A Graph-Theoretic Approach to Efficiently Reason about Partially Ordered Events in (Modal) Event Calculus


Annals of Mathematics and Artificial Intelligence **30:** 93–118, 2000.

A graph-theoretic approach to efficiently reason about partially ordered events in (Modal) Event Calculus

M. Franceschet and A. Montanari

Dipartimento di Matematica e Informatica, Università di Udine, 33100 Udine, Italy
E-mail: {francesc;montana}@dimi.uniud.it

In this paper, we show how well-known graph-theoretic techniques can be successfully exploited to efficiently reason about partially ordered events in Kowalski and Sergot's Event Calculus and in its skeptical and credulous modal variants. To overcome the computational weakness of the traditional generate-and-test algorithm of (Modal) Event Calculus, we propose two alternative graph-traversal algorithms that operate on the underlying directed acyclic graph of events representing ordering information. The first algorithm pairs breadth-first and depth-first visits of such an event graph in a suitable way, while the second one operates on its transitive closure and reduction. We prove the soundness and completeness of both algorithms, and thoroughly analyze and compare their computational complexity.

1. Introduction

The problem of efficiently computing which facts must be or may possibly be true over certain time periods, when only partial information about event ordering is available, is fundamental in a variety of applications, including planning and plan validation [7,10,14]. In this paper, we show how well-known graph-theoretic techniques can be successfully exploited to efficiently reason about partially ordered events in Kowalski and Sergot's Event Calculus [13], EC for short, and in its modal variants (in contrast with the original purely syntactical EC presentation, we adopt a *model-theoretical* description of EC and of its skeptical and credulous modal variants [2–4,6]). Given a set of events, EC is able to infer the largest intervals in which a property holds uninterruptedly (*maximal validity intervals*, MVIs for short). Events can be temporally qualified in several ways. We consider the relevant case where either the occurrence time of an event is totally unspecified or its relative temporal position with respect to (some of) the other events is given. Partial ordering information about events can be naturally represented by means of a directed acyclic graph $G = \langle E, o \rangle$, where the set of nodes E is the set of events and, for every $e_i, e_j \in E$, there exists $(e_i, e_j) \in o$ if and only if it is known that e_i occurs before e_j.

EC updates are of an additive nature only and they just consist in the acquisition of new atomic events and relative information about properties initiated and terminated by them, and/or of further ordering information about the given events [12]. Hence, update processing in EC reduces to the addition of such data, provided that they are consis-

tent and non-redundant with the current stored information. The set of MVIs for any given property p has been traditionally computed at query time according to a simple (and expensive) *generate-and-test* algorithm [2]: EC first blindly picks up every candidate pair of events (e_i, e_j), where e_i and e_j respectively initiate and terminate p; then, it checks whether or not e_i precedes e_j; finally, it looks for possible events e that occur between e_i and e_j and interrupt the validity of p. Checking whether e_i precedes e_j or not reduces to establish if the edge (e_i, e_j) belongs to the transitive closure o^+ of o; checking if there exists an interrupting event e requires to verify if both (e_i, e) and (e, e_j) belong to o^+. Chittaro et al. [8] outline an alternative (and efficient) *graph-traversal* algorithm for MVIs computation when all recorded events are concerned with the same unique property p (*single-property* case). According to such an algorithm, the graph $G = \langle E, o \rangle$ is replaced by its *transitive reduction* $G^- = \langle E, o^- \rangle$, which must be maintained whenever a new consistent and non-redundant pair of events (e_i, e_j) is entered (the addition of a new event e to E does not affect o^-). Since any event $e \in E$ either initiates or terminates p, the set of MVIs for p can be obtained by searching G^- for edges (e_i, e_j) such that e_i initiates p and e_j terminates it. Being G^- the transitive reduction of G ensures us that there are no interrupting events for p that occur between e_i and e_j. It is not difficult to prove that such an algorithm properly works also when all recorded events are concerned with a set of pairwise incompatible properties.

In this paper, we propose two efficient *graph-traversal* algorithms for MVIs computation in the general multiple-property case[1]. The first algorithm represents and maintains temporal information as a binary acyclic relation o and, in order to compute the current set of MVIs, it pairs breadth-first and depth-first visits of the graph $G = \langle E, o \rangle$ in a suitable way. The second algorithm stores and maintains the *transitive closure* $w = o^+$ of a knowledge state, and, for every property p, it stores the *transitive reduction* w_p^- of the subgraph w_p induced by the set of events that are relevant to p. Such an algorithm derives the set of MVIs for any property p by applying the procedure for the single-property case devised in [8] to the transitive reduction w_p^-.

As pointed out in [6], when only partial information about the occurred events and their exact order is available, the sets of MVIs derived by EC bear little relevance, since the acquisition of additional knowledge about the set of events and/or their occurrence times might both dismiss current MVIs and validate new MVIs. Cervesato and Montanari [6] propose a modal variant of EC, called *Modal Event Calculus* (MEC), that allows one to identify the set of MVIs that cannot be invalidated no matter how the ordering information is updated, as far as it remains consistent (*necessary* MVIs), and the set of event pairs that will possibly become MVIs, depending on which ordering data are acquired (*possible* MVIs). They extend the generate-and-test algorithms for MVIs computation in EC to MEC, without any rise in computational complexity. In this paper, we

[1] The generalization to the multiple-property case sketched in [8] is not guaranteed to properly work whenever the transitive reduction of the current knowledge state contains two or more paths between an ordered pair of events that respectively initiate and terminate a given property.

show that the proposed graph-traversal algorithms for MVIs computation in EC can be easily adapted to MEC.

The paper is organized as follows. In section 2, we introduce some basic notions about ordering relations, transitive closure, and transitive reduction. In section 3, we briefly recall syntax and semantics of (Modal) Event Calculus. In sections 4 and 5, we describe the two alternative graph-traversal algorithms for MVIs computation in EC. In section 6 we show how to adapt them to cope with MEC. The increase in efficiency of these algorithms with respect to the traditional generate-and-test one is demonstrated by the complexity analysis of section 7.1 and a comparison between the two algorithms is performed in section 7.2. In the conclusions we provide an assessment of the work done and outline future research directions.

2. On ordering relations, transitive closure and reduction

In this section we recall some basic notions about ordering relations and ordered sets upon which we will rely in the following.

Definition 1 (DAGs, generated DAGs, induced DAGs). Let E be a set and o a binary relation on E. o is called a (strict) *partial order* if it is irreflexive and transitive (and, thus, asymmetric), while it is called a *reflexive partial order* if it is reflexive, antisymmetric, and transitive. The pair $\langle E, o\rangle$ is called a *directed acyclic graph* (*DAG*) if o is a binary acyclic relation; a *strictly ordered set* if o is a partial order; a *non-strictly ordered set* if o is a reflexive partial order. Moreover, given a DAG $G = \langle E, o\rangle$ and a node $e \in E$, the subgraph $G(e)$ of G consisting of all and only the nodes which are accessible from e and of the edges that connect them is called the graph *generated by* e. Finally, given a DAG $G = \langle E, o\rangle$ and a set $T \subseteq E$, the subgraph of G *induced by* T consists of the nodes in T and the subset of edges in o that connect them.

When one is mainly interested in representing the path information of a DAG, two extreme approaches can be followed [16]: (i) *transitive reduction*, or minimum storage representation, and (ii) *transitive closure*, or minimum query-time representation. In this paper, we will make a massive use of the notions of transitive reduction and closure of a DAG. They are formally defined as follows.

Definition 2 (Transitive reduction and closure of DAGs). Let $G = \langle E, o\rangle$ be a DAG. The *transitive reduction* of G is the (unique) graph $G^- = \langle E, o^-\rangle$, with the smallest number of edges, such that, for any pair $e_i, e_j \in E$ there is a directed path from e_i to e_j in G if and only if there is a directed path from e_i to e_j in G^-. The *transitive closure* of G is the (unique) graph $G^+ = \langle E, o^+\rangle$ such that, for any pair of nodes $e_i, e_j \in E$ there is a directed path from e_i to e_j in G if and only if there is an edge $(e_i, e_j) \in o^+$ in G^+.

Aho et al. [1] show that every (directed) graph has a transitive reduction, which can be computed in polynomial time. They also show that such a reduction is unique

in the case of directed acyclic graphs. Furthermore, they prove that the time needed to compute the transitive reduction of a graph differs from the time needed to compute its transitive closure by at most a constant factor.

In the following, we will use the notations $o \uparrow (e_i, e_j)$ and $o \downarrow (e_i, e_j)$ as shorthands for $(o \cup \{(e_i, e_j)\})^+$ and $(o \cup \{(e_i, e_j)\})^-$, respectively. Furthermore, we will denote the sets of all binary acyclic relations and of all partial orders on E as O_E and W_E, respectively. It is easy to show that, for any set E, $W_E \subseteq O_E$. We will use the letters o and w, possibly subscripted, to denote binary acyclic relations and partial orders, respectively. Clearly, if o is a binary acyclic relation, then o^+ is a partial order. We say that two binary acyclic relations $o_i, o_j \in O_E$ are *equally informative* if $o_i^+ = o_j^+$. This induces an equivalence relation $\sim$ on O_E. It is easy to prove that, for any set E, $O_E/\sim$ (the quotient set of O_E with respect to $\sim$) and W_E are isomorphic.

3. Basic and Modal Event Calculus

A compact *model-theoretic* formalization of Kowalski and Sergot's *Event Calculus* has been provided by Cervesato and Montanari in [2]. It distinguishes between the time-independent and time-dependent components of EC. The time-independent component is captured by means of the notion of EC-*structure*.

