# Event Calculus with Explicit Quantifiers*

Iliano Cervesato†, Massimo Franceschet‡, and Angelo Montanari‡

† Department of Computer Science
Stanford University
Stanford, CA 94305-9045
*iliano@cs.stanford.edu*

‡ Dipartimento di Matematica e Informatica
Università di Udine
Via delle Scienze, 206 – 33100 Udine, Italy
*franzesc@dimi.uniud.it; montana@dimi.uniud.it*

## Abstract

*Kowalski and Sergot's Event Calculus (EC) is a simple temporal formalism that, given a set of event occurrences, derives the maximal validity intervals* (MVIs) *over which properties initiated or terminated by these events hold. We extend the range of queries accepted by EC, so far limited to boolean combinations of* MVI *verification or computation requests, to admit arbitrary quantification over events and properties. We demonstrate the added expressive power by encoding a medical diagnosis problem as a case study. Moreover, we give an implementation of this formalism and analyze the computational complexity of the extended calculus.*

## 1 Introduction

The *Event Calculus*, abbreviated *EC* [5], is a simple temporal formalism designed to model and reason about scenarios characterized by a set of *events*, whose occurrences have the effect of starting or terminating the validity of determined properties. Given a (possibly incomplete) description of when these events take place and of the properties they affect, *EC* is able to determine the *maximal validity intervals*, or *MVIs*, over which a property holds uninterruptedly. In practice, since this formalism is usually implemented as a logic program, *EC* can also be used to check the truth of *MVIs* and process boolean combinations of *MVI* verification or computation requests. The range of queries that can be expressed in this way is however too limited for modeling realistic situations.

A systematic analysis of *EC* has recently been undertaken in order to gain a better understanding of this calculus and determine ways of augmenting its expressive power. The keystone of this endeavor has been the definition of an extendible formal specification of the functionalities of this formalism [2]. This has had the effects of establishing a semantic reference against which to verify the correctness of implementations [2], of casting *EC* as a model checking problem [3], and of setting the ground for studying the complexity of this problem, which was proved polynomial [1]. Extensions of this model have been designed to accommodate constructs intended to enhance the expressiveness of *EC*. In particular, modal versions of *EC* [2], the interaction between modalities and connectives [3], and preconditions [4] have all been investigated in this context.

In this paper, we continue the endeavor to enhance the expressive power of *EC* by considering the possibility of quantifying over events and properties in queries. We also admit boolean connectives and requests to verify the relative order of two events. We show that the resulting language, that we call *QCEC*, can effectively be used to encode interesting problems in medical diagnosis. Moreover, we provide an elegant implementation in the higher-order logic programming language *λProlog* [6] and prove its soundness and completeness. Finally, we analyze the complexity of the model checking problem involving this language.

The main contributions of this work are: (1) the extension of the event calculus with quantifiers; (2) permitting queries to mention ordering information; and (3) the use of the higher-order features of modern logic programming languages in temporal reasoning.

This paper is organized as follows. In Section 2, we formalize *QCEC*. Section 3 is devoted to exemplifying how this calculus can adequately model certain medical diagnosis problems. In Section 4, we briefly introduce the logic programming language *λProlog*, give an implementation of *QCEC* in it and prove the sound-

*The first author was supported by ONR grant N00014-97-1-0505, Multidisciplinary University Research Initiative *Semantic Consistency in Information Exchange*. The work of the third author was partially supported by the CNR project *Programmazione logica: strumenti per analisi e trasformazione di programmi; tecniche di ingegneria del software; estensioni con vincoli, concorrenza ed oggetti (STE)*.

Appeared in the *Proceedings of the Fifth International Workshop on Temporal Representation and Reasoning — TIME'98* (R. Morris, L. Khatib editors), pp. ??–??, IEEE Computer Society Press, Sanibel Island, FL, 16–17 May 1998.

ness and completeness of the resulting program. In Section 5, we analyze the complexity of *QCEC*. We outline directions of future work in Section 6.

## 2 Event Calculus with Quantifiers

In this section, we first recall the syntax and semantics of the Event Calculus, *EC* for short (2.1). We then extend this basic definition to give a semantic foundation to the Event Calculus with Connectives and Quantifiers, abbreviated *QCEC* (2.2).

### 2.1 EC

The Event Calculus (*EC*) [5] and the extension we propose aim at modeling scenarios that consist of a set of events, whose occurrences over time have the effect of initiating or terminating the validity of properties, some of which may be mutually exclusive. We formalize the time-independent aspects of a situation by means of an *EC-structure* [2], defined as follows:

**Definition 2.1** (*EC-structure*)
*A* structure *for the* Event Calculus (EC-structure) *is a quintuple* $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$ *such that:*

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

The temporal aspect of *EC* is given by the order in which events happen. For the sake of generality [2], we admit scenarios in which the occurrence times of events are unknown or in which the relative order of event happenings is incomplete. Clearly our argument specializes to the common situation where every event has an associated occurrence time. We however require the temporal information to be consistent so that an event cannot both precede and follow some other event. In its most basic form, *EC* does not take the evolution of the event ordering into account, but operates on temporal snapshots. We can then formalize the time-dependent aspect of a scenario modeled by *EC* by means of a (strict) *partial order* for the involved event occurrences. We write $W_{\mathcal{H}}$ for the set of all partial orders over the set of events $\mathbf{E}$ in $\mathcal{H}$, use the letter $w$ to denote individual orderings and write $e_1 <_w e_2$ to indicate that $e_1$ precedes $e_2$ in the ordering $w$. For reasons of efficiency, implementations usually represent the temporal information of an *EC* problem as a *binary acyclic relation* $o$ from which $w$ can be recovered by taking its transitive closure, written $o^+$.

