Modal Event Calculus in Lolli

Iliano Cervesato, Luca Chittaro[*], Angelo Montanari[*]
October 1994
CMU-CS-94-198

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[*]Dipartimento di Matematica e Informatica
Università di Udine
Via Zanon, 6
33100 Udine - ITALY

A one-page abstract of this report will appear in the Proceedings of the *International Symposium on Logic Programming* (*ILPS'94*), Ithaca, NY, 14-17 November 1994.

The work of the first author was partially supported by NSF grant CCR-9303383. The work of the last two authors was partially supported by the P.A.O.L.A. Consortium (ASEM, INSIEL, and Università di Udine).

The authors can be reached by e-mail at *iliano@cs.cmu.edu*, *chittaro@dimi.uniud.it*, *montana@dimi.uniud.it* respectively.

This research was completed while the first author was visiting the Department of Computer Science of Carnegie Mellon University. His permanent address is: *Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino - ITALY.*

Keywords: temporal reasoning, reasoning about actions, nonmonotonic reasoning, logic programming, modal logic, linear logic.

Abstract

This paper introduces two variants of the Event Calculus (EC) with relative timing of events: the Credulous and the Skeptical Event Calculus (CREC and SKEC respectively). A model that takes into account the dynamics of event ordering updates is constructed for EC. In this model, that appears to be a restriction of the usual model for the modal logic S4, CREC and SKEC are proved to be the modal counterparts of EC. It is then shown how a full propositional modal logic based on these calculi can be conveniently and elegantly coded by means of the linear logic programming language Lolli.

1 Introduction

This paper studies the formalization of Kowalski and Sergot's Event Calculus [3, 9, 16, 17, 19, 20] (hereinafter EC) in cases where information about the ordering of events is incomplete [1, 18]. This situation arises in many real-world applications (e.g. [11]). Complete ordering information can be impossible to acquire or it can arrive asynchronously with respect to the recording of event occurrences.

EC is a formalism for reasoning about time and change in a logic programming framework. Given a set of event occurrences, EC allows to derive maximal time intervals (MVIs hereinafter) over which the properties they initiate or terminate hold. However, when only partial knowledge about the ordering of events is given, EC is neither able to derive all possible MVIs nor to distinguish which of the derived intervals are defeasible and which are not. On the contrary, a typical human reasoner who is told a narrative in which the ordering of events is not completely known is able to distinguish among conclusions which can be certainly or possibly drawn. Moreover, he/she is able to change his/her beliefs when provided with new information increasing the ordering. In order to produce comparable results, we developed two variants of EC, namely the skeptical and the credulous EC [4] (*SKEC* and *CREC* respectively). In the presence of a partially ordered sequence of events, SKEC and CREC derive the intervals that are necessarily and possibly true, respectively. The results obtainable with these two variants of EC are amenable to a rather intuitive modal interpretation. The first derives MVIs that will be true in whatever final completion of the ordering, while the second derives those that may be true at least in one possible refinement of the ordering. In [4] we provided both variants with a modal logic interpretation that allows us to formally characterize the state of knowledge about event ordering and its update as well as queries about maximal validity intervals. Furthermore, we discussed an example of their application to a diagnostic case study concerning temporally distributed information about devices behavior. In this work, we provide a different modal interpretation that integrates EC, SKEC and CREC in a uniform modal logic framework. In particular, we show how the two latter calculi can be respectively viewed as the operational counterpart of the modal operators of necessity and possibility.

In the last section, we show how the resulting modal logic can be conveniently and elegantly coded by means of the linear logic programming language Lolli.

2 The plain Event Calculus

EC proposes a general approach to representing and reasoning about events and their effects in a logic programming framework. It takes the notions of event, property, time-point and time-interval as primitives and defines a model of change in which *events* happen at *time-points* and initiate and/or terminate *time-intervals* over which some *property* holds. EC also embodies a notion of *default persistence* according to which properties are assumed to persist until an event that interrupts them occurs. Formally, we represent an event occurrence by means of the *happens* predicate:

```
happens(event).
```

A time-stamp can be attached to an event to record its occurrence time. In this work, we will focus our attention on situations where precise date information for event occurrences are not available. Events will therefore be ordered relatively to each other rather than with respect to an absolute time line. To this purpose, we introduce the possibility of providing factual knowledge about the relative ordering of events, expressed through the predicate *beforeFact*:

```
beforeFact(event1, event2).
```

This representation allows the management of incomplete ordering information. Each time a new piece of ordering information emerges, it is recorded by means of a *beforeFact* fact. EC exploits the transitive closure of this ordering information defined by means of the predicate *before*:

```
before(E1, E2) :-                 before(E1, E2) :-
  beforeFact(E1, E2).               beforeFact(E1, E3), before(E3, E2).
```

The relation between events and properties is defined by means of *initiates* and *terminates* predicates which express the effect of events on properties:

```
initiates(event1, property1).    terminates(event2, property2).
```

This *initiates* (*terminates*) predicate states that *event1* (*event2*) initiates (terminates) a period of time during which *property1* (*property2*) holds.

The plain EC model of time and change is defined by means of a set of axioms. The first axiom we introduce is *holds*. It allows us to state that a property *P* holds maximally between events *Ei* and *Et* if *Ei* initiates *P* and occurs before *Et* that terminates *P*, provided that there is no known interruptions in between:

```
holds(period(Ei, P, Et)) :-
  happens(Ei), initiates(Ei, P),
  happens(Et), terminates(Et, P),
  before(Ei, Et), not broken(Ei, P, Et).
```

The negation involving the *broken* predicate is interpreted as negation-as-failure. This means that properties are assumed to hold uninterrupted over an interval of time on the basis of failure to determine an interrupting event. Should we later record an initiating or terminating event within this interval, we can no longer conclude that the property holds over the interval. This is the non-monotonic aspect of the calculus. The predicate *broken* is defined as follows:

```
broken(Ei, P, Et) :-
  happens(E), before(Ei, E), before(E, Et),
  (initiates(E, Q); terminates(E, Q)),
  exclusive(P, Q).
```

This axiom states that a given property *P* ceases to hold if there is an event *E* that happens between *Ei* and *Et* and initiates or terminates a property *Q* that is incompatible with *P*. The *exclusive(P, Q)* predicate [9] has been introduced as a constraint to force the derivation of *P* to fail when it is possible to conclude that *Q* holds at the same time, and vice versa. Exclusivity is also a convenient means to constrain interferences due to incomplete sequences of events relating to the same property. By adding the axiom

```
exclusive(P, P).
```

we guarantee that the axiom *broken* succeeds also when an initiating or terminating event for property *P* is found between the pair of events *Ei* and *Et* starting and terminating *P*, respectively.

3 Managing the temporal ordering of events

Database updates in EC provide information about the occurrence of events and their occurrence times, and are of additive nature only [10]. Since EC computes MVIs by applying a default persistence rule, an upgrading of its knowledge about events may result in some MVIs no longer derivable. Therefore, the computed set of MVIs, i.e. the collection of *holds(period(ei, r, et))* queries that succeed against EC, can change cardinality and composition in response to updates.

In this work we focus on the case in which the set of event occurrences has been fixed once and for all, and the updates only regard the ordering of events. In such case, the input process consists of the addition of ordering pieces of information in the form of *beforeFact* facts. In [1] we introduced a monotonic version of EC such that the set of computed MVIs monotonically increases with respect to updates. In this work also the opposite view is taken: a variant of EC is defined which derives MVIs every time there is no evidence that the temporal ordering makes them unviable, i.e. when the terminating event does not precede the initiating one. The set of computed MVIs monotonically decreases with respect to updates.

We nickname these two variants of EC *skeptical Event Calculus* (SKEC) and *credulous Event Calculus* (CREC), respectively.

