A Calculus of Macro-Events: Progress Report*

Iliano Cervesato
Advanced Engineering and Sciences Division
ITT Industries, Inc.
Alexandria, VA 22303-1410, USA
iliano@itd.nrl.navy.mil

Angelo Montanari
Dipartimento di Matematica e Informatica
Università di Udine
Via delle Scienze, 206 – 33100 Udine, Italy
montana@dimi.uniud.it

Abstract

The need of constraining the temporal relationships among sets of related events arises in several temporal reasoning tasks, including monitoring, plan validation, planning, and diagnosis. Process constructors provide an effective way of packaging up related events into individual conceptual chunks, called macro-events. In this paper, we present a first attempt at defining a Calculus of Macro-Events that extends Kowalski and Sergot's Event Calculus with process constructors to express effects triggered by complex combinations of event occurrences. We apply this language to model the operations of a simple gas heater, and present a Prolog *implementation.*

1 Introduction

Classical formalisms for reasoning about actions and change make the simplifying assumptions that (i) only one action can be performed at any given time (some formalisms actually allow simultaneous actions under the assumption that they do not influence each other), and (ii) actions are instantaneous. When we remove the first assumption, we must take into account that concurrent actions may interact. Interactions may lead both to synergistic results (their combined effect is more than the sum of their individual outcomes) and to interferences (their individual effects may be partially or totally canceled). For example [2], if one agent lifts one end of a piano, while another agent lifts the other end, then the entire piano is lifted off the floor; instead, if one agent pushes a door open, while the other is pushing it closed, the two actions cancel each other out. To handle these situations, many formalisms support explicit axioms to model interaction among concurrent actions, e.g. [4, 5, 13, 14, 18, 23, 26]. If we also remove the assumption that actions are instantaneous, occurrences of actions may partially overlap which can influence the outcome of the interacting actions [20].

However, the ability of dealing with concurrent and/or temporally extended actions is not sufficient for many realistic applications. Modeling real-world domains may indeed require representing complex patterns of actions, which, besides sequentiality and concurrency, involve additional relations among actions, such as the occurrence of just one action in a given set, the iteration of occurrences of the same action or of a pattern of actions. The notion of *process* has been introduced to define compound actions in terms of patterns of simpler actions. As an example, the action of dialing a ten-digit number can be defined as a set of ten basic actions in strict sequence. Foundational work on process modeling, which inspired research in many Computer Science areas, including knowledge representation, has been done by Hoare [15] and Milner [21]. Limited contributions to (discrete) process modeling in the temporal representation and reasoning area have been proposed by Evans [12], Belegrinos and Georgeff [6], Lespérance et al. [17], and by Lin and Dean [19].

In this paper, we present preliminary results on the definition of a Macro-Event Calculus, an extension of Kowalski and Sergot's Event Calculus, *EC* [16]. Our proposal allows expressing basic forms of process interaction, including temporal delays between processes, sequential, simultaneous, and alternative occurrences of processes, and process iteration. This proposal builds on work by Chittaro and Montanari [10] on modeling discrete processes. The set of constructors of the current version of the Macro-Event Calculus is similar to the path expression operators of [3]. Originally developed for modeling operating system behavior, path expressions have been successfully used in several areas of Computer Science. In this paper, we show how their formalization within the Event Calculus can be usefully

*The first author is supporting the Formal Methods Section of the Naval Research Laboratory under contract N0014-96-D2024. The second author was partially supported by the MURST project *Software Architectures and Languages to Coordinate Distributed Mobile Components*.