Given a structure $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$ and an event ordering $w$, we call the pair $(\mathcal{H}, w)$ an *EC-problem*. *EC* permits inferring the *maximal validity intervals*, or *MVIs*, over which a property $p$ holds uninterruptedly. We represent an MVI for $p$ as $p(e_i, e_t)$, where $e_i$ and $e_t$ are the events that respectively initiate and terminate the interval over which $p$ holds maximally. Consequently, we adopt as the *query language* of an *EC* problem $(\mathcal{H}, w)$ the set

$$\mathcal{L}_{\mathcal{H}}(EC) = \{p(e_1, e_2) : p \in \mathbf{P} \text{ and } e_1, e_2 \in \mathbf{E}\}$$

of all such property-labeled intervals over $\mathcal{H}$. We interpret the elements of $\mathcal{L}_{\mathcal{H}}(EC)$ as propositional letters and the task performed by *EC* reduces to deciding which of these formulas are MVIs in $w$ and which are not. This is a model checking problem.

In order for $p(e_1, e_2)$ to be an MVI relative to the event ordering $w$, it must be the case that $e_1 <_w e_2$. 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 $br(p, e_1, e_2, w)$ 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 $p$ itself or a property exclusive with $p$ interrupts the validity of $p(e_1, e_2)$. These observations are formalized as follows.

**Definition 2.2** (*Intended model of EC*)
*Let* $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$ *be a EC-structure and* $w \in W_{\mathcal{H}}$*. The* intended EC-model *of* $(\mathcal{H}, w)$ *is the propositional valuation* $\upsilon_{(\mathcal{H},w)} \subseteq \mathcal{L}_{\mathcal{H}}(EC)$*, where* $p(e_1, e_2) \in \upsilon_{(\mathcal{H},w)}$ *if and only if*

*i.* $e_1 <_w e_2$*;*
*ii.* $e_1 \in [p\rangle$*;*
*iii.* $e_2 \in \langle p]$*;*
*iv.* $br(p, e_1, e_2, w)$ *does not hold, where* $br(p, e_1, e_2, w)$ *abbreviates:*
*there exists an event* $e \in \mathbf{E}$ *such that* $e_1 <_w e$*,* $e <_w e_2$ *and there exists a property* $q \in \mathbf{P}$ *such that* $e \in [q\rangle$ *or* $e \in \langle q]$*, and either* $]p, q[$ *or* $p = q$*.* □

## 2.2 QCEC

We will now enrich the query language of the Event Calculus with universal and existential quantifiers over both events and properties. In order to make the resulting formalism more interesting, we further add boolean connectives and the possibility of testing the relative order of events. Indeed, a logic programming implementation of *EC* can emulate existential quantification over individual formulas in $\mathcal{L}_{\mathcal{H}}(EC)$ by means of unification, and moreover, universally quantified formulas in this language always have trivial solutions. We call the resulting formalism the *Event Calculus with Connectives and Quantifiers*, or *QCEC* for short.

The addition of connectives, precedence testing and unrestricted quantification over events gives *QCEC* a considerably improved expressive power with respect to *EC*. This will be demonstrated in Section 3 where we will be able to encode a medical diagnosis problem that cannot be easily tackled by *EC*. The computational complexity of the extended calculus remains polynomial in the numbers of events, but becomes exponentials in the quantifiers nesting of the query, as we will see in Section 5. However, in realistic applications the query size is likely to be much smaller than the number of recorded events.

Quantifiers over property do not appear to enhance significantly the expressiveness of *EC* due to the tight relation between properties and events, hard-coded in the initiation and termination maps. However, we expect substantial benefits in a language that admits the use of preconditions [4]. We nonetheless treat property quantifiers since they are handled similarly to quantification over events.

In order to accommodate quantifiers, we need to extend the query language of an *EC* problem $(\mathcal{H}, w)$, with $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$, in several respects. We first assume the existence of infinitely many *event variables* that we denote $E$, possibly subscripted. We similarly need a countable set of *property variables*, indicated with the letter $P$ variously decorated. We write $\bar{e}$ for a syntactic entity that is either an event in $\mathbf{E}$ or an event variable. We adopt a similar notation in the case of properties. The query language of *QCEC*, denoted $\mathcal{L}_{\mathcal{H}}(QCEC)$, is then the set of closed formulas generated by the following grammar:

$$\varphi \ ::= \ \bar{p}(\bar{e}_1, \bar{e}_2) \mid \bar{e}_1 < \bar{e}_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \forall E.\,\varphi \mid \exists E.\,\varphi \mid \forall P.\,\varphi \mid \exists P.\,\varphi.$$

where $\bar{e}_1 < \bar{e}_2$ denotes the test of whether $\bar{e}_1$ precedes $\bar{e}_2$. Observe that $\forall$ and $\exists$ have been overloaded to indicate quantification over both events and properties; the nature of the syntactic variable that follows these symbols allows disambiguating their use. In addition to the operators above, we also admit implication ($\supset$) as a derived connective, where $\varphi_1 \supset \varphi_2$ is classically defined as $\neg\varphi_1 \vee \varphi_2$.