3.1 The skeptical and credulous Event Calculi

SKEC implements a sort of absolute persistence in order to exclude the possibility of deriving information that could be later retracted, provided that the given set of event occurrences does not change. 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* will ever occur after *ei* and before *et*. In such a way, the computed MVIs are indefeasible with respect to refinements of the ordering specification: new ordering pieces coming in may result in new MVIs being derived, but every old MVI is still valid. Thus, the set of MVIs computed by SKEC monotonically increases.

SKEC replaces the predicates *holds* and *broken* of EC with the predicates *skeHolds* and *skeBroken*, respectively, which are defined by the following axioms:

```
skeHolds(period(Ei, P, Et)) :-              skeBroken(Ei, P, Et) :-
   happens(Ei), initiates(Ei,P),               happens(E), E \== Ei, E \== Et,
   happens(Et), terminates(Et, P),             not before(E, Ei), not before(Et, E),
   before(Ei, Et),                             (initiates(E, P1); terminates(E, P1)),
   not skeBroken(Ei, P, Et).                   exclusive(P, P1).
```

Differently from SKEC, CREC stresses the non-monotonism of EC: whenever it is not possible to derive that a terminating event *et* precedes an initiating event *ei*, CREC assumes that *et* follows *ei*. Such an assumption allows CREC to compute all MVIs which are not incompatible with a given set of partially ordered events. More precisely, CREC computes every MVI that holds with respect to at least one possible completion of the given partial ordering of events. When new information about the event ordering is added, this set of computed MVIs monotonically decreases. Further constraining the ordering of events may indeed invalidate previously computed MVIs, but it never forces CREC to compute new MVIs. The input process can thus be viewed as a way of progressively selecting from the initial set of all possible MVIs the subset of MVIs derivable from a totally ordered set of events.

The axioms of CREC are the same of EC but for the replacement of *before(Ei,Et)* with the negation of *before(Et,Ei)* in the definition of *holds*. The resulting predicate *creHolds* is defined as follows:

```
creHolds(period(Ei, P, Et)) :-
   happens(Ei), initiates(Ei, P),
   happens(Et), terminates(Et, P),
   not before(Et, Ei), not broken(Ei, P, Et).
```

3.2 Summary of the axioms

For the ease of the reader, we collect together the Prolog axioms of the event calculi presented so far.

```
holds(period(Ei, P, Et)) :-                                        (1)
  happens(Ei), initiates(Ei, P),
  happens(Et), terminates(Et, P),
  before(Ei, Et),
  not broken(Ei, P, Et).

broken(Ei, P, Et) :-                                               (2)
  happens(E),
  before(Ei, E), before(E, Et),
  (initiates(E, P1); terminates(E, P1)),
  exclusive(P, P1).

skeHolds(period(Ei, P, Et)) :-                                     (3)
  happens(Ei), initiates(Ei, P),
  happens(Et), terminates(Et, P),
  before(Ei, Et),
  not skeBroken(Ei, P, Et).

skeBroken(Ei, P, Et) :-                                            (4)
  happens(E),
  E \== Ei, E \== Et,
  not before(E, Ei), not before(Et, E),
  (initiates(E, P1); terminates(E, P1)),
  exclusive(P, P1).

creHolds(period(Ei, P, Et)) :-                                     (5)
  happens(Ei), initiates(Ei, P),
  happens(Et), terminates(Et, P),
  not before(Et, Ei),
  not broken(Ei, P, Et).

exclusive(P, P).                                                   (6)

before(E1, E2) :-                                                  (7)
  beforeFact(E1, E2).
before(E1, E2) :-                                                  (8)
  beforeFact(E1, E3),
  before(E3, E2).
```

Clauses (1-8) are generically called the *axioms of the extended Event Calculus* and are referred to as `EC+`. Clauses (1-2, 6-8) constitute the *axioms of EC* and are denoted as `EC`. We define the *axioms of CREC* (`CREC`) as clauses (2, 4, 6-8) and the *axioms of SKEC* (`SKEC`) as clauses (3-4, 6-8) in a similar way. A collection of `happens`, `initiates`, `terminates` and `exclusive` facts is called *factual knowledge*; the letter `F` will be used to indicate it. Finally, a collection of `beforeFact` facts is referred to as *ordering information* and is denoted with the letter `O`.

4 A modal logic reconstruction of the Event Calculus

The clausal definitions for EC, SKEC and CREC have a formal counterpart as a modal theory where orderings are interpreted as possible worlds, each one denoting a different state of knowledge. According to this interpretation, an MVI derived by EC translates into a formula which is true in the current world, and its derivability according to SKEC and CREC corresponds to the truth of the formula in every accessible world and in at least one of them, respectively.

4.1 Formalization of the Event Calculus with relative timing

In order to formalize EC, we first give a precise description of the factual knowledge, i.e. the events under consideration, their relationships with the properties they initiate and terminate, and the exclusivity relation among the latter.

A *structure for the Event Calculus with relative timing* (hereinafter *EC structure*) is a quintuple $H = (E, P, [.\rangle, \langle.],].,.[)$ such that:

- $E = \{e^1, ..., e^n\}$ is a *set of events*.
- $P = \{p^1, ..., p^m\}$ is a *set of properties*.
- $[.\rangle: P \rightarrow 2^E$ and $\langle.]: P \rightarrow 2^E$ are respectively the *initiating* and the *terminating map of H*. For every property $p \in P$, $[p\rangle$ and $\langle p]$ represent respectively the set of initiating and the set of terminating events of p. We require $[p\rangle$ and $\langle p]$ to be disjoint sets for every $p \in P$.
- $].,.[\subseteq P \times P$ is a reflexive and symmetric relation called the *exclusivity relationship*, and models exclusivity among properties.

In order to formalize the notion of ordering, let us denote with R^+ the transitive closure of a relation R on a set A. We define $o \subseteq E \times E$ to be a *knowledge state* for E if o^+ is a (possibly partial) strict ordering on E, i.e. a relation over E that is irreflexive, transitive and antisymmetric. Let O_E be the set of all knowledge states for E and $W_E \subseteq O_E$ the set of all strict orderings on E (the pedices will be kept implicit when no confusion can arise). Two knowledge states o_1 and o_2 are *equally informative* if $o_1^+ = o_2^+$. This induces an equivalence relation ~ on O. It is easy to prove that $O_{/\sim}$ and W are isomorphic. Therefore, in the following, a knowledge state o will always refer to the corresponding element o^+ of W, unless explicitly stated otherwise.

$(W, \subseteq)$, where $\subseteq$ is the ordinary subset relation, has the structure of an ordered set. Moreover, it can be easily proved that $(W, \cap, \emptyset)$ forms a finite lower semilattice and that for every $w_1, w_2 \in W$, $w_1 \uparrow w_2 = (w_1 \cup w_2)^+$ is the lub of w_1 and w_2 w.r.t. $\subseteq$ whenever this element belongs to W (w_1 and w_2 may indeed contain incompatible pairs).

Let $w \in W$, any $w' \in W$ such that $w \subseteq w'$ is called an *extension* of w. We denote the set of all extensions of w as $Ext(w)$. The following lemma is obvious but extremely important.

Lemma 1 (Monotonicity of extensions)

If $(e_1, e_2) \in w$, then for every $w' \in Ext(w)$, $(e_1, e_2) \in w'$.

Proof

By the definition of extension, $(e_1, e_2) \in w \subseteq w'$. ☑

Let us now define a translation function $\tau(.)$ from an EC structure to its logic programming counterpart, as defined in Section 2. $\tau(.)$ is defined by cases:

- $\tau(E)$ = {`happens(`e`)` : $e \in E$};
- $\tau([.\rangle)$ = {`initiates(`e`, `p`)` : $e \in E, p \in P$ and $e \in [p\rangle$};
- $\tau(\langle.])$ = {`terminates(`e`, `p`)` : $e \in E, p \in P$ and $e \in \langle p]$};
- $\tau(].,.[)$ = {`exclusive(`p`, `q`)` : $p, q \in P$ and $]p, q[$}.