Definition 3 (EC-structure). A *structure* for the *Event Calculus* (abbreviated *EC-structure*) is a quintuple $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ such that:

- $E = \{e_1, \ldots, e_n\}$ and $P = \{p_1, \ldots, p_m\}$ are finite sets of atomic *events* and *properties*, respectively;
- $[\cdot\rangle : P \to \mathbf{2}^E$ and $\langle\cdot] : P \to \mathbf{2}^E$ are respectively the *initiating* and *terminating map* of $\mathcal{H}$. For every property $p \in P$, $[p\rangle$ and $\langle p]$ represent the set of events that initiate and terminate p, respectively;
- $]\cdot,\cdot[\subseteq P \times P$ is an irreflexive and symmetric relation, called the *exclusivity relation*, that models incompatibility among properties.

The time-dependent component is formalized by specifying a binary acyclic relation o, called *knowledge state*, on the set of events E, which represents our current knowledge about the time ordering between events. EC updates consist in the acquisition of new atomic events and relative information about properties initiated and terminated by them, and/or new ordering information about the given events [12]. Hence, *update processing* in EC reduces to the addition of such data, provided that they are consistent and non-redundant with respect to the already stored information.

Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be a structure for EC and o be a knowledge state. The *query language* $\mathcal{L}$(EC) of EC is the set of property-labeled pairs of events of the form $p(e_1, e_2)$, for every property p in P and events e_1 and e_2 in E. Given a knowledge state o or, equivalently, its transitive closure o^+ of o (recall that path information stored in o and

o^+ is the same), *query processing* in EC reduces to deciding which of the elements of $\mathcal{L}$(EC) are MVIs.

In order for $p(e_1, e_2)$ to be an MVI relative to $w = o^+$, (e_1, e_2) must belong to w. Moreover, e_1 and e_2 must witness the validity of the property p at the ends of this interval by initiating and terminating p, respectively. These requirements are enforced by conditions (i), (ii), and (iii), respectively, in the definition of valuation given below. The maximality requirement is caught by the negation of the meta-predicate $\texttt{broken}(p, e_1, e_2, w)$ in condition (iv), which expresses the fact that the truth of an MVI must not be *broken* by any interrupting event. Any event e which is known to have happened between e_1 and e_2 in w and that initiates or terminates a property that is either p or a property incompatible with p interrupts the truth of $p(e_1, e_2)$.

Definition 4 (Intended model of EC). Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be an EC-structure and $w \in W_E$ be the transitive closure of a knowledge state o. The *intended* EC-*model* of $\mathcal{H}$ is the propositional valuation $\upsilon_{\mathcal{H}} : W_E \to 2^{\mathcal{L}(\mathrm{EC})}$, where $\upsilon_{\mathcal{H}}$ is defined in such a way that $p(e_1, e_2) \in \upsilon_{\mathcal{H}}(w)$ if and only if

(i) $(e_1, e_2) \in w$;

(ii) $e_1 \in [p\rangle$;

(iii) $e_2 \in \langle p]$;

(iv) $\texttt{broken}(p, e_1, e_2, w)$ does not hold, where $\texttt{broken}(p, e_1, e_2, w)$ abbreviates "there exists an event $e \in E$ such that $(e_1, e) \in w$ and $(e, e_2) \in w$, and there exists a property $q \in P$ such that $e \in [q\rangle$ or $e \in \langle q]$, and either $]p, q[$ or $p = q$".

The previous definition adopts the so-called *strong interpretation* of the initiate and terminate relations: given a pair of events e' and e'', with e' occurring before e'', that respectively initiate and terminate a property p, we conclude that p does not hold over (e', e'') if an event e which initiates or terminates p, or a property incompatible with p, occurs during this interval, that is, (e', e'') is a candidate MVI for p, but e forces us to reject it. The strong interpretation is needed when dealing with incomplete sequences of events or incomplete information about their ordering. An alternative interpretation of the initiate and terminate relations, called *weak interpretation*, is also possible. According to such an interpretation, a property p is initiated by an initiating event unless it has been already initiated and not yet terminated (and dually for terminating events). Further details about the strong/weak distinction can be found in [6].

In the case of partially ordered events, the set of MVIs derived by EC is not stable with respect to the acquisition of new ordering information. Indeed, if we extend the current knowledge state with new pairs of events, current MVIs might become invalid and new MVIs can emerge. The *Modal Event Calculus* (MEC) [2] allows one to identify the set of MVIs that cannot be invalidated no matter how the ordering information is updated, as far as it remains consistent, and the set of event pairs that will possibly become MVIs depending on which ordering data are acquired. These two sets are called

necessary MVIs and *possible* MVIs, respectively, using $\Box$-MVIs and $\Diamond$-MVIs as abbreviations. The query language $\mathcal{L}$(MEC) of MEC consists of formulas of the forms $p(e_1, e_2)$, $\Box p(e_1, e_2)$, and $\Diamond p(e_1, e_2)$, for every property p and events e_1 and e_2 defined in $\mathcal{H}$. The intended model of MEC is obtained by shifting the focus from the current knowledge state to all knowledge states that are accessible from it. Since $\subseteq$ is a reflexive partial order, $(W_E, \subseteq)$ can be naturally viewed as a finite, reflexive, transitive, and antisymmetric modal frame. This frame, together with the straightforward modal extension of the valuation $\upsilon_{\mathcal{H}}$ to the transitive closure of an arbitrary knowledge state, provides a modal model for MEC.

Definition 5 (Intended model of MEC). Let $\mathcal{H}$ be an EC-structure and $\upsilon_{\mathcal{H}}$ be the propositional valuation of definition 4. The MEC-*frame* $\mathcal{F}_{\mathcal{H}}$ of $\mathcal{H}$ is the frame $(W_E, \subseteq)$. The *intended* MEC-*model* of $\mathcal{H}$ is the modal model $\mathcal{I}_{\mathcal{H}} = (W_E, \subseteq, \upsilon_{\mathcal{H}})$. Given $w \in W_E$ and $\varphi \in \mathcal{L}$(MEC), the truth of φ at w with respect to $\mathcal{I}_{\mathcal{H}}$, denoted by $\mathcal{I}_{\mathcal{H}}; w \models \varphi$, is defined as follows:

$\mathcal{I}_{\mathcal{H}}; w \models p(e_1, e_2)$	iff	$p(e_1, e_2) \in \upsilon_{\mathcal{H}}(w)$;
$\mathcal{I}_{\mathcal{H}}; w \models \Box p(e_1, e_2)$	iff	for every $w' \in W_E$, $w \subseteq w'$ implies $\mathcal{I}_{\mathcal{H}}; w' \models p(e_1, e_2)$;
$\mathcal{I}_{\mathcal{H}}; w \models \Diamond p(e_1, e_2)$	iff	there is $w' \in W_E$ such that $w \subseteq w'$and $\mathcal{I}_{\mathcal{H}}; w' \models p(e_1, e_2)$.

The sets of MVIs that are necessarily and possibly true in w respectively correspond to the $\Box$- and $\Diamond$-moded atomic formulas which are valid in w. We denote by MVI(w), $\Box$MVI(w), and $\Diamond$MVI(w) the sets of MVIs, necessary MVIs, and possible MVIs with respect to w, respectively. Computing necessary and possible MVIs is relevant to many applications. As an example, verifying whether a particular subgoal of a given goal is necessarily achieved is a central task in planning and plan validation applications [7,10,14]. Hence, the efficiency of procedures for computing necessary and possible conditions is crucial. Cervesato and Montanari [2] show that the sets of $\Box$- and $\Diamond$-MVIs can be computed by exploiting *local conditions* over w, thus avoiding a complete and expensive search of all the consistent refinements of w. More precisely, a property p necessarily holds between two events e_1 and e_2 if and only if the interval (e_1, e_2) belongs to the current order, e_1 initiates p, e_2 terminates p, and no event that either initiates or terminates p (or a property incompatible with p) will ever be consistently located between e_1 and e_2. The last requirement is caught by the meta-predicate `posBroken`(p, e_1, e_2, w) of proposition 6. Similarly, a property p may possibly hold between e_1 and e_2 if and only if the interval (e_1, e_2) is consistent with the current ordering, e_1 initiates p, e_2 terminates p, and there are no already known interrupting events between e_1 and e_2. Local conditions are formally captured by the following proposition [2].

Proposition 6 (Local conditions). Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be a EC-structure. For any atomic formula $p(e_1, e_2)$ on $\mathcal{H}$ and any $w \in W_E$,

- $\mathcal{I}_{\mathcal{H}}; w \models \Box p(e_1, e_2)$ if and only if
  - (i) $(e_1, e_2) \in w$;
  - (ii) $e_1 \in [p\rangle$;
  - (iii) $e_2 \in \langle p]$;
  - (iv) $\texttt{posBroken}(p, e_1, e_2, w)$ does not hold, where $\texttt{posBroken}(p, e_1, e_2, w)$ abbreviates "there exists an event $e \in E$ such that $(e, e_1) \notin w$, $e \neq e_1$, $(e_2, e) \notin w$, $e \neq e_2$, and there exists a property $q \in P$ such that $e \in [q\rangle$ or $e \in \langle q]$, and either $]p, q[$ or $p = q$".
- $\mathcal{I}_{\mathcal{H}}; w \models \Diamond p(e_1, e_2)$ if and only if
  - (i) $(e_2, e_1) \notin w$;
  - (ii) $e_1 \in [p\rangle$;
  - (iii) $e_2 \in \langle p]$;
  - (iv) $\texttt{broken}(p, e_1, e_2, w)$ does not hold.