The notions of free and bound variables are defined as usual and we identify formulas that differ only by the name of their bound variables. We write $[e/E]\varphi$ for the substitution of an event $e \in \mathbf{E}$ for every free occurrence of the event variable $E$ in the formula $\varphi$, and similarly for properties. Notice that this limited form of substitution cannot lead to variable capture.

We now extend the definition of intended model of an *EC*-problem $(\mathcal{H}, w)$ from formulas in $\mathcal{L}_{\mathcal{H}}(EC)$ to objects in $\mathcal{L}_{\mathcal{H}}(QCEC)$. To this aim, we need to define the notion of validity for the new constructs of *QCEC*.

**Definition 2.3** (*Intended model of QCEC*)
*Let* $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$ *be an EC-structure and* $w$ *an event ordering. The* intended QCEC-model *of* $\mathcal{H}$ *and* $w$ *is the classical model* $\mathcal{I}_{(\mathcal{H},w)}$ *built on top of the valuation* $\upsilon_{(\mathcal{H},w)}$. *Given a (closed) formula* $\varphi \in \mathcal{L}_{\mathcal{H}}(QCEC)$, *the truth of* $\varphi$ *at* $\mathcal{I}_{(\mathcal{H},w)}$, *denoted as* $\mathcal{I}_{(\mathcal{H},w)} \models \varphi$, *is inductively defined as follows:*

$\mathcal{I}_{(\mathcal{H},w)} \models p(e_1, e_2)$ *iff* $p(e_1, e_2) \in \upsilon_{(\mathcal{H},w)}$;
$\mathcal{I}_{(\mathcal{H},w)} \models e_1 < e_2$ *iff* $e_1 <_w e_2$;
$\mathcal{I}_{(\mathcal{H},w)} \models \neg\varphi$ *iff* $\mathcal{I}_{(\mathcal{H},w)} \not\models \varphi$;
$\mathcal{I}_{(\mathcal{H},w)} \models \varphi_1 \wedge \varphi_2$ *iff* $\mathcal{I}_{(\mathcal{H},w)} \models \varphi_1$ *and* $\mathcal{I}_{(\mathcal{H},w)} \models \varphi_2$;
$\mathcal{I}_{(\mathcal{H},w)} \models \varphi_1 \vee \varphi_2$ *iff* $\mathcal{I}_{(\mathcal{H},w)} \models \varphi_1$ *or* $\mathcal{I}_{(\mathcal{H},w)} \models \varphi_2$;
$\mathcal{I}_{(\mathcal{H},w)} \models \forall E.\,\varphi$ *iff for all* $e \in \mathbf{E}$, $\mathcal{I}_{(\mathcal{H},w)} \models [e/E]\varphi$;
$\mathcal{I}_{(\mathcal{H},w)} \models \exists E.\,\varphi$ *iff there exists* $e \in \mathbf{E}$ *such that* $\mathcal{I}_{(\mathcal{H},w)} \models [e/E]\varphi$;
$\mathcal{I}_{(\mathcal{H},w)} \models \forall P.\,\varphi$ *iff for all* $p \in \mathbf{P}$, $\mathcal{I}_{(\mathcal{H},w)} \models [p/P]\varphi$;
$\mathcal{I}_{(\mathcal{H},w)} \models \exists P.\,\varphi$ *iff there exists* $p \in \mathbf{P}$ *such that* $\mathcal{I}_{(\mathcal{H},w)} \models [p/P]\varphi$. □

The well-foundedness of this definition derives from the observation that if $\forall E.\,\varphi$ is a closed formula, so is $[e/E]\varphi$ for every event $e \in \mathbf{E}$, and similarly for the other quantifiers.

A universal quantification over a finite domain can always be expanded as a finite sequence of conjunctions. Similarly an existentially quantified formula is equivalent to the disjunction of all its instances. The following lemma, whose simple proof we omit, applies these principles to *QCEC*.

**Lemma 2.4** (*Unfolding quantifiers*)
*Let* $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$ *be an* EC-*structure, with* $\mathbf{E} = \{e_1, \ldots, e_n\}$ *and* $\mathbf{P} = \{p_1, \ldots, p_m\}$. *Then, for every* $w \in W_{\mathcal{H}}$,

(*i*) $\mathcal{I}_{(\mathcal{H},w)} \models \forall E.\,\varphi$ *iff* $\mathcal{I}_{(\mathcal{H},w)} \models \bigwedge_{i=1}^{n} [e_i/E]\varphi$;
(*ii*) $\mathcal{I}_{(\mathcal{H},w)} \models \exists E.\,\varphi$ *iff* $\mathcal{I}_{(\mathcal{H},w)} \models \bigvee_{i=1}^{n} [e_i/E]\varphi$;
(*iii*) $\mathcal{I}_{(\mathcal{H},w)} \models \forall P.\,\varphi$ *iff* $\mathcal{I}_{(\mathcal{H},w)} \models \bigwedge_{i=1}^{m} [p_i/P]\varphi$;
(*iv*) $\mathcal{I}_{(\mathcal{H},w)} \models \exists P.\,\varphi$ *iff* $\mathcal{I}_{(\mathcal{H},w)} \models \bigvee_{i=1}^{m} [p_i/P]\varphi$. ■

This property hints at the possibility of compiling a $QCEC$ query to a formula that does not mention any quantifier. Observe however that this is possible only after an $EC$-structure has been specified. Therefore, quantifiers are not simply syntactic sugar, but an effective extension over a query language with connectives.