We define $\tau(H)$ as the union of these sets, plus the axioms of EC, SKEC and CREC (except the exclusivity axiom) reported in section 3.2. It can be easily proven that $\tau(.)$ is well-defined and is indeed an isomorphism between EC theories and syntactically correct Prolog programs for the event calculi.

Moreover, we extend $\tau(.)$ to provide a translation of states of knowledge:

- for every state of knowledge $o \in O_E$, $\tau(o) = \{$`beforeFact(`e_1`, `e_2`)` $: (e_1, e_2) \in o\}$.

Notice that the resulting function continues to be an isomorphism.

4.2 A uniform modal framework for EC, SKEC and CREC

We investigate now the properties of EC structures and their relationships to the event calculi introduced in Sections 2 and 3. We first need a language where to interpret the query predicates (*holds*, *skeHolds* and *creHolds*) of EC, SKEC and CREC.

Let $H = (E, P, [.\rangle, \langle.],].,.[)$ be an EC structure. The *base language* for H is defined as $L_E = \{p(e_1, e_2) : e_1, e_2 \in E, p \in P\}$. The *modal language* for H is $L^{\#}_E = \{\alpha, \Box\alpha, \Diamond\alpha : \alpha \in L_E\}$. The *propositional modal language* for H, $L^{@}_E$, is the modal language obtained by drawing the propositional letters from the base language of H, and combining them by means of the usual propositional connectives and the modal operators. $L^{@}_E$ is defined by the following grammar, where, as above, α ranges over strings in the base language:

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Box\varphi \mid \Diamond\varphi$$

We call *EC formulae* the elements of $L^{@}_E$. The simplified notations L, $L^{\#}$ and $L^{@}$ will be used when no ambiguities arise. In this section, we focus on L and $L^{\#}$. In particular, we show that EC formulae in $L^{\#}$ can be given a meaning in a restriction of the S4 modal logic and that this meaning is the formal counterpart of the set of derivable MVIs in EC, SKEC and CREC. The full language $L^{@}$ will be analysed in the next section.

We now introduce a modal interpretation for EC formulae. Given an EC structure $H = (E, P, [.\rangle, \langle.],].,.[)$, an *EC-frame* for H is defined as the structure $F = (W, \subseteq, \upsilon)$ where

- $(W, \subseteq)$ is the ordered set of all strict orderings on E defined above (where $\subseteq$ assumes the role of *accessibility relation*);
- $\upsilon : L \times W \rightarrow \{true, false\}$ is called the *evaluation function* of H and is such that $\upsilon(p(e_1, e_2), w) = true$ iff
  - $e_1 \in [p\rangle$; (*i*)
  - $e_2 \in \langle p]$; (*ii*)
  - $(e_1, e_2) \in w$; (*iii*)
  - $\neg\exists e \in E. ((e_1, e) \in w \wedge (e, e_2) \in w \wedge \exists q \in P. ((e \in [q\rangle \vee e \in \langle q]) \wedge]p, q[))$. (*iv*)

  The last condition expresses the requirement that p holds uninterruptedly between e_1 and e_2. As a matter of convenience, let us denote it as $nb(p, e_1, e_2, w)$. Notice also that, since w is a strict ordering, $(e_1, e) \in w$ and $(e, e_2) \in w$ entail that $e_1 \neq e$ and $e \neq e_2$.

Notice that EC-frames structurally constitute a subclass of the usual S4-frames. Indeed, the latter are characterized by enforcing the reflexivity and the transitivity of their accessibility relation. Here, we add a request for antisymmetry. Therefore, every S4-valid formula holds in an EC-frame, but the converse is not true in general.

Finally, we define the notion of *satisfiability* of a formula in a knowledge state by means of the relation $w \models \varphi$:

- $w \models p(e_1, e_2)$ iff $\upsilon(p(e_1, e_2), w) = true$;
- $w \models \Box\varphi$ iff for every $w' \in W$ such that $w \subseteq w'$, $w' \models \varphi$;

- $w \models \lozenge\varphi$ iff there is $w' \in W$ such that $w \subseteq w'$ and $w' \models \varphi$;
- $w \models \neg\varphi$ iff $w \models \varphi$ does not hold;
- $w \models \varphi_1 \wedge \varphi_2$ iff $w \models \varphi_1$ and $w \models \varphi_2$;
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.

Lemma 2 (Pointwise condition for necessity)

Let $H = (E, P, [.\rangle, \langle.],].,.[)$ be an EC structure. Then for any $e_1, e_2 \in E$, $p \in P$ and $w \in W$, $w \models \Box p(e_1, e_2)$ iff the following conditions are satisfied:

- $e_1 \in [p\rangle$
- $e_2 \in \langle p]$
- $(e_1, e_2) \in w$
- $nsb(p, e_1, e_2, w)$,

where $nsb(p, e_1, e_2, w)$ stands for the expression

$$\begin{aligned}\forall e \in E.\ \forall q \in P.\ (e = e_1 & \\ \vee\ e = e_2 & \\ \vee\ (e, e_1) \in w & \\ \vee\ (e_2, e) \in w & \\ \vee\ (e \in [q\rangle \vee e \in \langle q] \Rightarrow \neg]p,q[)) &\end{aligned}$$

Proof

($\Leftarrow$) Let us proceed by contradiction. So, assume that $e_1 \in [p\rangle$, $e_2 \in \langle p]$, $(e_1, e_2) \in w$ and $nsb(p, e_1, e_2, w)$, but there exist an extension w' of w such that $w' \models p(e_1, e_2)$ does not hold, i.e. such that $nb(p, e_1, e_2, w')$ is false. After some logical manipulations, the latter statement rewrites to

$$\exists e \in E.\ \exists q \in P.\ ((e_1, e) \in w' \wedge (e, e_2) \in w' \wedge (e \in [q\rangle \vee e \in \langle q]) \wedge]p, q[)$$

Let $\hat{e}$ and q' witness the validity of this formula. By instantiation, we obtain:

$$(e_1, \hat{e}) \in w' \wedge (\hat{e}, e_2) \in w' \wedge (\hat{e} \in [q'\rangle \vee \hat{e} \in \langle q']) \wedge]p, q'[\qquad \textit{(A)}$$

We can instantiate the expression for $nsb(p, e_1, e_2, w)$ with these values too. The resulting formula is:

$$\hat{e} = e_1 \vee \hat{e} = e_2 \vee (\hat{e}, e_1) \in w \vee (e_2, \hat{e}) \in w \vee (\hat{e} \in [q'\rangle \vee \hat{e} \in \langle q'] \Rightarrow \neg]p,q'[) \qquad \textit{(B)}$$

We must show that none of the alternatives in formula *(B)* applies. Since w is a strict order, the validity of *(A)* implies that $\hat{e}$ can be neither e_1 nor e_2. Analogously, by lemma 1, either $(\hat{e}, e_1) \in w$ or $(e_2, \hat{e}) \in w$ would violate the antisymmetry of w'. Finally, the choice of q' contradicts the last alternative, i.e. that $(\hat{e} \in [q'\rangle \vee \hat{e} \in \langle q'] \Rightarrow \neg]p,q'[)$. This concludes this direction of the proof.

($\Rightarrow$) We will again proceed by contradiction. Clearly, if $e_1 \notin [p\rangle$ or $e_2 \notin \langle p]$, then we cannot obtain $w' \models p(e_1, e_2)$ in any state of knowledge w'. If $(e_1, e_2) \notin w$, then there exist extensions of w containing (e_2, e_1). Because of antisymmetry, these extensions cannot contain (e_1, e_2), thus, $p(e_1, e_2)$ cannot be valid in them.