Local conditions above resemble Chapman's *modal truth criterion* [7] and Nebel's and Bäckström's definition of the predicate *Maybe* [14]. Proposition 6 also allows us to give an alternative definition of the sets $\Box$MVI(w) and $\Diamond$MVI(w). Given $w \in W_E$ and $p \in P$, let $S(w)$ be the set of atomic formulas $p(e_1, e_2)$ such that all other events in E that initiate or terminate p, or a property incompatible with p, are ordered with respect to e_1 and e_2 in w, and let $C(w)$ be the set of atomic formulas $p(e_1, e_2)$ such that e_1 initiates p, e_2 terminates p, and e_1 and e_2 are unordered in w. Formally,

$$\begin{aligned} S(w) = \big\{ & p(e_1, e_2) \mid \forall e \in E((\exists q \in P((e \in [q\rangle \vee e \in \langle q]) \wedge (q = p \vee]p, q[))) \\ & \rightarrow (((e, e_1) \in w \vee (e_1, e) \in w) \wedge ((e, e_2) \in w \vee (e_2, e) \in w)))\big\}, \\ C(w) = \big\{ & p(e_1, e_2) \mid e_1 \in [p\rangle \wedge e_2 \in \langle p] \wedge (e_1, e_2) \notin w \wedge (e_2, e_1) \notin w\big\}. \end{aligned}$$

The set $\Box$MVI(w) (respectively $\Diamond$MVI(w)) can be alternatively defined as the intersection (respectively union) of the sets MVI(w) and $S(w)$ (respectively $C(w)$), as stated by the following corollary.

Corollary 7. Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be an EC-structure and $w \in W_E$ be a partial order. It holds that $\Box\text{MVI}(w) = \text{MVI}(w) \cap S(w)$ and $\Diamond\text{MVI}(w) = \text{MVI}(w) \cup C(w)$.

Proof. We prove that $\Diamond\text{MVI}(w) = \text{MVI}(w) \cup C(w)$. The proof for necessary MVIs is similar, and thus omitted. We first prove the $\subseteq$ inclusion. Let $p(e_1, e_2) \in \Diamond\text{MVI}(w)$. If $(e_1, e_2) \in w$, then $p(e_1, e_2) \in \text{MVI}(w)$, and thus the thesis. Hence, suppose that $(e_1, e_2) \notin w$. Since $p(e_1, e_2) \in \Diamond\text{MVI}(w)$, we have that $e_1 \in [p\rangle$, $e_2 \in \langle p]$, and $(e_2, e_1) \notin w$. Hence $p(e_1, e_2) \in C(w)$. We now prove the $\supseteq$ inclusion. Let $p(e_1, e_2) \in \text{MVI}(w) \cup C(w)$. If $p(e_1, e_2) \in \text{MVI}(w)$, then $p(e_1, e_2) \in \Diamond\text{MVI}(w)$. Now, suppose $p(e_1, e_2) \in C(w)$. It follows that $e_1 \in [p\rangle$, $e_2 \in \langle p]$, and e_1 and e_2 are unordered in w. To obtain the thesis, it remains to prove that $\texttt{broken}(p, e_1, e_2, w)$ does

not holds. By contradiction, suppose broken(p, e_1, e_2, w). This means that there exists an interrupting event e such that $(e_1, e) \in w$ and $(e, e_2) \in w$. Since w is transitive, it follows that $(e_1, e_2) \in w$, which is a contradiction, since e_1 and e_2 are unordered in w. □

In section 6, we will exploit corollary 7 to devise an algorithm for MVIs computation in MEC. Furthermore, from corollary 7 it is immediate to conclude that the sets of necessary MVIs, MVIs, and possible MVIs, with respect to the current state of knowledge, form an inclusion chain, as formally stated by the following proposition.

Proposition 8 (Necessary MVIs and possible MVIs enclose MVIs). Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be an EC-structure and $w \in W_E$ be a partial order. It holds that

$$\Box\text{MVI}(w) \subseteq \text{MVI}(w) \subseteq \Diamond\text{MVI}(w).$$

Notice that if w is a total order, then $S(w) = \mathcal{L}(\text{EC})$ and $C(w) = \emptyset$, and thus $\Box\text{MVI}(w) = \Diamond\text{MVI}(w) = \text{MVI}(w)$.

In sections 4 and 5, we will propose two graph-traversal algorithms for computing MVIs in EC and MEC. For each of them, we will describe (i) the form in which the algorithm stores the ordering information, (ii) the *update processing*, that is, the computation steps executed by the algorithm to update the current knowledge state with a new pair of events (*update time*), and (iii) the *query processing*, that is, the computation steps executed by the algorithm to determine the current set of MVIs (*query time*). We will not directly consider the case of updates consisting of the addition of single atomic events, but we will briefly explain how to extend the algorithms to cope with such a case.

4. A first graph-traversal algorithm for MVIs computation

In this section, we describe a first *graph-traversal* algorithm that computes the set of MVIs by pairing a depth-first and a breadth-first visit of the event graph representing ordering information. We provide a high-level description of the algorithm and prove its soundness and completeness with respect to the semantics of EC.

The algorithm stores ordering information in the form of an acyclic binary relation o. The addition of a new pair of events (e_1, e_2) to o is dealt with as follows (update processing). First, (e_1, e_2) is checked for consistency and non redundancy with respect to o. If (e_1, e_2) is consistent ($(e_2, e_1) \notin o^+$) and non-redundant ($(e_1, e_2) \notin o^+$), then (e_1, e_2) is added to o. Testing whether a pair of events (e', e'') is in o^+ or not can be performed by visiting depth-first the subgraph of (E, o) generated by e', searching for the node e''.

Query processing is more involved. Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be an EC-structure and $o \in O_E$ be an acyclic binary relation. We define an algorithm for MVIs computation that combines a breadth-first and a depth-first visit of the graph (E, o), which is directed and acyclic, but not necessarily connected (background knowledge on elementary graph algorithms can be found in [9]). In the following, whenever it does

not lead to ambiguities, we denote the graph (E, o) by G and the subgraph of (E, o) generated by e (cf. definition 1) by $G(e)$.

The algorithm behaves as follows: for every property $p \in P$ and every event $e_1 \in E$ initiating p, it searches the graph $G(e_1)$ for all events e_2 such that the interval (e_1, e_2) is an MVI for p. Given a property p and an event e_1, the algorithm associates the following labels with the nodes of $G(e_1)$:

- `unmarked`: it denotes nodes (events) to be visited;
- `visited`: it denotes nodes (events) already visited;
- `marked`: it denotes nodes (events) that initiate or terminate either p or a property incompatible with p;
- `cutoff`: it denotes nodes (events) which are cut off from the search space, because they cannot terminate any MVI for p initiated by e_1.

The set of events e_2 such that $p(e_1, e_2)$ is an MVI is computed as follows. Initially, all nodes in $G(e_1)$ are labeled with `unmarked`; then, the graph $G(e_1)$ is visited breadth-first. The breadth-first visit of $G(e_1)$ starts from the successors of e_1 (first layer) and proceeds, layer by layer, until the last layer is reached. The last layer is a layer followed by an empty layer; since $G(e_1)$ is acyclic, such a layer always exists and it is unique. At each layer, only `unmarked` events are processed. Let e be an `unmarked` event belonging to the current layer. The algorithm labels e as `visited` and checks whether or not it initiates or terminates either p or a property incompatible with p. If the outcome of the test is positive, then the following operations are executed before processing the next event in the layer:

1. e is labeled as `marked`;
2. the label `cutoff` is assigned to all nodes of $G(e)$ different from e;
3. if e terminates p, then the node e is saved.

Once the whole graph $G(e_1)$ has been visited, all the saved nodes, which are still labeled as `marked`, are returned; they are all and only the events that terminate an MVI for p initiated by e_1.

A pseudo-code description of such an algorithm for MVIs computation can be given as follows.

```
MVI ← ∅
for each p ∈ P do
    for each e1 ∈ [p⟩ do
        S ← ∅
        for each e ∈ G(e1) do
            set(e, unmarked)
        L ← nextlayer({e1})
        while L ≠ ∅ do
                for each e ∈ L do
```

```
            if is_relevant_to(e,p) then
              set(e, marked)
              cutoff(e)
              if e ∈ ⟨p] then
                S ← S ∪ {e}
        L ← nextlayer(L)
    for each e2 ∈ S do
      if label(e2, marked) then
        MVI← MVI ∪{p(e1, e2)}
return MVI
```

The procedure set(e, l) assigns a unique label l to the event e, possibly overwriting the previous one. The boolean function label(e, l) checks whether label l is associated with the event e or not, and the boolean function is_relevant_to(e, p) tests whether or not e initiates or terminates either p or a property incompatible with p. The procedure nextlayer(L) computes the next layer in the breadth-first visit of the graph $G(e_1)$:

```
nextlayer(L)
    L' ← ∅
    for each e ∈ L do
      if label(e, unmarked) or label(e, visited) then
        for each successor e' of e do
          if label(e', unmarked) then
            set(e', visited)
            L' ← L' ∪ {e'}
    return L'
```

Finally, the procedure cutoff(e) visits depth-first the subgraph generated by the event e and labels as cutoff all its nodes:

```
cutoff(e)
    for each successor e' of e do
      if not label(e', cutoff) then
        set(e', cutoff)
        cutoff(e')
```

Before proving that such an algorithm in sound and complete with respect to the semantics of EC, we illustrate its behaviour by means of two simple examples. Let e_1, e_2, and e_3 be three event occurrences, p and q be two incompatible properties, and $o = \{(e_1, e_2), (e_1, e_3), (e_2, e_3)\}$ be the current knowledge state depicted in figure 1, left side. It is worth noticing that the event graph of the example contains a transitive edge $(e_1 \rightarrow e_3)$. Suppose that e_1 initiates p, e_2 initiates q, and e_3 terminates p. The set of MVIs for p, which are initiated by e_1, is computed as follows. The algorithm first labels as unmarked all nodes of $G(e_1)$, and then it visits breadth-first $G(e_1)$. The first layer

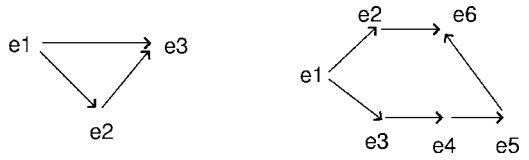

Figure 1. Two graphs representing two ordering relations.

contains both e_2 and e_3. Suppose that the algorithm first processes e_3 and then e_2. The node e_3 is labeled as `marked` and saved, because it terminates p. The propagation of the label `cutoff` has no effect, since e_3 has no successors. Hence, the node e_2 is processed and labeled as `marked`, because it initiates a property q which is incompatible with p. The effect of propagating the label `cutoff` is that of replacing the label `marked` of e_3 by the label `cutoff`. Then, the visit of $G(e_1)$ terminates (all nodes have already been visited) and the algorithm returns no MVIs for p initiated by e_1, because the label associated with e_2 (the only saved event) is `cutoff` and not `marked`.