We will rely on the above lemma in order to analyze the computational complexity of the formalism in Section 5. However, we will not take advantage of it to implement $QCEC$ in Section 4 since a model checker should be independent from the particular $EC$-problem it is operating on.

## 3 Example

In this section, we consider an example taken from the domain of medical diagnosis that shows how an extension of $EC$ with quantifiers and connectives can be conveniently used to deal with significant applications.

We focus our attention on repeated clusters of events whose correlation, if present, can entail conclusions about the state of the system under observation. As an example, consider the following rule of thumb for diagnosing malaria [7]:

> *A malaria attack begins with chills that are followed by high fever. Then the chills stop and some time later the fever goes away as well. Malaria is likely if the patient has repeated episodes of malaria attacks.*

Figure 1 describes the symptoms of a patient, Mr. Jones, who has just returned from a vacation to the Tropics. We have labeled the beginning and the end of chills and fever periods for reference. According to the rule above, Mr. Jones should be diagnosed with malaria. If however he had not had fever in the period between $e_6$ and $e_8$ for example, or if $e_7$ had preceded $e_6$, then further checks should be made in order to ascertain the kind of ailment he suffers from.

We will now show how the rule above can be expressed as a $QCEC$ query in order to automate the diagnosis of malaria. The first task is to give a representation of symptom records as $EC$-problems. In the case of Mr. Jones, the factual information of his condition is represented by the $EC$-structure $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$ below, which is a direct transliteration of the data in Figure 1.

- $\mathbf{E} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$,
- $\mathbf{P} = \{\mathit{chills}, \mathit{fever}\}$,
- $[\mathit{chills}\rangle = \{e_1, e_5, e_9\}$, $[\mathit{fever}\rangle = \{e_2, e_6, e_{10}\}$,
- $\langle \mathit{chills}] = \{e_3, e_7, e_{11}\}$, $\langle \mathit{fever}] = \{e_4, e_8, e_{12}\}$,
- $]\cdot,\cdot[ = \emptyset$.

The events that initiate and terminate the symptoms of Mr. Jones happened in ascending order of their indices. We call $w$ the corresponding ordering.

The decision rule for diagnosing malaria can then be reworded as saying that "*whenever there is an episode of chills, there is a successive period of fever that starts before the chills are over*".[1] It can in turn be expressed by the following formula in $\mathcal{L}_{\mathcal{H}}(QCEC)$:

$$\varphi = \forall E_1.\, \forall E_2.\, (\mathit{chills}(E_1, E_2) \supset (\exists E_1'.\, \exists E_2'.\\ (E_1 < E_1' \ \wedge\ E_1' < E_2 \ \wedge\ \mathit{fever}(E_1', E2'))))$$

that makes use of both universal and existential quantifiers over events, of all the connectives of $QCEC$ (once implication is expanded) and of the precedence test. It is easy to verify that $\mathcal{I}_{(\mathcal{H},w)} \models \varphi$, while this formula is not valid in models where $e_6$ or $e_8$ have been eliminated, or where the relative order of $e_6$ and $e_7$ has been reversed, for example.

There is no way to express this rule in $EC$, even when extended with connectives and the precedence test, unless quantifiers are unfolded as specified in Lemma 2.4. This would have however the undesirable effects of making the formula used to diagnose malaria problem-specific, and to augment considerably the size of this expression.

## 4 Implementation

The Event Calculus [5] has traditionally been implemented in the logic programming language *Prolog* [8]. Recent extensions to $EC$ have instead adopted $\lambda$*Prolog* [6] in order to achieve a declarative yet simple encoding, necessary to formally establish correctness issues [2]. In this section, we will rely on orthogonal features of $\lambda$*Prolog* to obtain an elegant encoding of quantifiers (4.2). Before doing so, we recall the meaning of relevant constructs of this language (4.1). We conclude this section by showing that this program faithfully realizes the specification of $QCEC$ (4.3).

### 4.1 λProlog in a nutshell

Due to space limitations, we shall assume the reader to be familiar with the logic programming language *Prolog* [8]. We will instead illustrate some of the characteristic constructs of $\lambda$*Prolog* at an intuitive level. We invite the interested reader to consult [6] for a more complete discussion, and [2] for a presentation in the context of the Event Calculus.

Unlike *Prolog* which is first-order, $\lambda$*Prolog* is a *higher-order* language, which means that the terms in

[1] The other possible interpretations can be rendered in $QCEC$.