Assume now that $nsb(p, e_1, e_2, w)$ does not hold. Therefore, there are an event $\hat{e}$ and a property q' such that:

$$\hat{e} \neq e_1 \wedge \hat{e} \neq e_2 \wedge (\hat{e}, e_1) \notin w \wedge (e_2, \hat{e}) \notin w \wedge (\hat{e} \in [q'\rangle \vee \hat{e} \in \langle q']) \wedge]p,q'[$$

Since the pair $(e_1, e_2) \in w$, there exists at least one extension w' of w such that $(e_1, \hat{e}) \in w'$ and $(\hat{e}, e_2) \in w'$. Therefore,

$(e_1, \hat{e}) \in w' \wedge (\hat{e}, e_2) \in w' \wedge (\hat{e} \in [q'\rangle \vee \hat{e} \in \langle q']) \wedge \,]p, q'[$

hence $nb(p, e_1, e_2, w')$ does not hold. This contradicts the hypothesis that $w \models \Box p(e_1, e_2)$. ☑

Notice that, differently from $nb(p, e_1, e_2, w)$, $nsb(p, e_1, e_2, w)$ requires that for all $e \in E$ satisfying certain conditions, $(e, e_1) \in w$ and not only $(e_1, e) \notin w$; similarly for (e_2, e).

Lemma 3 (Pointwise condition for possibility)

Let $H = (E, P, [.\rangle, \langle .],].,.[)$ be an EC structure. Then for any $e_1, e_2 \in E$, $p \in P$ and $w \in W$, $w \models \Diamond p(e_1, e_2)$ iff the following conditions are satisfied:

- $e_1 \in [p\rangle$
- $e_2 \in \langle p]$
- $(e_2, e_1) \notin w$
- $nb(p, e_1, e_2, w)$.

Proof

(⇐) Let us construct an extension w' of w such that $w' \models p(e_1, e_2)$. The state of knowledge w' is defined as $w' = (w \cup \{(e_1, e_2)\})^+$. First notice that w' is consistent (i.e. it does not violate antisymmetry) since w is consistent and $(e_2, e_1) \notin w$. Then observe that $nb(p, e_1, e_2, w')$ holds by the definition of w' and the monotonicity of extensions. Otherwise, we should be able to conclude that there is an event $e \in E$ such that $(e_1, e) \in w'$, $(e, e_2) \in w'$ and $e \in [q\rangle$ or $e \in \langle q]$ for some property $q \in P$ with $]p, q[$, but in that case, $(e_1, e) \in w$ and $(e, e_2) \in w$ contradicting the assumption that $nb(p, e_1, e_2, w)$. Therefore, conditions *(i-iv)* are satisfied w.r.t. w', hence $w' \models p(e_1, e_2)$, thus $w \models \Diamond p(e_1, e_2)$.

(⇒) We proceed by contradiction. Clearly, if $e_1 \notin [p\rangle$ or $e_2 \notin \langle p]$, then we cannot obtain $w' \models p(e_1, e_2)$ in any state of knowledge w'. Analogously, if $(e_2, e_1) \in w$, then (e_2, e_1) belongs to every extension of w, forbidding in this way the condition *(iii)* of the definition of evaluation to be satisfied. Finally, if $nb(p, e_1, e_2, w)$ does not hold (i.e. there is an event $e \in E$ such that $(e_1, e) \in w$, $(e, e_2) \in w$ and $e \in [q\rangle$ or $e \in \langle q]$ for some property $q \in P$ with $]p, q[$), then by lemma 1, the same condition would apply to every extension w' as well, thus $nb(p, e_1, e_2, w')$ would not hold in any extension w' of w. ☑

Lemma 4 (Soundness and completeness of `before` w.r.t. transitive closure)

Let $H = (E, P, [.\rangle, \langle .],].,.[)$ be an EC structure and o a state of knowledge, then

$\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`e_1`, `e_2`)` iff $(e_1, e_2) \in o^+$.

Proof

(⇒) We will prove the statement by induction on the height of a resolution tree for $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`e_1`, `e_2`)`. If the height is 1, then clause (7) in section 2.3 must have been used. Thus, $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `beforeFact(`e_1`, `e_2`)` and therefore, $(e_1, e_2) \in o \subseteq o^+$. Otherwise, let us assume that this tree has height h+1, the first rule applied must be clause (8). By unfolding, we obtain that $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `beforeFact(`e_1`, `e`)` and $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`e`, `e_2`)`, for some event e, where the latter has a derivation tree of height h. Thus, $(e_1, e) \in o$ and by the induction hypothesis, $(e, e_2) \in o^+$. Now, by the definition of transitive closure, $(e_1, e_2) \in o^+$.

(⇐) Let $\sigma = \hat{e}_1, ..., \hat{e}_l$, with $\hat{e}_1 = e_1$ and $\hat{e}_l = e_2$ be a sequence of events such that for i=1..l-1 $(\hat{e}_i, \hat{e}_{i+1}) \in o$, proving in this way that $(e_1, e_2) \in o^+$. We conduce the proof by induction on the length l of this sequence. If $l = 1$, then $(e_1, e_2) \in o$, thus $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `beforeFact(`e_1`, `e_2`)` and, by clause

(7), $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`e_1`, `e_2`)`. Otherwise, $\sigma = e_1, \hat{e}, ..., e_2$, with $(e_1, \hat{e}) \in o$ and $(\hat{e}, e_2) \in o^+$. By induction hypothesis, we have that $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `beforeFact(`e_1`, `$\hat{e}$`)` and, $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`$\hat{e}$`, `e_2`)`. Finally, by applying clause (8), we obtain that $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`e_1`, `e_2`)`. ☑

We can now state the major result of this paper:

Theorem 5 (Soundness and completeness of EC, SKEC and CREC w.r.t. the EC-frame semantics)

Let $H = (E, P, [.\rangle, \langle .],].,.[)$ be an EC structure and o a state of knowledge, then

a) $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `holds(period(`e_1`, `p`, `e_2`))` iff $o^+ \models p(e_1, e_2)$;

b) $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `skeHolds(period(`e_1`, `p`, `e_2`))` iff $o^+ \models \Box p(e_1, e_2)$;

c) $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `creHolds(period(`e_1`, `p`, `e_2`))` iff $o^+ \models \Diamond p(e_1, e_2)$.

Proof

a) $(\Rightarrow)$ Assume first that $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `holds(period(`e_1`, `p`, `e_2`))`. We first prove that, with this hypothesis, $\upsilon(p(e_1, e_2), o^+) = true$; the thesis will follow by the definition of validity.

By unfolding clause (1) we obtain the following relations, where the right column shows the translation of the Prolog code on the left column into the EC theory H, according to τ^{-1}.

• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `happens(`e_1`)`	a.1	$e_1 \in E$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `initiates(`e_1`, `p`)`	a.2	$e_1 \in [p\rangle$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `happens(`e_2`)`	a.3	$e_2 \in E$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `terminates(`e_2`, `p`)`	a.4	$e_2 \in \langle p]$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`e_1`, `e_2`)`	a.5	$(e_1, e_2) \in o^+$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `not broken(`e_1`, `p`, `e_2`)`	a.6	

By the soundness and completeness of SLDNF resolution for ground negative calls and non recursive negative clauses [12], a.6 is a valid relation if and only if `broken(`e_1`, `p`, `e_2`)` is not derivable from $\tau(H)$ and $\tau(o)$ through SLDNF resolution. By unfolding clause (2), we obtain that $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `broken(`e_1`, `p`, `e_2`)` if and only if:

• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `happens(`e`)`	a.7	$e \in E$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`e_1`, `e`)`	a.8	$(e_1, e) \in o^+$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `before(`e`, `e_2`)`	a.9	$(e, e_2) \in o^+$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `initiates(`e`, `q`); terminates(`e`, `q`)`	a.10	$e \in [q\rangle \vee e \in \langle q]$
• $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `exclusive(`p`, `q`)`	a.11	$]p, q[$

As highlighted in the right column, by the definition of τ and lemma 4, $\tau(H), \tau(o) \vdash_{\text{SLDNF}}$ `broken(`e_1`, `p`, `e_2`)` will succeed if and only if there is an event $e \in E$ such that $(e_1, e) \in o^+$, $(e, e_2) \in o^+$, $e \in [q\rangle$ or $e \in \langle q]$, and $]p, q[$ for some property $q \in P$. Therefore, the previously stated relation a.6 holds if and only if $\neg\exists e \in E.\ ((e_1, e) \in o^+ \wedge (e, e_2) \in o^+ \wedge \exists q \in P.\ ((e \in [q\rangle \vee e \in \langle q]) \wedge\ ]p, q[))$, i.e. iff $nb(p, e_1, e_2, o^+)$, but this is precisely the condition *(iv)* of the definition of evaluation.