The above example clarifies the role of the label `cutoff`: some events may be labeled as `marked` along a "short" path ($e_1 \rightarrow e_3$ in the example) and saved as candidate ending points of an MVI for the considered property. However, an interval is an MVI for a property if and only if *all* paths leading from the initiating event to the terminating one do not contain interrupting events, that is, events that initiate or terminate either the considered property or a property incompatible with it. If there exists a longer path ($e_1 \rightarrow e_2 \rightarrow e_3$ in the example) which contains an interrupting event, then the candidate node is cut off during the propagation of the label `cutoff`.

The next example shows that `cutoff` labels are needed also for reasoning about event graphs devoid of transitive edges, that is, even in the case in which we store and maintain the transitive reduction of the current knowledge state the `cutoff` procedure is necessary. Consider a scenario consisting of six event occurrences e_1, e_2, e_3, e_4, e_5, and e_6, two incompatible properties p and q, and the knowledge state o depicted in figure 1, right side, which has no transitive edges. Suppose that e_1 initiates p, e_5 initiates q, e_6 terminates p, and e_2, e_3, and e_4 affect neither p nor a property incompatible with p. The interval (e_1, e_6) is not an MVI for p, because there exists an interrupting event, namely e_5, which occurs between e_1 and e_6. The algorithm removes the node e_6 from the set of candidate terminating events associated with initiating event e_1 when it propagates the label `cutoff` during the processing of e_5.

The following theorem proves that the proposed algorithm computes exactly the set of MVIs as defined in definition 4.

Theorem 9. The proposed graph-traversal algorithm is sound and complete.

Proof. By definition, $p(e_1, e_2)$ is an MVI with respect to the current knowledge state if and only if e_1 initiates p, e_2 terminates p, e_2 belongs to $G(e_1)$, and every path $e_1 \rightsquigarrow e_2$ from e_1 to e_2 in $G(e_1)$ does not contain events relevant to p, that is, events that affect

(initiate or terminate) either p or a property incompatible with p and differ from both e_1 and e_2.

We first prove that the algorithm is sound, that is, if $p(e_1, e_2)$ is generated by the algorithm, then $p(e_1, e_2)$ is an MVI. Given a property p and an event e_1, the algorithm searches the acyclic graph $G(e_1)$ for terminating events e_2. Since the visit is breadth-first, each node is reached along the shortest path on $G(e_1)$ starting from e_1. Given a node e, we denote by $D(e)$ the length of the shortest path on $G(e_1)$ connecting e_1 to e.

We proceed by contradiction. Suppose that $p(e_1, e_2)$ is returned by the algorithm, but it is not an MVI. If e_1 does not initiate p or e_2 does not terminate p, then $p(e_1, e_2)$ cannot be retrieved. Moreover, if e_2 does not belong to $G(e_1)$, then the visit of $G(e_1)$ does not retrieve e_2, and hence $p(e_1, e_2)$ cannot be generated. Finally, suppose that there exists at least one path $e_1 \rightsquigarrow e_2$ in $G(e_1)$ that contains at least one node z which affects either p or a property incompatible with p and is different from e_1 and e_2. If $D(z) < D(e_2)$, then the node z is visited before e_1, it is labeled as `marked`, and the label `cutoff` is propagated to the nodes of $G(z)$ different from z. In particular, e_2 is labeled as `cutoff` during such a propagation and thus it cannot be chosen as the terminating event of an MVI for p initiated by e_1. Hence, $p(e_1, e_2)$ cannot be generated by the algorithm. If $D(z) > D(e_2)$ (notice that $D(z) \neq D(e_2)$, since $z \neq e_2$ and there are not simultaneous events), then the node e_2 is visited before z, it is labeled as `marked`, and the label `cutoff` is propagated to the nodes of $G(e_2)$ different from e_2. Since the graph $G(e_1)$ is acyclic and there exists a path from z to e_2, there are no paths from e_2 to z; hence the propagation of the label `cutoff` does not reach the node z. The node z is processed at some later stage, it is labeled as `marked`, and the label `cutoff` is propagated to the nodes of $G(z)$ different from z. In particular, the label of e_2 is changed from `marked` to `cutoff`, and thus $p(e_1, e_2)$ cannot be generated by the algorithm.

We now prove that the algorithm is complete, that is, if $p(e_1, e_2)$ in an MVI, then $p(e_1, e_2)$ is generated by the algorithm. Since (e_1, e_2) is an interval, e_2 is reachable from e_1 in the graph $G(e_1)$. Since $p(e_1, e_2)$ is an MVI, every path $e_1 \rightsquigarrow e_2$ from e_1 to e_2 in $G(e_1)$ does not contain interrupting events for p different from e_1 and e_2. Hence, the node e_2 is not cut off and, since it terminates p, it is labeled as `marked` and retrieved as the terminating event of an MVI for p initiated by e_1. Thus, $p(e_1, e_2)$ is generated by the algorithm. □

Notice that the proposed algorithm is based on a *forward* strategy: given a property p and an initiating event e_1, it visits the graph $G(e_1)$, looking for a terminating event e_2 such that $p(e_1, e_2)$ is an MVI. Nothing prevents us to define an equivalent *backward* algorithm as follows. Given a directed graph G, let us denote by $\widehat{G}$ the converse of G, i.e., the graph in which each edge (e_i, e_j) of G has been replaced by the edge (e_j, e_i). Given a property p and a terminating event e_2, we visit the graph $\widehat{G}(e_2)$ as before, looking for initiating events e_1 such that $p(e_1, e_2)$ is an MVI.

5. A second graph-traversal algorithm for MVIs computation

In this section, we first propose a sound (respectively complete) graph-traversal algorithm for MVIs computation in EC that exploits the notion of transitive reduction (respectively closure) of the ordering graph; then, we show how to pair transitive reduction and closure to devise a sound and complete algorithm.

5.1. Two partial graph-traversal algorithms

We start by describing a sound (but incomplete) and a complete (but unsound) graph-traversal algorithm for MVIs computation in EC. The first algorithm stores and maintains the transitive reduction of a knowledge state. Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be an EC-structure, o^- be the transitive reduction of a knowledge state o, and (e_1, e_2) be an ordered pair of events. The addition of (e_1, e_2) to o^- is dealt with as follows (update processing). First, (e_1, e_2) is checked for consistency and redundancy with respect to o^-. If (e_1, e_2) is neither inconsistent ($(e_2, e_1) \notin o^+$) nor redundant ($(e_1, e_2) \notin o^+$), then o^- is replaced by $o^- \downarrow (e_1, e_2)$. The set $o^- \downarrow (e_1, e_2)$, which can be proved to be the transitive reduction of $o \cup \{(e_1, e_2)\}$, is obtained as follows: first, the ordered pair (e_1, e_2) is added to o^-; then, the set of nodes from which e_1 is accessible ($\texttt{Pred}(e_1)$) and the set of nodes which are accessible from e_2 ($\texttt{Succ}(e_2)$) are computed; finally, any pair $(e', e'') \in o^-$ such that $e' \in \texttt{Pred}(e_1)$ and $e'' \in \texttt{Succ}(e_2)$ is deleted from $o^- \cup \{(e_1, e_2)\}$.

if $(e_1, e_2) \notin o^+$ **and** $(e_2, e_1) \notin o^+$ **then**
  $o^- \leftarrow o^- \cup \{(e_1, e_2)\}$
  put in $\texttt{Pred}(e_1)$ the nodes from which e_1 is accessible
  put in $\texttt{Succ}(e_2)$ the nodes accessible from e_2
  for each $e' \in \texttt{Pred}(e_1)$ **do**
    for each $e'' \in \texttt{Succ}(e_2)$ **do**
      if $(e', e'') \in o^-$ **then** $o^- \leftarrow o^- \setminus \{(e', e'')\}$

Given two events e' and e'', testing whether $(e', e'') \in o^+$ or not can be performed by visiting depth-first the subgraph of (E, o^-) generated by e', searching for the node e''. The set $\texttt{Succ}(e_2)$ can be computed by executing a depth-first visit of the subgraph of $(E, o^- \downarrow (e_1, e_2))$ generated by e_2 and retrieving all the visited nodes. Similarly, in order to compute the set $\texttt{Pred}(e_1)$, we visit depth-first the subgraph generated by e_1 with respect to the converse graph $\widehat{G}$ of the graph $G = o^- \downarrow (e_1, e_2)$ (the notion of converse graph has been given in the previous section).

At query time, for every property p, the algorithm selects as MVIs for p the *p-edges* of the transitive reduction o^-, i.e. the edges $(e_1, e_2) \in o^-$ such that e_1 initiates p and e_2 terminates p.