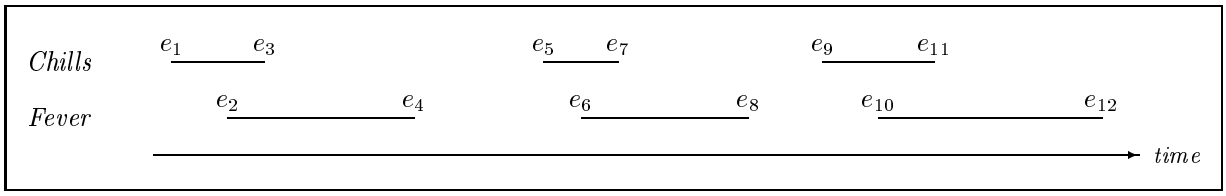


**Figure 1. Symptoms of Patient Jones**

this programming language are drawn from a *simply typed λ-calculus*. More precisely, the syntax of terms is given by the following grammar:

$$M \ ::= \ c \mid x \mid F \mid M_1 \, M_2 \mid x \setminus M$$

where $c$ ranges over *constants*, $x$ stands for a *bound variable* and $F$ denotes a *logical variable* (akin to *Prolog*'s variables). Identifiers beginning with a lowercase and an uppercase letter stand for constants and logical variables, respectively. Terms that differ only by the name of their bound variables are considered indistinguishable. "$x \setminus M$" is *λProlog*'s syntax for *λ-abstraction*, traditionally written $\lambda x.\, M$. In this language, terms and atomic formulas are written in curried form (e.g. "`before E1 E2`" rather than "`before(E1, E2)`").

Every constant, bound variable and logical variable is given a unique type $A$. Types are either user-defined *base types*, or *functional types* of the form $A_1$ `->` $A_2$. By convention, the predefined base type `o` classifies formulas. A base type $a$ is declared as "`kind` $a$`.`", and a constant $c$ of type $A$ is entered in *λProlog* as "`type` $c$ $A$`.`". Syntax is provided for declaring infix symbols. Application and λ-abstraction can be typed if their subexpression satisfy certain constraints. *λProlog* will reject every term that is not typable.

While first-order terms are equal solely to themselves, the equational theory of higher-order languages identifies terms that can be rewritten to each other by means of the *β-reduction* rule: $(x \setminus M)\, N \ = \ [N/x]M$, where the latter expression denotes the capture-avoiding substitution of the term $N$ for the bound variable $x$ in $M$. A consequence of this fact is that unification in *λProlog* must perform β-reduction on the fly in order to equate terms or instantiate logical variables. A further difference from *Prolog* is that logical variables in *λProlog* can stand for functions (i.e. expressions of the form $x \setminus M$) and this must be taken into account when unification is performed.

For our purposes, the language of formulas of *λProlog* differs from *Prolog* for the availability of an explicit existential quantifier in the body of clauses. The goal $\exists x.\, G$ is written "`sigma` $x \setminus G$" in the concrete syntax of this language. We will also take advantage of *negation-as-failure*, denoted `not`. We will not rely directly on the other powerful constructs offered by this language. Other connectives are denoted as in *Prolog*: "`,`" for conjunction, "`;`" for disjunction, "`:-`" for implication with the arguments reversed. The only predefined predicate we will use is the infix "`=`" that unifies its arguments. Given a well-typed *λProlog* program $\mathcal{P}$ and a goal $G$, the fact that there is a derivation of $G$ from $\mathcal{P}$, i.e. that $G$ is solvable in $\mathcal{P}$, is denoted $\mathcal{P} \vdash G$. See [6, 2] for details.

*λProlog* offers also the possibility of organizing programs into modules. A module $m$ is declared as "`module` $m$`.`" followed by the declarations and clauses that define it. Modules can access other modules by means of the `accumulate` declaration.

Finally, `%` starts a comments that extends to the end of the line.

### 4.2 Implementation of QCEC in λProlog

We will now give an implementation of *QCEC* in *λProlog*. The resulting module, called `qcec`, is displayed in Appendix A. The rule to diagnose malaria and the medical record of Mr. Jones from Section 3 are included in Appendices B and C, respectively. This code has been tested using the *Terzo* implementation of *λProlog*, version 1.0b, which is available from `http://www.cse.psu.edu/~dale/lProlog/`.

We define a family of representation functions $\ulcorner \cdot \urcorner$ that relate the mathematical entities we have been using in Section 2 to terms in *λProlog*. Specifically, we will need to encode *EC*-structures, the associated orderings, and the language of *QCEC*. In the remainder of this section, we will refer to a generic *EC*-structure $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$.

We represent $\mathcal{H}$ by giving an encoding of the entities that constitute it. We introduce the types `event` and `property` so that every event in $\mathbf{E}$ (property in $\mathbf{P}$) is represented by a distinct constant of type `event` (of type `property`, respectively). Event and property variables are represented as *λProlog* variables of the relative type. The initiation, termination and exclusivity relations, event occurrences (traditionally represented in *EC*) and property explicitation (needed to

guarantee groundness) are mapped to the predicate symbol `initiates`, `terminates`, `exclusive`, `happens` and `prop`, respectively, applied to the appropriate arguments. Declarations for these constants can be found in Appendix A.

For implementation purposes, it is more convenient to compute the relative ordering of two events on the basis of fragmented data (a binary acyclic relation) than to maintain this information as a strict order. We rely on the binary predicate symbol `beforeFact` to represent the edges of the binary acyclic relation. We encapsulate the clauses for the predicate `before`, which implements its transitive closure, in the module `transClo`. We do not show details for space reasons, but a quadratic implementation can be found in [1].