Now, it suffices to notice that the relation $a.2$, $a.4$ and $a.5$ correspond respectively to the conditions *(i)*, *(ii)* and *(iii)* of this definition. Therefore $\upsilon(p(e_1, e_2), o^+) = true$ and thus $o^+ \models p(e_1, e_2)$.

a) ($\Leftarrow$) It suffices to notice that the proof just given is indeed bi-directional, even if this fact has not been stressed for the sake of readability.

b) ($\Rightarrow$) The technique used for proving a) will be applied again. Therefore, assume that $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}}$ $\texttt{skeHolds(period(}e_1\texttt{, }p\texttt{, }e_2\texttt{))}$. Let us first prove that, with this hypothesis, $e_1 \in [p\rangle$, $e_2 \in \langle p]$, $(e_1, e_2) \in o^+$ and $nsb(p, e_1, e_2, o^+)$; the thesis will follow by lemma 2.

By unfolding clause (3) we obtain the following relations, where the right column shows the translation of the Prolog code on the left column into the EC theory H, according to τ^{-1}.

• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{happens(}e_1\texttt{)}$	$b.1$	$e_1 \in E$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{initiates(}e_1\texttt{, }p\texttt{)}$	$b.2$	$e_1 \in [p\rangle$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{happens(}e_2\texttt{)}$	$b.3$	$e_2 \in E$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{terminates(}e_2\texttt{, }p\texttt{)}$	$b.4$	$e_2 \in \langle p]$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{before(}e_1\texttt{, }e_2\texttt{)}$	$b.5$	$(e_1, e_2) \in o^+$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{not skeBroken(}e_1\texttt{, }p\texttt{, }e_2\texttt{)}$	$b.6$	

Again by [12], $b.6$ is a valid relation if and only if $\texttt{skeBroken(}e_1\texttt{, }p\texttt{, }e_2\texttt{)}$ is not derivable from $\tau(H)$ and $\tau(o)$ through SLDNF resolution. By unfolding clause (4), we obtain that $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}}$ $\texttt{skeBroken(}e_1\texttt{, }p\texttt{, }e_2\texttt{)}$ if and only if:

• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{happens(}e\texttt{)}$	$b.7$	$e \in E$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} e \texttt{ \textbackslash== } e_1$	$b.8$	$e \neq e_1$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} e \texttt{ \textbackslash== } e_2$	$b.9$	$e \neq e_2$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{not before(}e\texttt{, }e_1\texttt{)}$	$b.10$	$(e, e_1) \notin o^+$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{not before(}e_2\texttt{, }e\texttt{)}$	$b.11$	$(e_2, e) \notin o^+$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{initiates(}e\texttt{, }q\texttt{); terminates(}e\texttt{, }q\texttt{)}$	$b.12$	$e \in [q\rangle \vee e \in \langle q]$
• $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}} \texttt{exclusive(}p\texttt{, }q\texttt{)}$	$b.13$	$]p, q[$

As highlighted in the right column, by the definition of τ and by lemma 4, $\tau(H), \tau(o) \vdash_{\mathrm{SLDNF}}$ $\texttt{skeBroken(}e_1\texttt{, }p\texttt{, }e_2\texttt{)}$ succeeds if and only if there exists an event $e \in E$ distinct from e_1 and e_2 such that $(e, e_1) \notin o^+$, $(e_2, e) \notin o^+$, either $e \in [q\rangle$ or $e \in \langle q]$, and $]p, q[$ for some property $q \in P$. Therefore, the relation $b.6$ above holds if and only $nsb(p, e_1, e_2, o^+)$ holds.

Now, it suffices to notice that the relations $b.2$, $b.4$, $b.5$ and $b.6$ correspond to the conditions of lemma 2. Therefore $o^+ \models \Box p(e_1, e_2)$.

b) ($\Leftarrow$) Again, the previous proof is bi-directional.

c) ($\Rightarrow$) Assume that $\tau(H), \tau(o) \vdash_{SLDNF}$ `creHolds(period(`e_1`, `p`, `e_2`))`. Let us first prove that, with this hypothesis, $e_1 \in [p\rangle$, $e_2 \in \langle p]$, $(e_2, e_1) \notin o^+$ and $nb(p, e_1, e_2, o^+)$; the thesis will follow by lemma 3.

By unfolding clause (5) we obtain the following relations, where the right column shows the translation of the Prolog code on the left column into the EC theory H, according to τ^{-1}.

• $\tau(H), \tau(o) \vdash_{SLDNF}$ `happens(`e_1`)`	c.1	$e_1 \in E$
• $\tau(H), \tau(o) \vdash_{SLDNF}$ `initiates(`e_1`, `p`)`	c.2	$e_1 \in [p\rangle$
• $\tau(H), \tau(o) \vdash_{SLDNF}$ `happens(`e_2`)`	c.3	$e_2 \in E$
• $\tau(H), \tau(o) \vdash_{SLDNF}$ `terminates(`e_2`, `p`)`	c.4	$e_2 \in \langle p]$
• $\tau(H), \tau(o) \vdash_{SLDNF}$ `not before(`e_2`, `e_1`)`	c.5	$(e_2, e_1) \notin o^+$
• $\tau(H), \tau(o) \vdash_{SLDNF}$ `not broken(`e_1`, `p`, `e_2`)`	c.6	$nb(p, e_1, e_2, o^+)$

where the last line derives from the proof of a).

The relations c.2, c.4, c.5 and c.6 correspond to the conditions of lemma 3. Therefore $o^+ \models \Diamond p(e_1, e_2)$.

c) ($\Leftarrow$) Once more, the previous proof is bi-directional. ☑

These results are the formal counterpart of the intuitive description of the behavior of *holds*, *skeHolds* and *creHolds* presented in the previous sections. The relative cardinalities of the sets of values satisfying these predicates are related in the following corollary, which, again, agrees with our intuition.

Corollary 6

Let `F` and `O` be the factual knowledge and the ordering information of a specific instance of the Event Calculus, then

a) if `EC+`, `F`, `O` $\vdash_{SLDNF}$ `skeHolds(period(`e_1`, `p`, `e_2`))`
then `EC+`, `F`, `O` $\vdash_{SLDNF}$ `holds(period(`e_1`, `p`, `e_2`))`

b) if `EC+`, `F`, `O` $\vdash_{SLDNF}$ `holds(period(`e_1`, `p`, `e_2`))`
then `EC+`, `F`, `O` $\vdash_{SLDNF}$ `creHolds(period(`e_1`, `p`, `e_2`))`

Proof

These relations can be rewritten in the following way:

Let $H = \tau^{-1}(\texttt{EC+} \cup \texttt{F})$ and $o = \tau^{-1}(\texttt{O})$, then

a) if $H, o^+ \models \Box p(e_1, e_2)$, then $H, o^+ \models p(e_1, e_2)$;

b) if $H, o^+ \models p(e_1, e_2)$, then $H, o^+ \models \Diamond p(e_1, e_2)$.