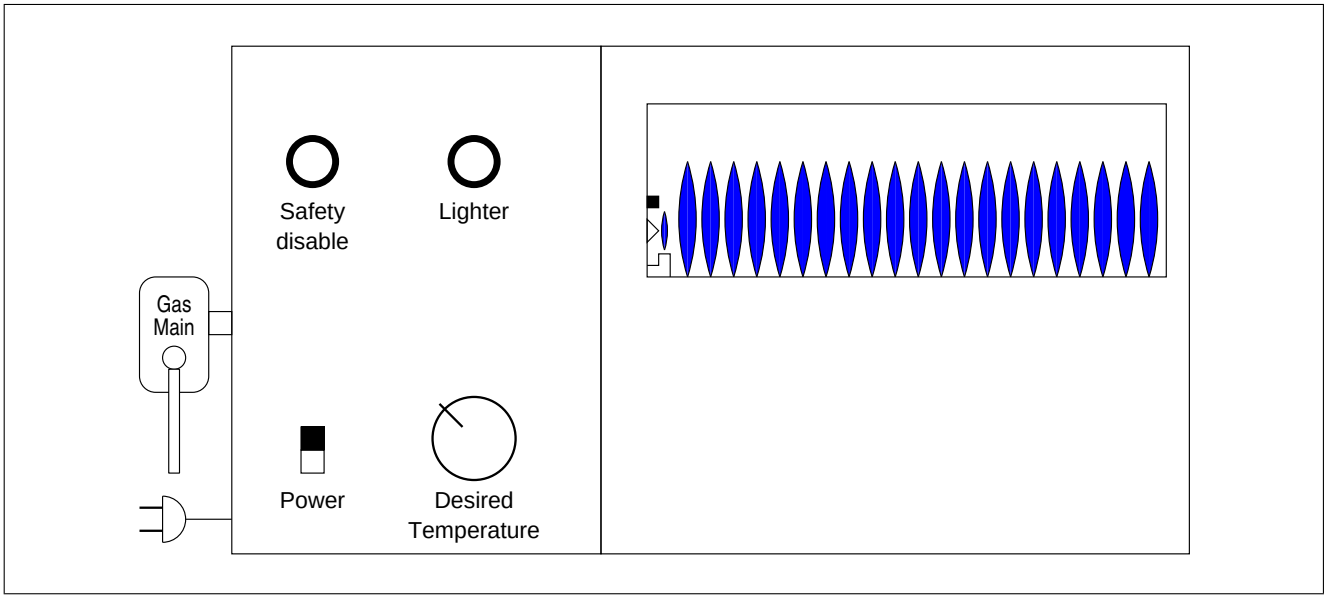

Figure 1. The Gas Heater: a User's Perspective

employed to reason about complex events. Compared to other approaches to the formalization of processes in temporal reasoning, our proposal is characterized by a logical bias: we aim at developing a calculus that gives a logical meaning to constructs which are operational in nature. We reconcile the two views by providing a logic programming implementation of the Macro-Event Calculus.

The paper is organized as follows: in Section 2, we describe the Gas Heater Problem, that we will use as our case study throughout this paper. In Section 3 we formalize versions of the Event Calculus that allow explicit time and event durations. In Section 4, we define macro-events and extend our specification of *EC* to handle them. We give a *Prolog* implementation of these various calculi in Section 5, and outline directions of future work in Section 6.

2 Case Study

We will illustrate the use of *EC* and of the proposed extensions by modeling some of the operations of a simple gas heater [10]. We now give an informal description of this case study, and later formalize it as we introduce concepts and definitions. In [22], we have successfully applied a variant of the Macro-Event Calculus to a larger-scale example, the Dagstuhl Steam-Boiler problem [1].

The gas heater presents to its user an interface (shown in Figure 1) consisting of two buttons (*Safety Disable* and *Lighter*) which must be used during the system start up procedure, a switch (*Power*) used to enable or disable system operation, a knob (*Desired Temperature*) used to select the desired temperature for the environment where the gas heater is placed, a tap (*Gas Main*) used to allow or prevent the supply of gas to the heater and a plug to supply electricity to the heater. The user typically starts up the system by: connecting the plug to a socket, opening the *Gas Main* tap, turning on the *Power* switch, pressing the *Safety Disable* button together with the *Lighter* button and keeping them pressed until he/she sees the pilot light turning on, releasing then the two buttons. When the pilot light is on, heating is automatically controlled by the system, which burns gas when the room temperature is lower than desired and keeps only the pilot light on otherwise (details below).