MVI$\leftarrow \emptyset$
for each $p \in P$ **do**
  for each $(e_1, e_2) \in o^-$ **do**
    if $e_1 \in [p\rangle$ **and** $e_2 \in \langle p]$ **then** MVI $\leftarrow$ MVI $\cup \{p(e_1, e_2)\}$
return MVI

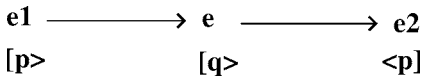

Figure 2. An example of incompleteness.

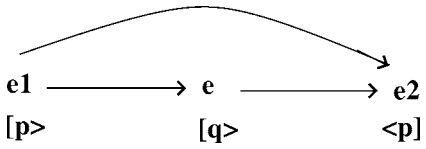

Figure 3. An example of unsoundness.

Such an algorithm is *sound*: (e_1, e_2) is selected as an MVI for p if e_1 initiates p, e_2 terminates p, e_1 precedes e_2, and there are no events between e_1 and e_2 (the truth of this last condition immediately follows from the fact that (e_1, e_2) is an edge of the transitive reduction o^-); hence, by definition, $p(e_1, e_2)$ is an MVI. As shown in [8], this algorithm is also complete in the single-property case. Unfortunately, it is *incomplete* in the general multiple-property case. Consider the simple scenario depicted in figure 2, where p and q are two compatible properties. Since e does not interrupt the validity of p, $p(e_1, e_2)$ is an MVI for p; however, since there is not an edge from e_1 to e_2, the proposed algorithm does not return $p(e_1, e_2)$.

Let us now describe a complete (but not sound) graph-traversal algorithm for MVIs computation. The idea is to store and maintain the transitive closure of the knowledge state, instead of its transitive reduction. Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be an EC-structure, $w \in W_E$ be the transitive closure of a knowledge state o, and (e_1, e_2) be an ordered pair of events. The addition of (e_1, e_2) to w is dealt with as follows (update processing). Whenever both $(e_1, e_2) \notin w$ and $(e_2, e_1) \notin w$, the update procedure determines $w \uparrow (e_1, e_2)$ by executing the following steps: first, the edge (e_1, e_2) is added to w; then, for every pair of events $e', e'' \in E$ such that $(e', e_1) \in w$, $(e_2, e'') \in w$, and $(e', e'') \notin w$, the edge (e', e'') is added to $w \cup (e_1, e_2)$. It is worth noting that, since w is transitive, the set of predecessors (respectively successors) of e_1 (respectively e_2) coincides with the set of nodes from which e_1 is accessible (respectively accessible from e_2).

if $(e_1, e_2) \notin w$ **and** $(e_2, e_1) \notin w$ **then**
  $w \leftarrow w \cup \{(e_1, e_2)\}$
  for each predecessor e' of e_1 **do**
    for each successor e'' of e_2 **do**
      if $(e', e'') \notin w$ **then** $w \leftarrow w \cup \{(e', e'')\}$

At query time, for every property p, the algorithm retrieves as MVIs for p the p-edges of w as before.

The proposed algorithm is *complete*: if $p(e_1, e_2)$ is an MVI, then, by definition, e_1 initiates p, e_2 terminates p, and e_1 precedes e_2; hence, the interval (e_1, e_2) is selected as an MVI for the property p. Unfortunately, it is immediate to show that such an algorithm is *not sound*. Consider the scenario of figure 3, where p and q are incompatible

properties. The interval $p(e_1, e_2)$ is not an MVI for p, since e interrupts the validity of p over (e_1, e_2). However, since there is an edge from e_1 to e_2, $p(e_1, e_2)$ is retrieved as an MVI for p.

5.2. *Pairing transitive closure and reduction*

In this section, we pair the notions of transitive closure and reduction to obtain a sound and complete graph-traversal algorithm for MVIs computation in EC. The algorithm stores and maintains the *transitive closure* w of a knowledge state and, for every property p, it stores the *transitive reduction* of the subgraph w_p induced by the set of events that are relevant to p. Let $\mathcal{H}$ be an EC-structure, w be the transitive closure of a knowledge state o, and (e_1, e_2) be an ordered pair of events. The addition of (e_1, e_2) to w is dealt with as follows (update processing):

1. if $(e_1, e_2) \notin w$ and $(e_2, e_1) \notin w$, then w is replaced by $w \uparrow (e_1, e_2)$;
2. for every property $p \in P$, the subgraph w_p induced by the set of events that are relevant to p, that is, the events that initiate or terminate either p or a property incompatible with p, is extracted from $w \uparrow (e_1, e_2)$;
3. for every property $p \in P$, the transitive reduction w_p^- of the graph w_p is computed by using one of the standard algorithms, e.g., [11,15].

```
// computing w ↑ (e1, e2)
if (e1, e2) ∉ w and (e2, e1) ∉ w then
  w ← w ∪ {(e1, e2)}
  for each predecessor e' of e1 do
          for each successor e'' of e2 do
                  if (e', e'') ∉ w then w ← w ∪ {(e', e'')}

// computing w_p for every property p
for each p ∈ P do
        w_p ← ∅
        for each (e', e'') ∈ w do
            if is_relevant_to(e',p) and is_relevant_to(e'',p)
            then w_p ← w_p ∪ {(e', e'')}

// computing w_p^- for every property p
for each p ∈ P do
        compute the transitive reduction w_p^-
```

Notice that computing the transitive closure of the knowledge state (item 1 above) is necessary since neither the transitive reduction of the subgraph induced by p, nor the subgraph of the transitive reduction induced by p would work, as shown by the example depicted in figure 2.

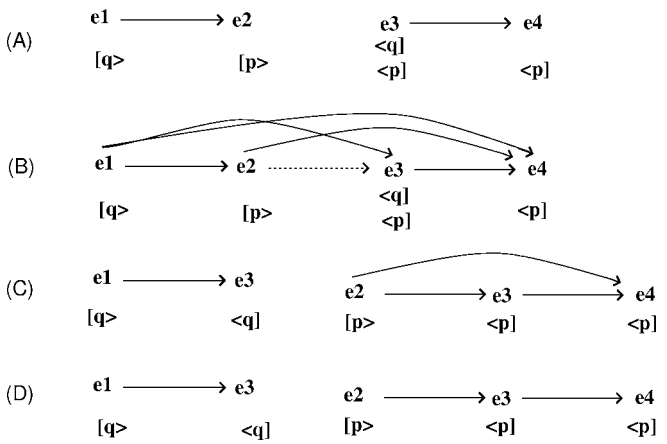

Figure 4. MVIs computation using the proposed graph-traversal algorithm.

The set of MVIs for p includes all and only the p-edges of w_p^-. Hence, for every property p, query processing reduces to the retrieval of the p-edges of w_p^-.

MVI$\leftarrow \emptyset$
for each $p \in P$ **do**
  for each $(e_1, e_2) \in w_p^-$ **do**
    if $e_1 \in [p\rangle$ **and** $e_2 \in \langle p]$ **then** MVI $\leftarrow$ MVI $\cup \{p(e_1, e_2)\}$
return MVI

An example of MVIs computation using the proposed graph-traversal algorithm is given in figure 4. Let the initial situation be that depicted in figure 4(A), where p and q are two compatible properties. Once the edge (e_2, e_3) is entered, the transitive closure of the resulting graph is computed (cf. figure 4(B)). Then, the subgraphs induced by the events that respectively are relevant to q and p are extracted from the original graph (the two resulting subgraphs are described in figure 4(C)). Finally, the transitive reductions w_q^- and w_p^- are computed (cf. figure 4(D)). At query time, $q(e_1, e_3)$ and $p(e_2, e_3)$ are returned as MVIs for q and p, respectively.

Theorem 10. The proposed graph-traversal algorithm for MVIs computation is sound and complete with respect to the given semantics of EC.

Proof. Let $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ be an EC-structure and w be the transitive closure of a knowledge state o. To prove that the proposed algorithm is sound, we must show that if (e_1, e_2) is a p-edge of w_p^-, then $p(e_1, e_2)$ is an MVI for p with respect to w and $\mathcal{H}$. The proof is by contradiction. If $p(e_1, e_2)$ is not an MVI, then one of the following propositions must hold: e_1 does not initiate p, e_2 does not terminate p, $(e_1, e_2) \notin w$, or there exists an interrupting event e for p that occurs between e_1 and e_2. If e_1 does not initiate p or e_2 does not terminate p, then (e_1, e_2) is not a p-edge. If $(e_1, e_2) \notin w$, then $(e_1, e_2) \notin w_p^-$, since $w_p^- \subseteq w$, and thus (e_1, e_2) is not a p-edge of w_p^-. Finally, if there

exists an interrupting event e for p such that both $(e_1, e) \in w$ and $(e, e_2) \in w$, then there exist a path $e_1 \rightsquigarrow e$ and a path $e \rightsquigarrow e_2$ in w_p^-. Hence, the edge (e_1, e_2) is a transitive one, and thus it does not belong to w_p^-. This allows us to conclude that (e_1, e_2) is not a p-edge of w_p^-.