In order to encode the syntax of *QCEC*, we define the type `mvi`, intended to represent the formulas of this language (as opposed to the formulas of *λProlog*, that have type `o`). The representation of formulas is then relatively standard [2], except for quantifiers:

$$
\begin{aligned}
\ulcorner \bar{p}(\bar{e}_1, \bar{e}_2) \urcorner &= \texttt{period}\ \ulcorner \bar{e}_1 \urcorner\ \ulcorner \bar{p} \urcorner\ \ulcorner \bar{e}_2 \urcorner \\
\ulcorner \bar{e}_1 < \bar{e}_2 \urcorner &= \ulcorner \bar{e}_1 \urcorner\ \texttt{precedes}\ \ulcorner \bar{e}_2 \urcorner \\
\ulcorner \neg \varphi \urcorner &= \texttt{neg}\ \ulcorner \varphi \urcorner \\
\ulcorner \varphi_1 \wedge \varphi_2 \urcorner &= \ulcorner \varphi_1 \urcorner\ \texttt{and}\ \ulcorner \varphi_2 \urcorner \\
\ulcorner \varphi_1 \vee \varphi_2 \urcorner &= \ulcorner \varphi_1 \urcorner\ \texttt{or}\ \ulcorner \varphi_2 \urcorner \\
\ulcorner \varphi_1 \supset \varphi_2 \urcorner &= \ulcorner \varphi_1 \urcorner\ \texttt{implies}\ \ulcorner \varphi_2 \urcorner \\
\ulcorner \forall E.\, \varphi \urcorner &= \texttt{forAllEvent}\ (E \setminus \ulcorner \varphi \urcorner) \\
\ulcorner \exists E.\, \varphi \urcorner &= \texttt{forSomeEvent}\ (E \setminus \ulcorner \varphi \urcorner) \\
\ulcorner \forall P.\, \varphi \urcorner &= \texttt{forAllProp}\ (P \setminus \ulcorner \varphi \urcorner) \\
\ulcorner \exists P.\, \varphi \urcorner &= \texttt{forSomeProp}\ (P \setminus \ulcorner \varphi \urcorner)
\end{aligned}
$$

Quantifiers differ from the other syntactic entities of a language such as *QCEC* by the fact that they *bind* a variable in their argument (e.g. $E$ in $\exists E.\, \varphi$). Bound variables are then subject to implicit renaming to avoid conflicts and to substitution. Encoding binding constructs in traditional programming languages such as *Prolog* is painful since these operations must be explicitly programmed. *λProlog* and other higher-order languages permit a much leaner emulation since λ-abstraction ($X \setminus M$) is itself a binder and their implementations come equiped with (efficient) ways of handling it. The idea, known as *higher-order abstract syntax* [6], is then to use *λProlog*'s abstraction mechanism as a universal binder. Binding constructs in the object language are then expressed as constants that takes a λ-abstracted term as its argument (for example `forSomeEvent` is declared of type `(event -> mvi) -> mvi`). Variable renaming happens behind the scenes, and substitution is delegated to the meta-language as β-reduction.

An example will shed some light on this technique. Consider the formula $\varphi = \exists E.\, p(E, e_2)$, whose representation is

```
forSomeEvent (E \ (period E p e2))
```

where we have assumed that $p$ and $e_2$ are encoded as the constants `p` and `e2`, of the appropriate type. It is easy to convince oneself that this expression is well-typed. In order to ascertain the truth of $\varphi$, we need to check whether $p(e, e_2)$ holds for successive $e \in \mathbf{E}$ until such an event is found. Automating this implies that, given a candidate event $e_1$ (represented as `e1`), we need to substitute `e1` for `E` in `period E p e2`. This can however be achieved by simply applying the argument of `forSomeEvent` to `e1`. Indeed, `(E \ (period E p e2)) e1` is equal to `period e1 p e2`, modulo β-reduction. This technique is used in clauses *8–11* in our implementation.

We represent the truth of a formula in *QCEC* my means of the predicate `holds`. Clauses *1* to *11* in Appendix A implement the specification of this language given in Section 2. More precisely, clauses *1* and *2* provide a direct encoding of Definition 2.1, where clause *2* faithfully emulates the meta-predicate *br*. Clause *3* captures the meaning of the precedence construct, while clauses *4* to *7* reduce the truth check for the connectives of *QCEC* to the derivability of the corresponding *λProlog* constructs. Notice that implication is translated back to a combination of negation and disjunction in clause *7*. Existential quantifiers are handled similarly to connectives in clauses *9* and *11*. Although *λProlog* offers a form of universal quantification, we are forced to take a detour and express our universal quantifiers as negations and existentials in clauses *8* and *10*. A lengthy discussion of the logical reasons behind this step can be found in [2]. The conjunct `prop P` in clause *10* is needed in order to prevent passing uninstantiated logical variables to the negation-as-failure operator.

### 4.3 Soundness and Completeness

The encoding we have chosen as an implementation of *QCEC* permits an easy proof of its faithfulness with respect to the formal specification of this formalism. Key factors in the feasibility of this endeavor are the precise semantic definition of *QCEC* given in Section 2, and the exploitation of the declarative features of *λProlog*.

We only show the statement of our soundness and completeness result since a fully worked out proof would require a very detailed account of the semantics of *λProlog*, and is rather long, although simple. Space constraints prevent us from doing so. The interested reader can find the full development of a proof that relies on the same techniques in [2].