Looking inside the gas heater (Figure 2), six main components deserve attention. The *Power Switch* allows or prevents the supply of electrical power to the *Thermostat* and to the *Lighter System*. The *Lighter System* is devoted to producing sparks in order to light up the pilot light during the start up procedure. The *Safety System* prevents dangerous gas leaks into the environment: this thermo-mechanical device closes the *Safety Valve* when it is not heated by the pilot light and opens it when the pilot light is on. If the *Safety Disable* button is pressed, the *Safety Valve* allows gas to flow through the pipe connected to the pilot light (thus allowing, if needed, to ignite the pilot light with the *Lighter*) but not through the pipe connected to the *Thermostatic Valve*. The *Thermostat* senses room

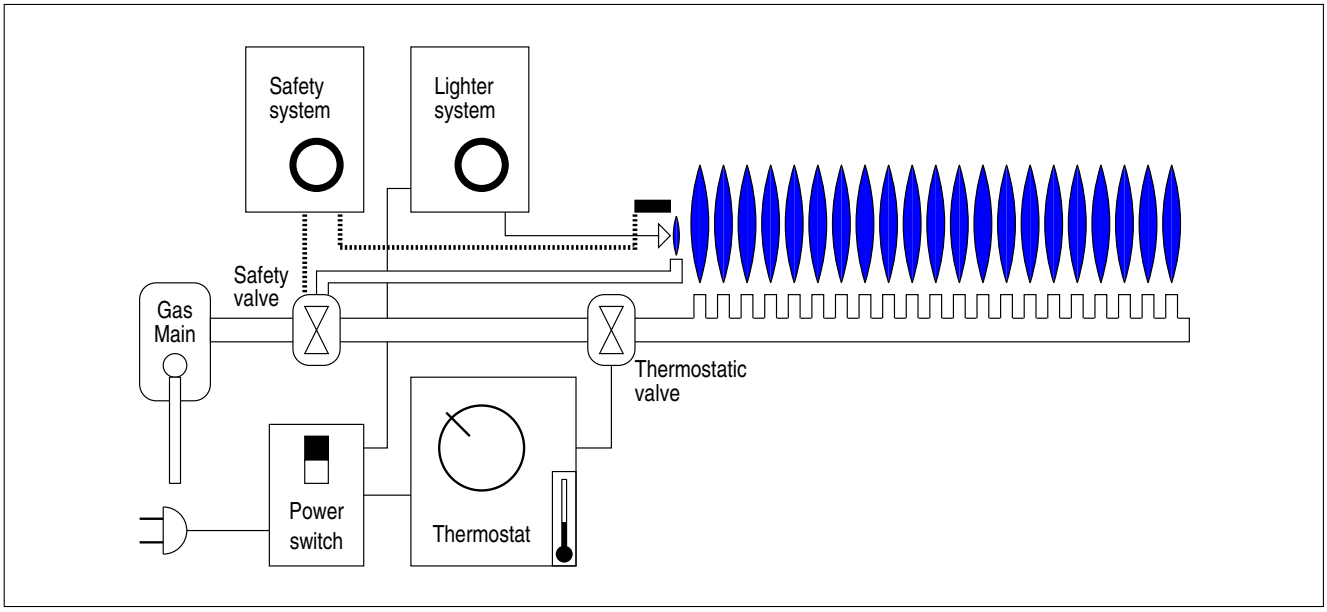

Figure 2. The Gas Heater: an Engineer's Perspective

temperature and opens the *Thermostatic Valve* if the temperature is one degree or less lower than the *Desired Temperature* preset by the user; it closes the valve when temperature is at least one degree higher than the *Desired Temperature*. When both the *Safety Valve* and the *Thermostatic Valve* are open, a huge quantity of gas is allowed to reach the main burner and be burnt, possibly ignited by the pilot light.

3 Explicit Time and Event Duration

We will now introduce the flavors of *EC* we will consider in this paper. In Section 3.1, we formalize *EC* with explicit time. In Section 3.2, we extend this model to admit non-instantaneous events. Finally, in Section 3.3, we apply these notions to our case study.

3.1 Explicit Time

While the original informal presentation of the Event Calculus [16] anchors the occurrence of instantaneous events to explicit time points, recent work on the formalization of *EC* [7, 8, 9] abstracted the time line and only considered events that are ordered relative to one another. We will extend this definition so to admit explicit time. We will operate on the formulation in [8], that embeds preconditions.