To prove that the proposed algorithm is complete, we must show that if $p(e_1, e_2)$ is an MVI for p with respect to w and $\mathcal{H}$, then (e_1, e_2) is a p-edge of w_p^-. By hypothesis, e_1 initiates p, e_2 terminates p, and $(e_1, e_2) \in w$. It follows that (e_1, e_2) is a p-edge of w_p. Moreover, since there are no interrupting events for p that occur between e_1 and e_2, the edge (e_1, e_2) is the unique path from e_1 to e_2 in w_p. This implies that the edge (e_1, e_2) is not transitive, and thus it is a p-edge of w_p^-. □

6. Adapting the algorithms to the Modal Event Calculus

Two efficient algorithms, that respectively compute necessary and possible MVIs of MEC, can be obtained from corollary 7 taking advantage of the algorithms for MVIs computation in EC described in sections 4 and 5. Let o be a knowledge state and $w = o^+$ be its transitive closure. In order to compute the sets $C(w)$ and $S(w)$ (cf. section 3), we proceed as follows. $C(w)$ is obtained by selecting all property-labeled pairs of events $p(e_1, e_2)$ such that e_1 initiates p, e_2 terminates p, and e_1 and e_2 are unordered in w:

compute $w = o^+$
sort w
$C \leftarrow \emptyset$
for each $p \in P$ **do**
  for each $(e_1, e_2) \in E \times E$ **do**
    if $e_1 \in [p\rangle$ **and** $e_2 \in \langle p]$ **and**
      $(e_1, e_2) \notin w$ **and** $(e_2, e_1) \notin w$ **then**
      $C \leftarrow C \cup \{p(e_1, e_2)\}$
return C

The computation of $S(w)$ is more involved. First, we compute the set $U(w)$ containing all pairs $(e, p) \in E \times P$ such that there exists another event e', which affects either p or a property incompatible with p and is unordered with respect to e in w. It is easy to see that if $(e, p) \in U(w)$, then e neither initiates nor terminates a $\Box$-MVI for p. The set $S(w)$ is obtained by selecting those atomic formulas $p(e_1, e_2)$ such that neither (e_1, p) nor (e_2, p) belong to $U(w)$:

// compute $U(w)$
compute $w = o^+$
sort w
$U \leftarrow \emptyset$
$S \leftarrow \emptyset$
for each $p \in P$ **do**
  for each $e \in E$ **do**

```
        Found ← False
        V ← E
        while not Found and V ≠ ∅ do
              let e′ ∈ V
              if (e, e′) ∉ w and
                 (e′, e) ∉ w and
                 is_relevant_to(e′, p) then
                 Found ← True
                 U ← U ∪ {(e, p)}
              else
                 V ← V \ {e′}
// compute S(w) taking advantage of U(w)
for each p ∈ P do
    for each (e1, e2) ∈ E × E do
        if (e1, p) ∉ U and
           (e2, p) ∉ U then
           S ← S ∪ {p(e1, e2)}
return S
```

Let us first consider the algorithm for MVIs computation in MEC obtained by exploiting the procedure for MVIs computation in EC described in section 4. Update processing in MEC consists in checking whether the new pair of events is consistent and non-redundant with respect to the current knowledge state, as in EC. As for query processing, the set MVI(w) is obtained as shown in section 4 and, in addition, it is sorted. The sets $C(w)$ and $S(w)$ are computed by the above described algorithms. The set $\Box$MVI(w) is obtained by intersecting MVI(w) and $S(w)$, while the set $\Diamond$MVI(w) is computed by taking the union of MVI(w) and $C(w)$.

We now describe the algorithm for MVIs computation in MEC obtained by using the procedure for MVIs computation in EC described in section 5. Update processing first determines $w \uparrow (e_1, e_2)$ and then, for every property p, derives w_p^- as shown in section 5. In addition, w_p^- is sorted. The sets $C(w)$ and $S(w)$ are obtained as in the previous case (notice that in such a case the computation of the transitive closure of the current knowledge state is actually not necessary). Query processing reduces to the computation of the sets MVI(w), by means of the procedure for MVIs computation in EC shown in section 5, $\Box$MVI(w), by intersecting MVI(w) and $S(w)$, and $\Diamond$MVI(w), by taking the union of MVI(w) and $C(w)$.

It is worth noting that the proposed algorithms can be easily extended to cope with updates consisting of the addition of new events. It is easy to see that the addition of a new event to the structure does not affect the set of MVIs derived by EC. On the contrary, when a new event is added, the sets $C(w)$ and $S(w)$ grows and shrinks, respectively. It follows that, in order to cope with such updates, it is sufficient to recompute the sets $C(w)$ and $S(w)$, while the set MVI(w) remains unchanged.

7. Complexity analysis

In this section, we first analyze and then compare the computational complexity of the proposed algorithms for MVIs computation.

Given an EC-structure $\mathcal{H} = (E, P, [\cdot\rangle, \langle\cdot],]\cdot,\cdot[)$ and an acyclic binary relation $o \in O_E$, we determine the complexity of computing the set of MVIs with respect to o and $\mathcal{H}$, i.e., the set of formulas $p(e_1, e_2)$ (respectively $\Box p(e_1, e_2)$ and $\Diamond p(e_1, e_2)$) such that $o^+ \models p(e_1, e_2)$ (respectively $o^+ \models \Box p(e_1, e_2)$ and $o^+ \models \Diamond p(e_1, e_2)$), by using the proposed algorithms. We measure the complexity in terms of the size n of the structure $\mathcal{H}$ (where n is the number of recorded events in E) and the size m of the relation o, or the size m^- of its transitive reduction o^-. This choice can be explained as follows. Given an EC-structure $\mathcal{H}$, the set E of events can be arbitrarily large, while the set P of properties is fixed once and for all, since it is an invariant characteristic of the considered domain. Since the cardinality of P does not change from one problem instance to another one (unless we change the application domain), while the cardinality of E may grow arbitrarily, we choose the cardinality of E as the size of $\mathcal{H}$ and consider the number of properties as a constant. Moreover, when analyzing the complexity of graph algorithms, it is standard to consider the number m of edges as a relevant complexity parameter, albeit it is known that $m = \mathcal{O}(n^2)$.

7.1. The complexity of query and update processing

We assume P and E to be sorted. Furthermore, we assume that the knowledge state o as well as the sets $[p\rangle$ and $\langle p]$, for every property $p \in P$, are maintained sorted. Under such assumptions, given an event e and a property p, the tests $e \in [p\rangle$ and $e \in \langle p]$ can be performed, by using a binary search, in time $\mathcal{O}(\log n)$. Similarly, given two distinct events e_1 and e_2, the test $(e_1, e_2) \in o$ costs $\mathcal{O}(\log m)$. These logarithmic factors can actually be eliminated by using suitable hashing techniques. Finally, the test $(e_1, e_2) \in o^+$ can be performed in $\mathcal{O}(m + n)$ by executing a depth-first visit of the subgraph of (E, o) generated by e_1.

In the following, we will denote by $\mathtt{GO}_0$ the traditional generate-and-test algorithm for MVIs computation in EC [2], by $\mathtt{GO}_1$ the graph-traversal algorithm proposed in section 4, and by $\mathtt{GO}_2$ the graph-traversal algorithm described in section 5. We first recall complexities of $\mathtt{GO}_0$ during update and query processing [3,4].

Theorem 11 (Complexities of update and query processing in $\mathtt{GO}_0$). The complexity of update processing is $\mathcal{O}(n + m)$, while the complexity of query processing is $\mathcal{O}(n^3(n + m))$.

The algorithm $\mathtt{GO}_0$ can be extended to compute necessary and possible MVIs in MEC without any increase in computational complexity [3,4].

We now analyze the complexity of the algorithm $\mathtt{GO}_1$.

Theorem 12 (Complexities of update and query processing in $\texttt{GO}_1$). The complexity of update processing is $\mathcal{O}(n + m)$, while the complexity of query processing is $\mathcal{O}(nm + n^2 \log n)$.

Proof. Update processing consists in checking whether or not a pair of events (e_1, e_2) is consistent with o, i.e., whether $(e_2, e_1) \in o^+$ or not, and whether it is non-redundant with respect to o, i.e., whether $(e_1, e_2) \in o^+$ or not. This can be performed by visiting depth-first the subgraph generated by e_2 (respectively e_1) and looking for e_1 (respectively e_2). Since a depth-first visit of a graph costs $\mathcal{O}(n + m)$, this is also the cost of update processing.

As for query processing, $\texttt{GO}_1$ behaves as follows. For every property p and every event e_1 initiating p, the algorithm visits the graph $G(e_1)$ and retrieve all the events e_2 such that $p(e_1, e_2)$ is an MVI. Since the number of properties is constant, the complexity is $\mathcal{O}(n \cdot f(n, m))$, where $f(n, m)$ is the complexity of the procedure that visits the graph $G(e_1)$ and retrieves the nodes that terminate the MVIs for p initiated by e_1. It holds that $f(n, m)$ is the sum of the costs of the visit of $G(e_1)$ and of the processing of the nodes of $G(e_1)$.

The graph $G(e_1)$ is visited breadth-first to construct the layers and to retrieve the terminating events, while it is visited depth-first to propagate the labels `cutoff`. Each edge of the graph $G(e_1)$ is visited at least once (depth-first or breadth-first) and at most twice (first breadth-first, and then depth-first). Indeed, if an edge (e_1, e_2) is depth-first visited, then e_1 is labeled as `marked` or `cutoff`. Hence, neither a breadth-first visit nor a depth-first one will later reconsider it. However, edges which have been already breadth-first visited can also be visited depth-first in order to propagate the label `cutoff`. It follows that the cost of visiting $G(e_1)$ is $\mathcal{O}(m)$.

Similarly, each node of the graph $G(e_1)$ is processed at least once (depth-first or breadth-first) and at most twice (first breadth-first, and then depth-first). Indeed, if the depth-first visit cuts off a node, then it will not be processed anymore. However, nodes labeled as `marked` or `visited`, which have been already processed during the breadth-first visit, can also be processed during a depth-first visit and labeled as `cutoff`. The processing of a node consists of the operations of labeling and testing for relevant events. Both these operations cost $\mathcal{O}(\log n)$. Therefore, processing all nodes of $G(e_1)$ costs $\mathcal{O}(n \log n)$.