**Theorem 4.1** (*Soundness and completeness of* `qcec`)
*Let* $\mathcal{H} = (\mathbf{E},\ \mathbf{P},\ [\cdot\rangle,\ \langle\cdot],\ ]\cdot,\cdot[)$ *be an EC-structure,* $o$ *a binary acyclic relation over* $\mathbf{E}$ *and* $\varphi$ *and formula in* $\mathcal{L}_{\mathcal{H}}(QCEC)$*, then*

$\texttt{qcec}, \ulcorner\mathcal{H}\urcorner, \ulcorner o\urcorner \vdash \texttt{holds}\ \varphi$ *iff* $\mathcal{I}_{(\mathcal{H},o^+)} \models \varphi$. ■

## 5 Complexity Analysis

Given an *EC*-structure $\mathcal{H}$, an ordering $w \in W_{\mathcal{H}}$ and a formula $\varphi$, we want to characterize the computational complexity of establishing whether $\mathcal{I}_{(\mathcal{H},w)} \models \varphi$ is true as a function of the size of both $\mathcal{H}$ and $\varphi$. This is a model checking problem. We call the triple $(\mathcal{H}, w, \varphi)$ an *instance* of the problem.

The notion of cost we adopt is as follows: we assume that verifying the truth of the propositions $e \in [p\rangle$, $e \in \langle p]$ and $]p, p'[$ has constant cost $O(1)$, for given event $e$ and properties $p$ and $p'$. Although possible in principle, it is disadvantageous in practice to implement event orderings so that the test $e_1 <_w e_2$ has constant cost. We instead maintain an acyclic binary relation $o$ on events whose transitive closure $o^+$ is $w$ (see Section 4). Verifying whether $e_1 <_w e_2$ holds becomes a reachability problem in $o$ and it can be solved in time $O(n^2)$ in the number $n$ of events [1]. The cost of solving the query $e_1 < e_2$ is therefore quadratic.

Model checking in *EC* is known to have cubic cost $O(n^3)$, where $n$ is the number of events in $\mathcal{H}$ [1]. Admitting connectives implies solving as many *EC* problems as there are binary operators in the query, plus 1. Therefore, given a formula $\varphi$ containing $k$ binary operators, model checking it has cost $O(kn^3)$ [3]. This bound does not change if we consider precedence queries: solving $e_1 < e_2$ has complexity $O(n^2)$, and therefore, *ab abundantia*, $O(kn^3)$ for any positive $k$.

We will exploit the unfolding lemma (2.4) to reduce the determination of the complexity of model checking in *QCEC* to the analogous problem in a setting deprived of quantifiers. Consider first the case of quantification over events. This lemma affirms that every formula involving one event quantifier at its top-level can be replaced by the conjunction of $n$ instances of it. If we have a nesting of $q_e$ such quantifiers, we are led to solve $n^{q_e}$ instances. In general, if we eliminate in this manner all event quantifiers in a formula $\varphi$ with $k$ binary connectives, we will produce a formula $\varphi'$ without quantifiers but with at most $kn^{q_e}$ connectives. This implies that the cost of solving a *QCEC* query without property quantifiers is at most $O(kn^{q_e+3})$, where $q_e$ is now the maximum nesting of event quantifiers in $\varphi$. Notice that the strategy suggested by the unfolding lemma has optimal cost since proving that $\forall E.\,\varphi$ holds requires checking $[e_i/E]\varphi$ for all $n$ events $e_i$ in $\mathbf{E}$, while disproving $\exists E.\,\varphi$ implies checking similarly $[e_i/E]\varphi$ for all these events. Similar considerations are in order if the formula at hand contains a nesting of at most $q_p$ quantifications over properties and there are $m$ properties. These results are combined in the following theorem, where the complexity parameters $n$, $m$, $k$, $q_e$ and $q_p$ have been defined above. Notice that $n$ and $m$ are bound to the *EC*-structure $\mathcal{H}$, while the remaining quantities depend on the query $\varphi$.

**Theorem 5.1** (*Complexity of model checking*)
*Given an instance* $(\mathcal{H}, w, \varphi)$*, the test* $\mathcal{I}_{(\mathcal{H},w)} \models \varphi$ *has cost* $O(kn^{q_e+3}m^{q_p})$. ■

The program in Appendix A is a direct transliteration of the definition of *QCEC* in *λProlog*. It is therefore easy to check that the complexity of this algorithm coincides with the bound we just achieved for the problem it implements, if we assume a quadratic implementation of `before`. Moreover, it is possible to show that model checking in *QCEC* is PSPACE-complete, and thus, unless P = PSPACE, there are no algorithms for this problem that perform significantly better (in polynomial time, say) than the one we propose.

Practical applications using event calculus techniques are expected to model situations involving a large number of events, while the size of the queries will in general be limited. The medical example in Section 3 falls into this category. In such contexts, the fact that *QCEC* is polynomial in the number of events is essential. The weight of the high exponents (checking for malaria has cost $O(n^7)$ for example) can often be lowered by pushing quantifiers inside formulas and detecting vacuous quantifications.

## 6 Conclusions and Future Work

In this paper, we have extended the Event Calculus [2, 5] with the possibility of using quantifiers, connectives and precedence tests in queries. The net effect of these combined additions has been a substantial gain in expressiveness with acceptable extra computational cost for queries of a reasonable size. We have provided an implementation of the resulting calculus in the higher-order logic programming language *λProlog* [6], which we used to encode a case study from the area of medical diagnosis. We intend to explore the interaction of these ideas with recently proposed extensions of the Event Calculus with operators from modal logic [2, 3] and preconditions [4].