The *Event Calculus with Explicit Time and Instantaneous Events* (*ECT*) aims 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. The time-independent aspects of a situation are formalized by means of an *ECT-structure*. It alters the notion of *PEC*-structure given in [8] only by the addition of a temporal domain.

Definition 3.1 (*ECT-structure*)

A structure *for the* Event Calculus with Explicit Time (ECT-structure for short) *is a quintuple* $\mathcal{H} = (E,\ P,\ [\cdot|\cdot\rangle,\ \langle\cdot|\cdot],\ T)$ *such that:*

- $E = \{e_1, \ldots, e_n\}$ *and* $P = \{p_1, \ldots, p_m\}$ *are finite sets of* event types *and* properties*, respectively. Elements of* $\mathbf{2}^P$ *are called* contexts *and the properties in them are referred to as* preconditions*.*

- $[\cdot|\cdot\rangle : P \times \mathbf{2}^P \to \mathbf{2}^E$ *and* $\langle\cdot|\cdot] : P \times \mathbf{2}^P \to \mathbf{2}^E$ *are respectively the* initiating *and* terminating map *of* $\mathcal{H}$*. For every property* $p \in P$*,* $[p|C\rangle$ *and* $\langle p|C]$ *represent the set of events that initiate and terminate* p*, respectively, in case all preconditions in* C *hold at their occurrence time.*

- *The* temporal domain T *is some ordered set* $(\mathbb{T}, \prec)$*. In our implementation, we used the natural numbers* $\mathbb{N}$ *with their usual ordering.* □

The instance-specific part of a specification is captured through the following notion of *time-structure*:

Definition 3.2 (*Time-structure*)

Given an ECT-structure $\mathcal{H} = (E,\ P,\ [\cdot|\cdot\rangle,\ \langle\cdot|\cdot],\ T)$, *a* time-structure *for* $\mathcal{H}$ *is a set* $\mathcal{T} \subseteq E \times \mathbb{T}$ *of* event instances*, where a pair* $(e,t) \in \mathcal{T}$ *expresses the fact that an event of type* e *has occurred at time* t. □

Given the formalization of a situation in terms of an *ECT*-structure and a time-structure, *ECT* provides means to check whether a given property p is valid at a given point t in time [predicate $\mathsf{holdsAt}(p,t)$ below]. This problem is easily reduced to the determination of the *maximal validity intervals*, abbreviated MVI, over which a given property holds [$\mathsf{mvi}(p,t_1,t_2)$]: a property p holds maximally over an interval $[t_1,t_2]$ if *i)* $t_1 \prec t_2$, *ii)* an event e initiating p subject to preconditions C_1 has occurred at t_1 and all properties in C_1 hold at t_1 [$\mathsf{initiate}(p,t_1)$], *iii)* dually, an event e terminating p subject to preconditions C_2 has occurred at t_2 and all property in C_2 hold at t_2 [$\mathsf{terminate}(p,t_2]$, and *iv)* p is not initiated or terminated at any time point between t_1 and t_2 [$\neg\mathsf{broken}(p,t_1,t_2)$].

Definition 3.3 (*ECT-model*)

Let $\mathcal{H} = (E,\ P,\ [\cdot|\cdot\rangle,\ \langle\cdot|\cdot],\ T)$ *be a ECT-structure and* $\mathcal{T}$ *be a time-structure over* $\mathcal{H}$. *We define the following recursive predicates:*

$\mathsf{holdsAt}(p,t)$ *iff* $\exists t_1,t_2 \in \mathbb{T}.\ t_1 \preceq t \prec t_2 \ \wedge\ \mathsf{mvi}(p,t_1,t_2)$

$\mathsf{mvi}(p,t_1,t_2)$ *iff* $t_1 \prec t_2 \ \wedge\ \mathsf{initiate}(p,t_1) \ \wedge\ \mathsf{terminate}(p,t_2) \ \wedge\ \neg\mathsf{broken}(p,t_1,t_2)$

$\mathsf{initiate}(p,t)$ *iff* $\exists e \in E.\ \exists C \in \mathbf{2}^P.\ (e,t) \in \mathcal{T} \ \wedge\ p \in [e|C\rangle \ \wedge\ \forall q \in C.\ \mathsf{holdsAt}(q,t)$

