unpublished*).

[16] M. Shanahan, *Prediction is Deduction, but Explanation is Abduction.* Proceedings of IJCAI'89 Conference, Detroit. The MIT press, 1989, pp. 1055–1060.

[17] M. Shanahan, *Representing Continuous Change in the Event Calculus*, Proceedings of ECAI'90 Conference, Stockholm, 1990, pp. 598–603.

[18] S. Sripada, *Temporal Reasoning in Deductive Databases.* PhD thesis in Computing, Imperial College. London, 1990.

I. Cervesato
Dipartimento di Informatica
Università di Torino
Corso Svizzera 185, Torino,
I-10149, Italy.
iliano@di.unito.it

A. Montanari
Dip. di Matematica e Informatica
Università di Udine
Via Zanon 6, Udine,
I-33100 Italy.
montanari@uduniv.cineca.it

A. Provetti
C.I.R.F.I.D. *"H. Kelsen"*
Università di Bologna
Via Galliera 3/a, Bologna,
I-40121, Italy.
provetti@cirfid.unibo.it