## References


[1] I. Cervesato, L. Chittaro, and A. Montanari. Speeding up temporal reasoning by exploiting the notion of kernel of an ordering relation. In S. Goodwin and H. Hamilton, editors, *Proceedings of the Second International Workshop on Temporal Representation and Reasoning — TIME'95*, pages 73–80, Melbourne Beach, FL, 26 April 1995.

[2] I. Cervesato, L. Chittaro, and A. Montanari. A general modal framework for the event calculus and its skeptical and credulous variants. Technical Report 37/96-RR, Dipartimento di Matematica e Informatica, Università di Udine, July 1996. Submitted for publication.

[3] I. Cervesato, M. Franceschet, and A. Montanari. A hierarchy of modal event calculi: Expressiveness and complexity. In H. Barringer, M. Fisher, D. Gabbay, , and G. Gough, editors, *Proceedings of the Second International Conference on Temporal Logic — ICTL'97*, pages 1–17, Manchester, England, 14–18 July 1997. Kluwer, Applied Logic Series. To appear.

[4] I. Cervesato, M. Franceschet, and A. Montanari. Modal event calculi with preconditions. In R. Morris and L. Khatib, editors, *Fourth International Workshop on Temporal Representation and Reasoning — TIME'97*, pages 38–45, Daytona Beach, FL, 10–11 May 1997. IEEE Computer Society Press.

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

[6] D. Miller. Lambda Prolog: An introduction to the language and its logic. Current draft available from `http://cse.psu.edu/~dale/lProlog`, 1996.

[7] D. Schroeder. *Staying Healthy in Asia, Africa and Latin America*. Moon publications, 1995.

[8] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. MIT Press, 1994.


## A Implementation of *QCEC*

```
module qcec.
accumulate transClo.

kind event    type.
kind property type.
kind mvi      type.

type initiates  event -> property -> o.
type terminates event -> property -> o.
type exclusive  property -> property -> o.
type happens    event -> o.
type prop       property -> o.

% ------- MVIs
type period  event -> property -> event -> mvi.
type holds   mvi -> o.
type broken  event -> property -> event -> o.

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 Q; terminates E Q),
        (exclusive P Q; P = Q).

% ------- Ordering
type precedes  event -> event -> mvi.  infixr precedes 6.

holds (E1 precedes E2) :-  before E1 E2.                     % 3 %

% ------- Connectives
type neg      mvi -> mvi.
type and      mvi -> mvi -> mvi.      infixr and     5.
type or       mvi -> mvi -> mvi.      infixr or      5.
type implies  mvi -> mvi -> mvi.      infixl implies 4.

holds (neg X) :-  not (holds X).                             % 4 %
holds (X and Y) :-  holds X, holds Y.                        % 5 %
holds (X or Y) :-  holds X; holds Y.                         % 6 %
holds (X implies Y) :-  holds ((neg X) or Y).                % 7 %

% ------- Quantifiers
type forAllEvent  (event -> mvi) -> mvi.
type forSomeEvent (event -> mvi) -> mvi.
type forAllProp   (property -> mvi) -> mvi.
type forSomeProp  (property -> mvi) -> mvi.

holds (forAllEvent X) :-                                     % 8 %
       not (sigma E \ (happens E, not (holds (X E)))).
holds (forSomeEvent X) :-  sigma E \ holds (X E).            % 9 %
holds (forAllProp X) :-                                     % 10 %
       not (sigma P \ (prop P, not (holds (X P)))).
holds (forSomeProp X) :-  sigma P \ holds (X P).            % 11 %
```

## B Diagnosing Malaria

```
module malaria.
accumulate qcec.

type fever   property.      prop fever.
type chills  property.      prop chills.
type malaria o.

malaria :- holds (forAllEvent E1 \
                  forAllEvent E2 \
                    ((period E1 chills E2) implies
                        (forSomeEvent E1' \
                         forSomeEvent E2' \
                           ((E1 precedes E1') and
                            (E1' precedes E2) and
                            (period E1' fever E2'))))).
```

## C Mr. Jones's Medical Record

```
module jones.
accumulate malaria.

type  e1 event.    happens  e1.    initiates   e1 chills.
type  e2 event.    happens  e2.    initiates   e2 fever.
type  e3 event.    happens  e3.    terminates  e3 chills.
type  e4 event.    happens  e4.    terminates  e4 fever.
type  e5 event.    happens  e5.    initiates   e5 chills.
type  e6 event.    happens  e6.    initiates   e6 fever.
type  e7 event.    happens  e7.    terminates  e7 chills.
type  e8 event.    happens  e8.    terminates  e8 fever.
type  e9 event.    happens  e9.    initiates   e9 chills.
type e10 event.    happens e10.    initiates  e10 fever.
type e11 event.    happens e11.    terminates e11 chills.
type e12 event.    happens e12.    terminates e12 fever.

beforefact  e1  e2.    beforefact  e2  e3.    beforefact e3  e4.
beforefact  e4  e5.    beforefact  e5  e6.    beforefact e6  e7.
beforefact  e7  e8.    beforefact  e8  e9.    beforefact e9 e10.
beforefact e10 e11.    beforefact e11 e12.
```