$\mathsf{terminate}(p,t)$ *iff* $\exists e \in E.\ \exists C \in \mathbf{2}^P.\ (e,t) \in \mathcal{T} \ \wedge\ p \in \langle e|C] \ \wedge\ \forall q \in C.\ \mathsf{holdsAt}(q,t)$

$\mathsf{broken}(p,t_1,t_2)$ *iff* $\exists t \in \mathbb{T}.\ t_1 \prec t \ \wedge\ t \prec t_2 \ \wedge\ (\mathsf{initiate}(p,t) \ \vee\ \mathsf{terminate}(p,t))$. □

The meta-predicates holdsAt, mvi, initiate, terminate and broken are mutually recursive in the above definition. In particular, an attempt at checking properties or computing MVIs by simply unfolding their definition is non-terminating in pathological situations [8]. However, most *ECT* problems encountered in practice satisfy syntactic conditions ensuring the termination of this procedure. See [8] for a detailed discussion of these restrictions.

The above definition can be directly transcribed in the programming language *Prolog* [25]. The resulting code is given in Section 5.1.

3.2 Event Duration

Although thinking of events as instantaneous is a sufficient abstraction for some situations, in many cases the occurrence of an event happens over a period of time [24]. Capturing this possibility enables finer models, as we can now reason about changes that take place while an event is in the process of occurring. In particular, we must revise our notion of precondition as it can now have at least two meanings: a property that should hold in order for an event to start happening, or a property that ought to be uninterruptedly valid during the entire duration of the occurrence of an event. In our formalization below, we will allow both options, keeping the name precondition for the former, and calling the latter *constraint* [10].

Definition 3.4 (*ECTD-structure*)

A structure *for the* Event Calculus with Event Duration (ECTD-structure for short) *is a sextuple* $\mathcal{H} = (E,\ P,\ [\cdot|\cdot|\cdot\rangle,\ \langle\cdot|\cdot|\cdot],\ T,\ \delta)$ *such that:*

- E *and* P *are the sets of event types and properties, respectively.*
- $[\cdot|\cdot|\cdot\rangle$: $P \times \mathbf{2}^P \times \mathbf{2}^P \to \mathbf{2}^E$ *and* $\langle\cdot|\cdot|\cdot]$: $P \times \mathbf{2}^P \times \mathbf{2}^P \to \mathbf{2}^E$ *are respectively the initiating and terminating map of* $\mathcal{H}$. *For every property* $p \in P$, $[p|C|K\rangle$ *and* $\langle p|C|K]$ *represent the set of events that initiate and terminate* p*, respectively, in case all preconditions in* C *hold at the time they start occurring, and all* constraints *in* K *hold for the entire duration of their occurrence.*
- $T = (\mathbb{T}, \prec, +, 0)$ *is the* temporal domain*, where the ordered set* $(\mathbb{T}, \prec)$ *considered above is augmented with a monoid component* $(\mathbb{T}, +, 0)$.
- $\delta : E \to \mathbb{T}$ *is a* duration function*, that gives the duration of every event type.* □

Observe that *ECT*-structures are degenerate *ECTD*-structures where no constraints are present, and $\delta(e) = 0$ for every event $e \in E$. We do not need to modify the definition of time-structure.

In spite of the drastic changes in Definition 3.4 with respect to the notion of *ECT*-structure, the modifications to the specification of how to check whether a property holds at a given time point are localized to the predicates initiate and terminate. Let us focus on the former: in order for $\mathsf{initiate}(p,t)$ to hold, there must be an event e with preconditions C and constraints K that initiates p. If e starts occurring at time t', it must be the case that t is the endpoint of the occurrence

interval of e (*i.e.* $t = t' + \delta(e)$). Moreover, every precondition in C must hold at its startpoint t', and every constraint in K must be valid without interruption throughout the interval $[t', t]$. Notice in particular that an event has effectively initated a property at the end of its occurrence interval. Similar considerations apply to the case of terminate. This is captured in the following definition, where the predicates holdsDuring and happens are accessory.