Putting together the above results, we conclude that $f(n, m) = \mathcal{O}(m) + \mathcal{O}(n \log n) = \mathcal{O}(m + n \log n)$. This allows us to conclude that the cost of the algorithm is $\mathcal{O}(n \cdot f(n, m)) = \mathcal{O}(nm + n^2 \log n)$. □

A bit surprisingly, the complexity of query processing in $\texttt{GO}_1$ does not increase when moving from the computation of MVIs in EC to the (more significant) computation of possible and necessary MVIs in MEC (the complexity of update processing for EC and MEC in $\texttt{GO}_1$ is exactly the same). Let o be the current knowledge state and $w = o^+$ be its transitive closure. The set $\Box\text{MVI}(w)$ is obtained by intersecting the sets $\text{MVI}(w)$ and $S(w)$, while the set $\Diamond\text{MVI}(w)$ can be obtained by taking the union of $\text{MVI}(w)$ and

$C(w)$. It is easy to see that computing the sets $C(w)$ and $S(w)$ costs $\mathcal{O}(m^- n + n^2 \log n)$, where m^- is the size of o^-. Indeed, computing the transitive closure $w = o^+$ costs $\mathcal{O}(m^- n)$, and sorting w costs $\mathcal{O}(n^2 \log n)$. Computing the set $C(w)$, whenever w is sorted, costs $\mathcal{O}(n^2 \log n)$. The same for the set $S(w)$. Hence, obtaining $C(w)$ and $S(w)$ costs $\mathcal{O}(m^- n + n^2 \log n)$. Since, by hypothesis, E and P are sorted, the resulting sets $C(w)$ and $S(w)$ are sorted too. Moreover, MVI(w) can be sorted in $\mathcal{O}(n^2 \log n)$. Finally, taking the intersection or the union of two sorted sets of size $\mathcal{O}(n^2)$ can be performed in time $\mathcal{O}(n^2)$ by using a simple variant of the algorithm for merging sorted vectors. It follows that $\Box$MVI(w) and $\Diamond$MVI(w) can be computed in $\mathcal{O}(m^- n + n^2 \log n)$, which is not worse than the time needed to compute MVI(w).

We conclude the analysis by determining the computational complexity of the algorithm $\mathtt{GO}_2$.

Theorem 13 (Complexities of update and query processing in $\mathtt{GO}_2$). The complexity of update processing is $\mathcal{O}(nm^- + n^2 \log n)$, while the complexity of query processing is $\mathcal{O}(m^- \log n)$.

Proof. Update processing in $\mathtt{GO}_2$ is performed in three steps. At the first step, EC verifies that neither (e_1, e_2) nor (e_2, e_1) belong to w. If this is the case, it determines the set $\widehat{w} = w \uparrow (e_1, e_2)$. Let $m = |w|$, $m^- = |w^-|$, $\widehat{m} = |\widehat{w}|$, and $\widehat{m}^- = |\widehat{w}^-|$. The tests $(e_1, e_2) \notin w$ and $(e_2, e_1) \notin w$ cost $\mathcal{O}(\log m) = \mathcal{O}(\log n)$. The set $\widehat{w}$ is computed as follows: first, the edge (e_1, e_2) is added to w; then, for every pair of events $e', e'' \in E$ such that $(e', e_1) \in w$, $(e_2, e'') \in w$, and $(e', e'') \notin w$, the edge (e', e'') is added to $w \cup (e_1, e_2)$. The sets of predecessors and successors of a given node can be computed in $\mathcal{O}(n)$ and have cardinality $\mathcal{O}(n)$, and the addition of (e', e'') to $w \cup (e_1, e_2)$ (checking whether or not $(e', e'') \in w$) costs $\mathcal{O}(\log \widehat{m}) = \mathcal{O}(\log n)$; hence, the complexity of computing $\widehat{w}$ is $\mathcal{O}(n^2 \log n)$.

The second step consists in the extraction of $\widehat{w}_p$ from $\widehat{w}$, for every property p. Since $\widehat{w}_p$ contains the edges (e', e'') of $\widehat{w}$ such that both e' and e'' are relevant to p and since each test costs $\mathcal{O}(\log n)$, this step has complexity $\mathcal{O}(\widehat{m} \log n)$.

The last step is the computation of the transitive reduction $\widehat{w}_p^-$, for every property p. Since $\widehat{w}_p$ is acyclic, $\widehat{w}_p^-$ can be computed in $\mathcal{O}(n\widehat{m}^-)$. The resulting cost of update processing is thus $\mathcal{O}(n\widehat{m}^- + \widehat{m} \log n + n^2 \log n) = \mathcal{O}(n\widehat{m}^- + n^2 \log n)$. Since $\widehat{m}^-$ is $\mathcal{O}(m^-)$, update processing is $\mathcal{O}(nm^- + n^2 \log n)$.

Given the transitive closure w of a knowledge state, query processing consists in picking up, for every property p, the p-edges of w_p^-. Since the cardinality of w_p^- is $\mathcal{O}(m^-)$ and verifying whether or not an edge is a p-edge costs $\mathcal{O}(\log n)$, the complexity of query processing is $\mathcal{O}(m^- \log n)$. $\Box$

Moving from EC to MEC does not change the complexity of update processing. Indeed, computing the sets $C(w)$ and $S(w)$ costs $\mathcal{O}(n^2 \log n)$ (note that in this case we do not need to compute and sort the transitive closure of the current knowledge state, because we store it and maintain it sorted). Since, by hypothesis, E and P are sorted,

Table 1

	EC-update	EC-query	MEC-update	MEC-query
GO_0	$\mathcal{O}(n+m)$	$\mathcal{O}(n^3(n+m))$	$\mathcal{O}(n+m)$	$\mathcal{O}(n^3(n+m))$
GO_1	$\mathcal{O}(n+m)$	$\mathcal{O}(nm+n^2\log n)$	$\mathcal{O}(n+m)$	$\mathcal{O}(nm+n^2\log n)$
GO_2	$\mathcal{O}(nm^-+n^2\log n)$	$\mathcal{O}(m^-\log n)$	$\mathcal{O}(nm^-+n^2\log n)$	$\mathcal{O}(m^-\log n+n^2)$

the resulting sets $C(w)$ and $S(w)$ are sorted too. Moreover, for every property p, the set w_p^- can be sorted in $\mathcal{O}(n^2\log n)$.

As for query processing in MEC, there is a little increase in complexity. The set MVI(w) can be computed in $\mathcal{O}(m^-\log n)$. Notice that, since both P and w_p^-, for every property p, are sorted, the resulting set MVI(w) is sorted without any additional cost. The set $\Box$MVI(w) is obtained by intersecting the ordered sets MVI(w) and $S(w)$ by using a simple variant of the algorithm for merging sorted vectors. In a similar way, the set $\Diamond$MVI(w) can be obtained by taking the union of MVI(w) and $C(w)$. Since the intersection or the union of two sorted sets of size $\mathcal{O}(n^2)$ can be determined in $\mathcal{O}(n^2)$, query processing costs $\mathcal{O}(m^-\log n+n^2)$.

7.2. *A comparison of* GO_0, GO_1, *and* GO_2

In this section, we compare the complexities of query and update processing in GO_0, GO_1, and GO_2. Table 1 summarizes the complexity results that have been proved in the previous section. First, notice that the complexity bounds for EC and MEC may differ only in the case of query processing in GO_2, when $\mathcal{O}(m^-)$ is strictly less than $\mathcal{O}(n^2)$. For this reason, we will confine our analysis to EC. In order to properly compare the proposed algorithms, we must take into account that algorithms GO_0 and GO_1 are *query-driven*, that is, they perform most of their computation at query time, whereas GO_2 is *update-driven*, that is, it mostly works at update time. As a consequence, GO_0 and GO_1 are easily comparable: query processing in GO_1 is more efficient than query processing in GO_0, while the complexity of update processing in GO_0 and GO_1 is the same. Comparing GO_0 and GO_1 with GO_2 is a more difficult task, because (i) GO_2 is an update-driven algorithm, whereas GO_0 and GO_1 are query-driven algorithms, and (ii) the complexity bounds for GO_2 are expressed in terms of the number m^- of edges of the transitive reduction of the knowledge state, whereas those for GO_0 and GO_1 refer to the number m of edges of the knowledge state. To cope with these problems, we execute a case-based analysis, that distinguishes among the following alternative possibilities (cf. figure 5):

- **A1** both the current knowledge state and its transitive reduction are *dense* graphs, i.e., both m and m^- are $\Theta(n^2)$;
- **A2** both the current knowledge state and its transitive reduction are *sparse* graphs, i.e., both m and m^- are $\Theta(n)$;
- **A3** the current knowledge state is a *dense* graph, while its transitive reduction is a *sparse* graph, i.e., $m=\Theta(n^2)$ and $m^-=\Theta(n)$.

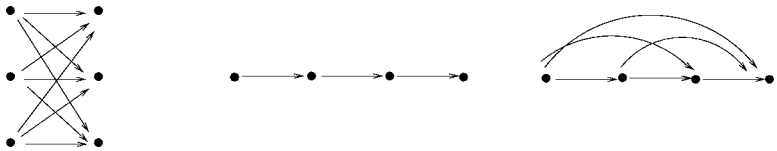

Figure 5. Examples of scenarios **A1**, **A2**, and **A3** (left to right).

Table 2

	EC-update	EC-query
GO_0	$\mathcal{O}(n^2)$	$\mathcal{O}(n^5)$
GO_1	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
GO_2	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2 \log n)$

Let $\#q$ and q (respectively $\#u$ and $u = 1 - q$) be the number of queries and the *query frequency* (respectively the number of updates and the *update frequency*) of the EC-system. We will consider the following cases:

B1 $\#q \gg \#u$ (we can assume $q = 1$ and $u = 0$);

B2 $\#u \gg \#q$ (we can assume $q = 0$ and $u = 1$);

B3 neither $\#q \gg \#u$ nor $\#u \gg \#q$, and $n \gg 0$;

B4 neither $\#q \gg \#u$ nor $\#u \gg \#q$, and n is small.