The validity of these expressions is a direct consequence of the reflexivity of the accessibility relation of EC-frames. Indeed, $H, o^+ \models \Box p(e_1, e_2)$ iff $p(e_1, e_2)$ is valid in every extension of o^+, in particular in o^+ itself. Analogously, if $H, o^+ \models p(e_1, e_2)$, then $H, o^+ \models \Diamond p(e_1, e_2)$ by reflexivity. ☑

We can restate this corollary in a form closer to the Event Calculus terminology. Let MVI_{EC}, MVI_{SKEC} and MVI_{CREC} denote the set of MVIs derivable from EC, SKEC and CREC respectively in a given state of knowledge o, then $\text{MVI}_{\text{SKEC}}(o) \subseteq \text{MVI}_{\text{EC}}(o) \subseteq \text{MVI}_{\text{CREC}}(o)$.

5 Handling propositional connectives in EC, SKEC and CREC

In the previous chapter, we proved that EC, SKEC and CREC can be given a uniform modal semantics by interpreting these calculi into EC-frames. These are indeed specific forms of S4-frames. Therefore, the theoretical results obtained in the latter setting apply to the former. In particular, S4-frames provide a model for the usual boolean connectives. We could take advantage of this fact, obtaining in this way a more powerful unified calculus, if we only were able to implement the connectives in such a way to take into account their interaction with the modalities. This is problematic in Prolog. In fact, when dealing with the modalities, we often need to explore some or even all of the possible future states of knowledge. Using *assert* and *retract* statements for this purpose is awkward and lacks flexibility. Another solution, experimented to a limited extent in [4], consists in maintaining the extensions to the current knowledge state into a hypotheses list; this is bulky and solves the problem only partially.

With these considerations in mind, we decided to adopt a different implementation language. Specifically, we turned our attention to the linear logic programming language Lolli, that is indeed an extension of Prolog. Lolli is based on linear logic and therefore allows to model the retraction of a fact from the current knowledge state as a resource consumption. Moreover, by allowing implications in goal, this language, as well as its precursor λProlog, provide an effective and logical means of temporarily asserting new facts into the database.

5.1 The linear logic programming language Lolli

Miller et al. [15] have defined a general criterion (existence of *uniform proofs*) for extracting fragments of logical systems suited to be implemented as logic programming languages. The language of Horn-clauses, on which Prolog is based, has been proven to have the required properties, and has indeed been shown to exploit only to a minimal extent the connective patterns of predicate logic that ensure the existence of such proofs. In particular, they show that it is possible to go beyond Horn-clauses by allowing implication and universal quantification in goals. While not prejudicing the possibility of an efficient implementation, this enhancement increases enormously the expressiveness of the language [14]. The language λProlog is a realization of this idea in a simply-typed higher-order setting [13, 15].

λProlog stems from intuitionistic predicate logic. However, uniform proofs are a general concept in proof theory. The logic programming language Lolli [6, 7] is the result of applying this criterion to linear logic [5]. The short space available forbids us to give a detailed account of linear logic and to present a complete description of Lolli. The interested reader is invited to read the above mentioned references. We intend instead to illustrate only those aspects that are needed in the subsequent.

Linear logic is a refinement of traditional logic that constrains the number of times an assumption is used in a proof. A formula G is derivable from a set of formulae D if and only if there is a proof of G that uses exactly once every formula in D, with the exception of formulae that are preceded by the modal operator ! (read *of-course* or *bang*), which indicates that this formula can be used as many times as needed in the derivation, eventually zero. It is convenient to write linear sequents as $D; R \vdash G$, where G, D and R are respectively the formula to prove, the set of banged assumptions and the set of non-banged premises. R can be seen as the resources available for the proof of G, while the elements of D are costless items. R and D are called the bound and unbound context respectively.

The duplication of the left-hand side of linear sequents, a direct consequence of the constraints on the use of formulae in proofs, entails the definition of a new set of connectives for the resulting logic.

With a drastic simplification, we can say that every binary connective in traditional logic is transformed in a pair of linear connectives, one requiring the bound context to be duplicated when proving its subformulae, and the other having it split among them. The linear connectives on which Lolli is based are ⊗ (multiplicative conjunction, or *times*), & (additive conjunction, or *with*), ⊕ (multiplicative disjunction, or *oplus*), -o (linear implication, or *lollipop*), => (traditional implication), plus ! (bang) and the two constants *true* and *erase*.

The syntax of Lolli is reminiscent of that of Prolog, with some notational differences. Variables begin with an uppercase letter, unless explicitly quantified, while constants begin always with a lowercase letter. Terms and atoms are written in curried form. For instance, the Prolog term `f(a, g(b, c), d)` is written in Lolli as `(f a (g b c) d)` (the outermost parentheses are not needed).

Lolli program and goals are constructed according to the grammar show below, where *A*, *G*, *R* and *D* are syntactic variables for atoms, goal formulae, linear clauses and banged clauses; the use of the *G*, *R* and *D* is coherent with their use in linear sequents above. In the following, we will use expressions built out of the productions in bold face only.

R ::= **`true`** | $\boldsymbol{A}$ | $\boldsymbol{R_1}$ **&** $\boldsymbol{R_2}$ | $\boldsymbol{R}$ **`:-`** $\boldsymbol{G}$ | R <= G | `forall` $x \setminus R$

D ::= $\boldsymbol{R}$ | $\{R\}$ | D_1 , D_2

G ::= **`true`** | **`erase`** | $\boldsymbol{A}$ | $\{G\}$ | $\boldsymbol{G_1}$ **&** $\boldsymbol{G_2}$ | $\boldsymbol{G_1}$ **,** $\boldsymbol{G_2}$ | $\boldsymbol{D}$ **-o** $\boldsymbol{G}$ | $\boldsymbol{R}$ **=>** $\boldsymbol{G}$ | `forall` $x \setminus G$
| `exists` $x \setminus G$ | $\boldsymbol{G_1}$ **;** $\boldsymbol{G_2}$ | $\boldsymbol{G_1}$ **->** $\boldsymbol{G_2}$ **|** $\boldsymbol{G_3}$

We give a connective-directed operational semantics for goals, limiting ourselves to the patterns that will be used in the following. We write $S = S_1 + S_2$ to indicate that S_1 and S_2 form a partition of the set S. In order to keep the notation simple, we write $S+s$ and $S \cup s$ instead of $S+\{s\}$ and $S \cup \{s\}$ when no ambiguity arises.

`true` always succeeds without consuming any resource. Therefore, any formula in the bound context must be used in other parts of the proof of the current goal.

$D; \varnothing \vdash_{\text{LOLLI}} \texttt{true}$

`erase` always succeeds too, but consumes every resource in the bound context. It is usually exploited to ignore information eventually present into the bound context and not needed in the proof of the current goal.

$D; R \vdash_{\text{LOLLI}} \texttt{erase}$

A looks for a linear or banged clause h`:-`b (eventually a fact h) which head unifies with the atomic goal A and tries to solve its body in the current context. If the selected clause is linear, it is removed from the bound context.

$D \cup h\texttt{:-}b; R \vdash_{\text{LOLLI}} A$ if $A^\sigma = h^\sigma$ and $D; R \vdash_{\text{LOLLI}} b^\sigma$

$D ; R + h\texttt{:-}b \vdash_{\text{LOLLI}} A$ if $A^\sigma = h^\sigma$ and $D; R \vdash_{\text{LOLLI}} b^\sigma$

$\{G\}$ (*bang G*) attempts to solve the goal G without using any formula in the bound context.

$D; R \vdash_{\text{LOLLI}} \{G\}$ iff $D; \varnothing \vdash_{\text{LOLLI}} G$

G_1 , G_2 (G_1 *times* G_2) first attempts to find a proof for G_1 and, in case of success, attempts a proof for G_2 that uses whatever element of the bound context were not used in the proof of G_1 . Therefore, it divides the resources in the bound context between G_1 and G_2.

$D; R_1 + R_2 \vdash_{\text{LOLLI}} G_1 , G_2$ iff $D; R_1 \vdash_{\text{LOLLI}} G_1$ and $D; R_2 \vdash_{\text{LOLLI}} G_2$

G_1 `&` G_2 — (G_1 *with* G_2) operates as in the previous case with the difference that it tries to prove both G_1 and G_2 using the same resources. Therefore, it duplicates the resources in the bound context for each conjunct.