Definition 3.5 (*ECTD-model*)

Let $\mathcal{H} = (E,\ P,\ [\cdot|\cdot|\cdot\rangle,\ \langle\cdot|\cdot|\cdot],\ T,\ \delta)$ *be a ECTD-structure and* $\mathcal{T}$ *be a time-structure over* $\mathcal{H}$. *We modify Definition 3.3 by overriding the specification of* initiate *and* terminate *given there with the clauses below, and adding the predicates* happens *and* holdsDuring.

$$\begin{array}{l}\mathsf{initiate}(p,t) \qquad \textit{iff} \\ \quad \exists e \in E.\ \exists t', d \in \mathbb{T}.\ \exists C, K \in \mathbf{2}^P. \\ \qquad \mathsf{happens}(e, t', d) \ \ \wedge \ \ t = t' + d \ \ \wedge \\ \qquad p \in [e|C|K\rangle \ \ \wedge \ \ \forall q \in C.\ \mathsf{holdsAt}(q, t') \ \ \wedge \\ \qquad \forall q \in K.\ \mathsf{holdsDuring}(q, t', t)\end{array}$$

$$\begin{array}{l}\mathsf{terminate}(p,t) \qquad \textit{iff} \\ \quad \exists e \in E.\ \exists t', d \in \mathbb{T}.\ \exists C, K \in \mathbf{2}^P. \\ \qquad \mathsf{happens}(e, t', d) \ \ \wedge \ \ t = t' + d \ \ \wedge \\ \qquad p \in \langle e|C|K] \ \ \wedge \ \ \forall q \in C.\ \mathsf{holdsAt}(q, t') \ \ \wedge \\ \qquad \forall q \in K.\ \mathsf{holdsDuring}(q, t', t)\end{array}$$

$$\mathsf{happens}(e, t, d) \qquad \textit{iff} \ \ (e, t) \in H \ \ \wedge \ \ d = \delta(e)$$

$$\begin{array}{l}\mathsf{holdsDuring}(p, t_1, t_2) \ \textit{iff} \ \ \exists t_0, t_3 \in \mathbb{T}.\ \ \mathsf{mvi}(p, t_0, t_3) \ \ \wedge \\ \qquad\qquad\qquad\qquad t_0 \preceq t_1 \ \ \wedge \ \ t_2 \preceq t_3. \qquad \square\end{array}$$

It is easy to verify that, whenever $\mathcal{H}$ corresponds to an ECT-structure, the definition reduces to Definition 3.1.

It is interesting to observe that, in the absence of constraints, the Event Calculus with Event Duration can be *compiled* into the formalism presented in Section 3.1. Every lasting event e is translated into a pair of instantaneous events *start_e* and *end_e*, and a property *occ_e*. Instances (e, t) are mapped to the instantaneous instances $(start\_e, t)$ and $(end\_e, t + \delta(e))$. Finally, if $p \in [e|C|\cdot\rangle$, we get $occ\_e \in [start\_e|C\rangle$, moreover $occ\_e \in \langle end\_e|\cdot]$, finally $p \in [end\_e|\cdot\rangle$. The termination map is processed analogously. The treatment of constraints is problematic because of the requirement that they ought not to be interrupted during the event's occurrence interval.

Prolog code implementing Definition 3.5 is given in Section 5.2.

3.3 Example — Part I

The first phase of the representation of the gas heater in EC consists in the identification of the relevant types of events. We consider eleven types: eight of them represent possible actions of the user of the gas heater, i.e. open or close the *Main Gas* tap (gasOn, gasOff), turn on or off the *Power* switch (powerOn, powerOff), press or release the *Safety Disable* button (prDisable, relDisable), and press or release the *Lighter* button (prLighter, relLighter). In this simple model, we will not represent the action of setting the *Thermostat* to the desired temperature. Two other types of event represent the possible temperature changes in the environment as reported by a *Thermometer* (coolDown, warmUp). In order to represent the properties that are initially valid in the system [11], we add the fictitious event start which represent the beginning of time. A possible time structure $\mathcal{T}$ built using the above types of events is the following:

(start,	0).	(relDisable,	6).
(coolDown,	0).	(warmUp,	8).
(gasOn,	1).	(coolDown,	10).
(powerOn,	2).	(warmUp,	11).
(prDisable,	3).	(powerOff,	18).
(prLighter,	3).	(gasOff,	19).
(relLighter,	5).	(coolDown,	25).