The complexity bounds (EC only) in case **A1** are given in table 2. The *global* (i.e., update plus query) computational cost is $G(q, n) = (1 - q) \cdot C_u(n) + q \cdot C_q(n)$, where $C_u(n)$ and $C_q(n)$ are the complexities of update and query processing, respectively (figure 6, left side). As for the global cost, in case **B1**, GO_2 is better than GO_1, which, in its turn, is better than GO_0, while, in case **B2**, GO_0 and GO_1 are equivalent and better than GO_2. In case **B3**, the actual value of q and u does not matter, since q and u are comparable and n is large. Hence, the global cost is $C_u(n) + C_q(n)$, that is, $\mathcal{O}(n^5)$ for GO_0, and $\mathcal{O}(n^3)$ for GO_1 and GO_2. It follows that, in this case, GO_1 and GO_2 are equivalent and both of them are more efficient than GO_0. On the contrary, in case **B4**, the actual value of q and u must be taken into account, because n is small. However, if we do not know the actual value of q or u, we can compute the *average value* $\mu(n)$ of the function $G(q, n)$ varying q over the interval [0, 1]. It holds that

$$\mu(n) = \int_0^1 G(q, n)\, \mathrm{d}q.$$

We have that $\mu(n) = \frac{1}{2}n^2(n^3 + 1)$ for GO_0, $\mu(n) = \frac{1}{2}n^2(n + 1)$ for GO_1, and $\mu(n) = \frac{1}{2}n^2(n + \log n)$ for GO_2. Hence, regarding to the average value of $G(q, n)$, GO_1 is the best solution.

The complexity bounds in case **A2** are given in table 3. This case is similar to the previous one, and the outcomes of the comparison are exactly the same (cf. figure 6, middle).

Table 3

	EC-update	EC-query
GO_0	$\mathcal{O}(n)$	$\mathcal{O}(n^4)$
GO_1	$\mathcal{O}(n)$	$\mathcal{O}(n^2 \log n)$
GO_2	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n \log n)$

Table 4

	EC-update	EC-query
GO_0	$\mathcal{O}(n^2)$	$\mathcal{O}(n^5)$
GO_1	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
GO_2	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n \log n)$

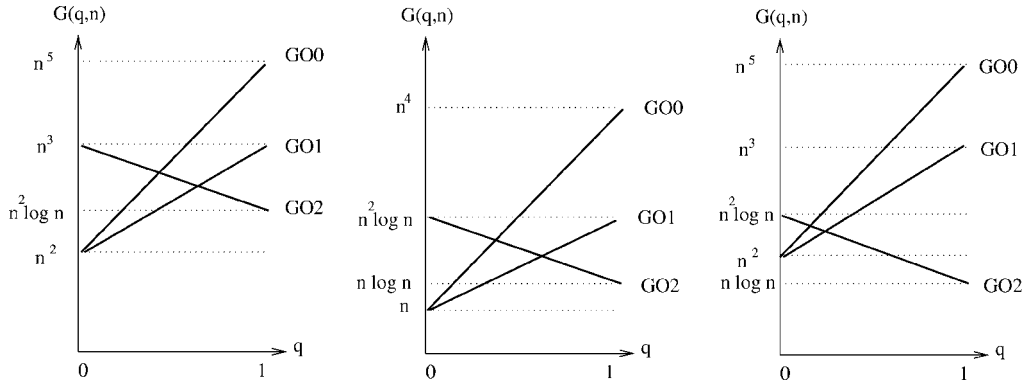

Figure 6. The behaviour of the global cost function $G(q, n)$ in cases **A1** (left), **A2** (middle), and **A3** (right).

Finally, in case **A3**, the complexity bounds are given in table 4. The *global* computational cost $G(q, n) = (1 - q) \cdot C_u(n) + q \cdot C_q(n)$ is illustrated in figure 6, right side. Cases **B1** and **B2** are as above, while cases **B3** and **B4** are different. In case **B3**, the global cost of GO_2 ($\mathcal{O}(n^2 \log n)$) is less than the global cost of GO_1 ($\mathcal{O}(n^3)$). The same in case **B4**. Indeed, $\mu(n) = \frac{1}{2}n^2(n + 1)$ for GO_1 and $\mu(n) = \frac{1}{2}n \log n(n + 1)$ for GO_2.

8. Conclusions and future work

In this paper, we have shown how well-known graph-theoretic techniques can be successfully exploited to efficiently reason about partially ordered events in EC and MEC. Even though we developed our solution in the context of (Modal) Event Calculus, we expect it to be applicable to any formalism for reasoning about partially ordered events.

Whenever the system is regularly queried and updated, i.e., the frequencies of queries and updates are comparable, and the number of recorded events is quite large (the usual case), both the alternative graph-traversal algorithms GO_1 and GO_2 are much

more efficient than the standard generate-and-test one GO_0. Moreover, GO_2 is better than GO_1 whenever the transitive reduction of the current knowledge state is sparse and the current knowledge state is dense (this was conjectured by Chittaro et al. [8]). In the other relevant cases, the two algorithms are equivalent.

As for future work, we are mainly interested in the following two research directions. On the one hand, we are looking for meaningful lower bounds to the (tractable instances of the) problem of reasoning about partially ordered events. On the other hand, we would like to apply the proposed graph-traversal algorithms to other (polynomial) extensions of EC [4,5]. As an example, Cervesato et al. [4] studied the effects of the addition of preconditions to (Modal) Event Calculus. The resulting *Event Calculus with Preconditions* (PEC) and its modal variant (MPEC) aim at modeling situations that consist of a set of events, whose occurrences over time have the effect of initiating or terminating the validity of properties *when given preconditions are met*. Computing MVIs in PEC remains a polynomial task, albeit the polynomial degree depends on the nesting level of preconditions. On the contrary, as already pointed out by Dean and Boddy [10], an unconstrained use of preconditions makes the problem of deriving the set of necessary and possible MVIs *intractable*. To overcome this negative complexity outcome, Cervesato et al. [4] developed polynomial approximate procedures for the computation of necessary and possible MVIs in MPEC, that are in general either sound (but not complete) or complete (but not sound) with respect to the semantics of the corresponding modal event calculi. We believe that the graph-traversal algorithms we have proposed in this paper can be successfully exploited in order to speed up the computation of MVIs in PEC and of the approximations of MVIs in MPEC.

Acknowledgements

The authors were supported by the MURST Project SALADIN (Software Architectures and LAnguages to coordinate DIstributed mobile compoNents).

References

[1] A.V. Aho, M.R. Garey and J.D. Ullman, The transitive reduction of a directed graph, SIAM Journal on Computing 1(2) (1972) 131–137.

[2] I. Cervesato, L. Chittaro and A. Montanari, A modal calculus of partially ordered events in a logic programming framework, in: *Proceedings of the Twelfth International Conference on Logic Programming – ICLP'95*, ed. L. Sterling, Kanagawa, Japan, 13–16 June 1995 (MIT Press, 1995) pp. 299–313.

[3] I. Cervesato, M. Franceschet and A. Montanari, The complexity of model checking in modal event calculi with quantifiers, Electronic Transactions on Artificial Intelligence 2 (1998) 1–23.

[4] I. Cervesato, M. Franceschet and A. Montanari, A guided tour through some extensions of the event calculus, Computational Intelligence 16(2) (2000) 307–347.

[5] I. Cervesato, M. Franceschet and A. Montanari, A hierarchy of modal event calculi: Expressiveness and complexity, in: *Advances in Temporal Logic*, eds. H. Barringer, M. Fisher, D. Gabbay and G. Gough, Applied Logic Series (Kluwer Academic, 2000) pp. 1–20.

[6] I. Cervesato and A. Montanari, A general modal framework for the event calculus and its skeptical and credulous variants, Journal of Logic Programming 38(2) (1999) 111–164.
[7] D. Chapman, Planning for conjunctive goals, Artificial Intelligence 32 (1987) 333–377.
[8] L. Chittaro, A. Montanari and I. Cervesato, Speeding up temporal reasoning by exploiting the notion of kernel of an ordering relation, in: *Proc. of the 2nd International Workshop on Temporal Representation and Reasoning – TIME'95*, Melbourne Beach, FL (26 April 1995) pp. 73–80.
[9] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms* (MIT Press, 1989).
[10] T. Dean and M. Boddy, Reasoning about partially ordered events, Artificial Intelligence 36 (1988) 375–399.
[11] A. Goralcikova and V. Koubek, A reduct and closure algorithm for graphs, in: *Proc. of the 8th Symposium on Mathematical Foundations of Computer Science*, Olomouc, CZ, Lecture Notes in Computer Science Vol. 74 (Springer, 1979) pp. 301–307.
[12] R. Kowalski, Database updates in the event calculus, Journal of Logic Programming 12 (1992) 121–146.
[13] R. Kowalski and M. Sergot, A logic-based calculus of events, New Generation Computing 4 (1986) 67–95.
[14] B. Nebel and C. Bäckström, On the computational complexity of temporal projection, planning, and plan validation, Artificial Intelligence 66 (1994) 125–160.
[15] K. Simon, An improved algorithm for transitive closure on acyclic digraphs, Theoretical Computer Science 58(1–3) (1988) 325–346.
[16] J. van Leeuwen, Graph algorithms, in: *Handbook of Theoretical Computer Science. Vol. A: Algorithms and Complexity*, ed. J. van Leeuwen (Elsevier, 1990) pp. 525–632.