$D; R \vdash_{\text{LOLLI}} G_1$ `&` G_2 iff $D; R \vdash_{\text{LOLLI}} G_1$ and $D; R \vdash_{\text{LOLLI}} G_2$

G_1 `;` G_2 — (G_1 *oplus* G_2) first attempts to find a proof for G_1 and, in case of success, have the overall goal succeed. In case of failure, a proof for G_2 is attempted using the same set of resources as for G_1.

$D; R \vdash_{\text{LOLLI}} G_1$ `;` G_2 iff $D; R \vdash_{\text{LOLLI}} G_1$ or $D; R \vdash_{\text{LOLLI}} G_2$

r `-o` G — (r *lollipop* G) causes the clause r to be added to the bound context and a proof for G to be attempted from this augmented program.

$D; R \vdash_{\text{LOLLI}} r$ `-o` G iff $D; R + r \vdash_{\text{LOLLI}} G$

$d \Rightarrow G$ — (d *implies* G) causes the clause d to be added to the unbound context and to attempt a proof for the goal G from this augmented context. It is logically equivalent to $\{d\}$`-o` G.

$D; R \vdash_{\text{LOLLI}} d \Rightarrow G$ iff $D \cup d; R \vdash_{\text{LOLLI}} G$

G`->`G_s`|`G_f — (guarded expression) is the one extra-logical operator included into the current implementation of Lolli. Such a goal attempts a proof of G. In case of success, the overall goal succeeds if G_s succeeds. If instead the interpreter fails to find a proof for G, it tries to find a proof for G_f and its success determines the success of the overall goal.

$D; R \vdash_{\text{LOLLI}} G$ `->` G_s `|`G_f iff if $D; R \vdash_{\text{LOLLI}} G$ then $D; R \vdash_{\text{LOLLI}} G_s$
else $D; R \vdash_{\text{LOLLI}} G_f$

Such guarded expressions have been used in our implementation to model negation as failure. In fact,

```
  not G :- G -> fail | true.
```

Linear and banged clauses are entered into the appropriate context either as the effect of solving an implication goal, or by reading them from a file. In the latter case, clauses are implicitly banged; linear clauses are preceded by the keyword `LINEAR`.

Implication in goals combined with universal quantification yields a powerful scoping mechanism [14]. Lolli provides a module system realized through the combination of the two. Modules are parametric and permit abstraction and information hiding by means of the `LOCAL` declaration.

5.2 Lolli coding of EC, SKEC and CREC with connectives

Let us now define a coding function σ(.) from EC formulas of the complete language $L^@$ to well-formed Lolli expression. σ(.) is defined recursively as follows.

$$\begin{aligned}
\sigma(p(e_1, e_2)) &= \texttt{(period } e_1\ p\ e_2\texttt{)}\\
\sigma(\Box\varphi) &= \texttt{(must } \sigma(\varphi)\texttt{)}\\
\sigma(\Diamond\varphi) &= \texttt{(may } \sigma(\varphi)\texttt{)}\\
\sigma(\neg\varphi) &= \texttt{(not } \sigma(\varphi)\texttt{)}\\
\sigma(\varphi_1 \wedge \varphi_2) &= \texttt{(and } \sigma(\varphi_1)\ \sigma(\varphi_2)\texttt{)}\\
\sigma(\varphi_1 \vee \varphi_2) &= \texttt{(or } \sigma(\varphi_1)\ \sigma(\varphi_2)\texttt{)}
\end{aligned}$$

Notice that σ(.) is indeed an isomorphism between $L^@$ and Lolli expressions built out of the functors `must`, `may`, `not`, `and` and `or` (where the first three are unary and the last two are binary), and expressions of the form `(period` e_1 p e_2`)` where e_1 and e_2 are the representation of events and p is the representations of a property.

We now turn to the theoretical results that will lead to the implementation of $L^@$. The following lemma provides us with the means for handling generic formulae having a modal quantification as their outermost connective.

Lemma 7

Let $L^@$ be the modal propositional language of an EC structure $H = (E, P, [.\rangle, \langle.],].,.[)$, φ a formula in $L^@$ and $w \in W$, then

a) $w \models \Box\varphi$ iff $w \models \varphi$ and $w \uparrow \{(e_1, e_2)\} \models \Box\varphi$ for every (e_1, e_2) such that $(e_1, e_2) \notin w$ and $(e_2, e_1) \notin w$;

b) $w \models \Diamond\varphi$ iff $w \models \varphi$ or $w \uparrow \{(e_1, e_2)\} \models \Diamond\varphi$ where $(e_1, e_2) \notin w$ and $(e_2, e_1) \notin w$.

Proof

First notice that if $(e_1, e_2) \notin w$ and $(e_2, e_1) \notin w$, then $w \uparrow \{(e_1, e_2)\} \in W$ since, in this case, upgrading with (e_1, e_2) cannot violate antisymmetry in any way.

Moreover, for every $w \in W$,

$$Ext(w) = \{w\} \cup \bigcup_{\substack{(e_1,e_2)\notin w \\ (e_2,e_1)\notin w}} Ext\big(w \uparrow \{(e_1, e_2)\}\big).$$

In fact, let $w' \in Ext(w)$, then, by definition, $w \subseteq w'$. Therefore, either $w' = w$ or there exist a pair $(e_1, e_2) \in w' \setminus w$. In the latter case, $w' \in Ext(w \uparrow \{(e_1, e_2)\})$. The opposite inclusion is straightforward.

We have now all the tools needed to prove the statement of the lemma.

a) $w \models \Box\varphi$ iff for each $w' \in Ext(w)$, $w' \models \varphi$

iff for each $w' \in \{w\} \cup \bigcup_{\substack{(e_1,e_2)\notin w \\ (e_2,e_1)\notin w}} Ext\big(w \uparrow \{(e_1, e_2)\}\big)$, $w' \models \varphi$

iff $w \models \varphi$ and for every (e_1, e_2) such that $(e_1, e_2) \notin w$ and $(e_1, e_2) \notin w$, then for each $w' \in Ext(w \uparrow \{(e_1, e_2)\})$, $w' \models \varphi$

iff $w \models \varphi$ and for every (e_1, e_2) such that (e_1, e_2), $(e_1, e_2) \notin w$, $w \uparrow \{(e_1, e_2)\} \models \Box\varphi$.

b) The proof is completely analogous to that for a). ☑

In the subsequent, we will write a) into a different but clearly equivalent form:

$w \models \Box\varphi$ iff it is not the case that
$w \models \varphi$ does not hold or
$w \uparrow \{(e_1, e_2)\} \models \Diamond\neg\varphi$ for some (e_1, e_2) such that $(e_1, e_2) \notin w$ and $(e_2, e_1) \notin w$.

In S4, many connective have a simple interaction with the modal operators and can be coded by means of elementary rewriting rules. When implementing the unified theory, we can avoid or at least delay visiting the extensions of the current state of knowledge by applying these rules. The following properties are valid in S4, and therefore in EC-frames. A proof can be found in [8].