In this scenario, the user notices a drop in temperature (time 0) and takes all the actions needed in order to ignite the pilot light: she opens the *Gas Main* (time 1), switches the power on (time 2), and presses the *Lighter* and *Safety Disable* buttons simultaneously (time 3). She releases these buttons at time 5 and 6, respectively. Between time 8 and 11 the thermometer reports some temperature changes in the environment. At time 18 the user interrupts the supply of gas, and shortly after she turns the power off. Eventually, the rooms becomes cold again (time 25).

The second step in the representation of the gas heater is the identification of the interesting properties that are initiated and terminated by events. We model this system by means of nine properties: whether gas is flowing into the heater (gas), whether electrical power is supplied (power), whether the room is cold (cold), whether the thermostatic and the safety valves are open (thermoVOpen and safetyVOpen, respectively), whether the lighter system produces sparkles (sparkling), whether the pilot light is on (pilotOn), and whether gas is burning in the main combustion chamber (burning). Finally, it will be convenient to know when the pilot light is off (pilotOff).

These properties fall in three classes: properties ini-

tiated or terminated by the simple occurrence of an event; properties initiated or terminated by the occurrence of an event in a specific context; properties initiated or terminated by a combination of events.

Properties gas, power, cold, and thermoVOpen fall in the first class: they are unconditionally initiated and terminated by the events gasOn and gasOff, powerOn and powerOff, coolDown and warmUp, and coolDown and warmUp, respectively. This is captured in *EC* as follows:

$$\begin{array}{lcl}
[\mathsf{gas}|\cdot|\cdot\rangle & = & \{\mathsf{gasOn}\},\\
[\mathsf{power}|\cdot|\cdot\rangle & = & \{\mathsf{powerOn}\},\\
[\mathsf{cold}|\cdot|\cdot\rangle & = & \{\mathsf{coolDown}\},\\
[\mathsf{thermoVOpen}|\cdot|\cdot\rangle & = & \{\mathsf{coolDown}\},\\
\langle\mathsf{gas}|\cdot|\cdot] & = & \{\mathsf{gasOff}\}\\
\langle\mathsf{power}|\cdot|\cdot] & = & \{\mathsf{powerOff}\}\\
\langle\mathsf{cold}|\cdot|\cdot] & = & \{\mathsf{warmUp}\}\\
\langle\mathsf{thermoVOpen}|\cdot|\cdot] & = & \{\mathsf{warmUp}\}
\end{array}$$

The second class consists of the properties sparking and safetyVOpen. While the former simply requires that power be supplied when the lighter button is pressed, the latter has a more complex behavior. Consider first how to terminate the property safetyVOpen: if the pilot light is off, releasing the safety button is sufficient to close the valve; however if this is not the case, the only way to shut the safety valve is by extinguishing the pilot light, which is achieved, as we will see, by interrupting the supply of gas. We have the following formalization:

$$\begin{array}{lcl}
[\mathsf{sparking}|\cdot|\mathsf{power}\rangle & = & \{\mathsf{prLighter}\}\\
[\mathsf{safetyVOpen}|\cdot|\cdot\rangle & = & \{\mathsf{prDisable}\}\\
\langle\mathsf{sparking}|\cdot|\cdot] & = & \{\mathsf{relLigher}, \mathsf{powerOff}\}\\
\langle\mathsf{safetyVOpen}|\cdot|\mathsf{pilotOn}] & = & \{\mathsf{gasOff}\}\\
\langle\mathsf{safetyVOpen}|\cdot|\mathsf{pilotOff}] & = & \{\mathsf{relDisable}\}
\end{array}$$

The remaining properties pilotOn and pilotOff, which records whether the pilot light is on or off, depends on constraints such as the availability of power and gas. However, they are initiated and terminated respectively by the simultaneous occurrence of two events (prDisabled and prLighter). Similar remarks apply to the property burning. The extensions to *EC* we devised so far are insufficient to specify these situations.