Property 8

Let L be any propositional modal language, $\models_{S4}$ the validity relation in S4-frames [8] and φ a formula in L, then

- $\models_{S4} \Box\neg\varphi$ iff $\models_{S4} \neg\Diamond\varphi$
- $\models_{S4} \Diamond\neg\varphi$ iff $\models_{S4} \neg\Box\varphi$
- $\models_{S4} \Box\Box\varphi$ iff $\models_{S4} \Box\varphi$
- $\models_{S4} \Diamond\Diamond\varphi$ iff $\models_{S4} \Diamond\varphi$
- $\models_{S4} \Box(\varphi_1 \wedge \varphi_2)$ iff $\models_{S4} \Box\varphi_1 \wedge \Box\varphi_2$
- $\models_{S4} \Diamond(\varphi_1 \vee \varphi_2)$ iff $\models_{S4} \Diamond\varphi_1 \vee \Diamond\varphi_2$ ☑

We can now give the implementation of our extension of EC, SKEC and CREC with the connectives.

```
MODULE ec.

LOCAL broken skeBroken skeHolds_or skeHolds_may.

%-------- Propositional formulae
holds (period Ei P Et) :-                (1)
  happens Ei,
  initiates Ei P,
  happens Et,
  terminates Et P,        erase &
  before Ei Et,           erase &
  not (broken Ei P Et),   erase.

broken Ei P Et :-                        (2)
  happens E,
  before Ei E,            erase &
  before E Et,            erase &
  (initiates E Q ; terminates E Q),
  exclusive P Q,          erase.

exclusive P P.                           (3)

holds (not X) :-                         (4)
  not (holds X),          erase.

holds (and X Y) :-                       (5)
  holds X & holds Y.

holds (or X Y) :-                        (6)
  holds X; holds Y.

%-------- Must-formulae
holds (must (period Ei P Et)) :-         (7)
  happens Ei,
  initiates Ei P,
  happens Et,
  terminates Et P,        erase &
  before Ei Et,           erase &
  not (skeBroken Ei P Et),erase.

skeBroken Ei P Et :-                     (8)
  happens E,
  not (E = Ei),
  not (E = Et),           erase &
  not (before E Ei),      erase &
  not (before Et E),      erase &
  (initiates E Q ; terminates E Q),
  exclusive P Q,          erase.

holds (must (not X)) :-                  (9)
  not (holds (may X)),    erase.

holds (must (and X Y)) :-                (10)
  holds (must X) & holds (must Y).

holds (must (or X Y)) :-                 (11)
  not (skeHolds_or X Y),  erase.

skeHolds_or X Y :-                       (12)
  not (holds X), not (holds Y).
skeHolds_or X Y :-                       (13)
  happens E1, happens E2, erase &
  not (before E1 E2),     erase &
  not (before E2 E1),     erase &
  beforeFact E1 E2 =>
           may (not (hold (or X Y))).

holds (must (must X)) :-                 (14)
  holds (must X).

holds (must (may X)) :-                  (15)
  not (skeHolds_may X),   erase.
```

```
skeHolds_may X :-                          (16)
  not (holds X).
skeHolds_may X :-                          (17)
  happens E1, happens E2, erase &
  not (before E1 E2),     erase &
  not (before E2 E1),     erase &
  beforeFact E1 E2 =>
          may (not (hold (may X))).
```

```
%-------- May-formulae
holds (may (period Ei P Et)) :-            (18)
  happens Ei,
  initiates Ei P,
  happens Et,
  terminates Et P,        erase &
  not (before Et Ei),     erase &
  not (broken Ei P Et),   erase.

holds (may (not X)) :-                     (19)
  not (holds (must X)),   erase.

holds (may (and X Y)) :-                   (20)
  holds (and X Y).
holds (may (and X Y)) :-                   (21)
  happens E1, happens E2,   erase &
  not (before E1 E2),       erase &
  not (before E2 E1),       erase &
  beforeFact E1 E2 =>
             holds (may (and X Y)).
holds (may (or X Y)) :-                    (22)
  holds (may X); holds (may Y).

holds (may (may X)) :-                     (23)
  holds (may X).

holds (may (must X)) :-                    (24)
  holds (must X).
holds (may (must X)) :-                    (25)
  happens E1, happens E2,   erase &
  not (before E1 E2),       erase &
  not (before E2 E1),       erase &
  beforeFact E1 E2 =>
          holds (may (must X)).
```

Clauses 13, 17, 21 and 25 are particularly interesting since they make use of the implication in the body of clauses. Notice in particular how clauses 11 and 15 implement the alternative formulation of case a) of lemma 7.

The following theorem guarantees that the previous Lolli program is a sound and complete implementation of EC, SKEC and CREC with connectives.

Theorem 9 (Soundness and completeness of EC, SKEC and CREC with connectives w.r.t. the EC-frame semantics)

Let $H = (E, P, [.\rangle, \langle.],].,.[)$ be an EC structure and o a state of knowledge, then

$\tau(H), \tau(o); \emptyset \vdash_{\text{LOLLI}}$ `holds(`$\sigma(\varphi)$`)` iff $\quad o^+ \models \varphi$

Proof

Apply the technique used in the proof of theorem 5 together with lemma 7, property 8 and the statement of theorem 5 itself. ☑

References

[1] I. Cervesato, A. Montanari, A. Provetti: "On the Non-monotonic Behavior of Event Calculus for Deriving Maximal time Intervals", to appear in *The International Journal of Interval Computation*, 1994.

[2] I. Cervesato, L. Chittaro, A. Montanari: "*The Event Calculus as a Modal Theory*", Research Report (in preparation), Dipartimento di Matematica e Informatica, Università di Udine, Italy, 1994.

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

[4] L. Chittaro, A. Montanari, A. Provetti: "Skeptical and Credulous Event Calculi for Supporting Modal Queries", to appear in *Proc. ECAI' 94- 11th European Conference on Artificial Intelligence*, Amsterdam, The Netherlands, August 1994.

[5] J.Y. Girard: "Linear Logic", in *Theoretical Computer Science*, Vol. 50, pp 1-101, North-Holland, Amsterdam, 1987.

[6] J.S. Hodas: Documentation of the Release 0.7 of Lolli, available by anonymous ftp from *ftp.cis.upenn.edu*, directory */pub/Lolli*.

[7] J.S. Hodas, D. Miller: "Logic Programming in a Fragment of Linear Logic", to appear in the *Journal of Information and Computation*.

[8] G. Hughes, M. Cresswell: "*A Companion to Modal Logic*", Methuen, London, 1984.

[9] R. Kowalski, M. Sergot: "A Logic-based Calculus of Events", in *New Generation Computing*, Vol. 4, Ohmsha Ltd and Springer-Verlag, 1986, pp 67-95.

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

[11] D.C. Moffat, G.D. Ritchie: "Modal Queries about Partially-ordered Plans", in *Journal of Expt. Theor. Artificial Intelligence*, Vol. 2, 1990, pp 341-368.

[12] J.W. Lloyd: "*Foundations of Logic Programming*" (second edition), Springer-Verlag, 1987.

[13] D. Miller: "A logic programming language with lambda-abstraction, function variables, and simple unification", in *Proceedings of the International Workshop on Proof-Theoretical Extensions of Logic Programming* (P. Schroeder-Heister Ed.), pp 253-281, Tuebingen, Gemany, 1989, Springer-Verlag LNAI 475.

[14] D. Miller: "A logical analysis of modules in logic programming", in *Journal of Logic Programming*, Vol. 6, 1989, pp 79-108.

[15] D. Miller, G. Nadathur, F. Pfenning, A. Scedrov: "Uniform proofs as a foundation for logic programming", *Annals of Pure and Applied Logic* 51 (1991), pp 125-157, North-Holland.

[16] A. Montanari, E. Maim, E. Ciapessoni, E. Ratto: "Dealing with Time Granularity in the Event Calculus", in *Proc. of FGCS' 92 Conference*, Tokyo, Japan, June 1992, pp 702-712.

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

[18] M.P. Shanahan: "Prediction is Deduction but Explanation is Abduction", in *Proc. of IJCAI' 89- 11th International Joint Conference on Artificial Intelligence*, Detroit, 1989, pp 1055-1050.

[19] M.P. Shanahan: "Representation of Continuous Change in the Event Calculus", in *Proc. of ECAI' 90- 9th European Conference on Artificial Intelligence Conference*, Stockholm, 1990, pp 598-603.

[20] S. Sripada: "A Metalogic Programming Approach to Reasoning about Time in Knowledge Bases", in *Proc. of IJCAI' 93- 13th International Joint Conference on Artificial Intelligence*, Chambery (France), pp 860-865.