4 Event Calculus with Macro-Events

As we just saw, even when events with durations are available, *EC* does not lend itself to easily expressing situations where properties are initiated or terminated not by single events, but by the occurrence of multiple events. We will address this shortcoming of *ECTD* by allowing the validity status of properties to be changed on the basis of the occurrence of structured conglomerations of events that we will call *macro-events*. In Section 4.1, we define this notion and extend *ECTD*-structures to accommodate it. It takes into account the cumulative effects of a set of related events, but for simplicity, excludes interference issues. In Section 4.2, we simultaneously address the problems of determining whether a macro-event has occurred, and extend the specification of validation of a property (Definition 3.5) to these entities. In Section 4.3, we complete the treatment of our case-study.

4.1 Definition

In our current model, macro-events are obtained by considering sequential, alternative, parallel, or iterated occurrences of elementary events, or any combination of these constructions.

Definition 4.1 (*Macro-Events*)

Given a set of events E and a temporal domain $T = (\mathbb{T}, \prec, +, 0)$, macro-events*, denoted with m possibly subscripted, are expressions defined by the following grammar:*

$$\begin{array}{lcll}
m & ::= & e & (\textit{Basic event})\\
 & | & m_1 ;_d^D m_2 & (\textit{Sequence with delay } d \textit{ to } D)\\
 & | & m_1 + m_2 & (\textit{Alternative})\\
 & | & m_1 \,||\, m_2 & (\textit{Parallelism})\\
 & | & m^* & (\textit{Iteration})
\end{array}$$

where $d, D \in \mathbb{T}$ and $d \preceq D$. Let $\mathcal{M}_T$ be the set of the macro-events over T. □

This definition formalizes the core of the notion of process studied at length in [10, 22], which in turn extends the limited notion of macro-events (essentially delayed sequencing) presented in [12].

The constructors we included in this language are based on the path expression operators of [3] and on the process calculi operators found in [15, 21]. Observe that a number of useful constructs are easily expressible with the language in Definition 4.1. In particular sequencing with arbitrary delay (;), immediate sequencing (;;), non-empty iteration ($\cdot^+$) and fixed-length iteration ($\cdot^n$) are defined as follows in [22]:

$$\begin{array}{rcll}
m_1; m_2 & = & m_1 ;_0^\infty m_2 & \\
m^+ & = & m; m^* & \\
m_1;; m_2 & = & m_1 ;_0^0 m_2 & \\
m^n & = & m; \ldots; m & (n \text{ times})
\end{array}$$

$$
\begin{array}{lll}
\mathsf{me}(e,[t_1,t_2],s,l) & \text{iff} & (e,s)\in\mathcal{T} \ \wedge \ l=\delta(e) \ \wedge \ [s,s+l]\preceq[t_1,t_2] \\
\mathsf{me}(m_1\ ;_d^D\ m_2,[t_1,t_2],s,l) & \text{iff} & \exists t_1',t_2',l_1,l_2\in\mathbb{T}.\ \mathsf{me}(m_1,[t_1,t_1'],s,l_1) \ \wedge \ \mathsf{me}(m_2,[t_2',t_2],s_2,l_2) \ \wedge \\
 & & \quad s+l_1\preceq t_1'\preceq t_2'\preceq s_2 \ \wedge \ s+l_1+d\preceq s_2\preceq s+l_1+D \ \wedge \\
 & & \quad l=s_2+l_2-s \\
\mathsf{me}(m_1+m_2,[t_1,t_2],s,l) & \text{iff} & \mathsf{me}(m_1,[t_1,t_2],s,l) \ \vee \ \mathsf{me}(m_2,[t_1,t_2],s,l) \\
\mathsf{me}(m_1\,||\,m_2,[t_1,t_2],s,l) & \text{iff} & \exists s_1,l_1,s_2,l_2\in\mathbb{T}.\ \mathsf{me}(m_1,[t_1,t_2],s_1,l_1) \ \wedge \ \mathsf{me}(m_2,[t_1,t_2],s_2,l_2) \ \wedge \\
 & & \quad s=\mathsf{min}(s_1,s_2) \ \wedge \ l=\mathsf{max}(s_1+l_1,s_2+l_2)-s \\
\mathsf{me}(m^*,[t_1,t_2],s,l) & \text{iff} & \exists l_1,s_2,l_2\in\mathbb{T}.\ (s=t_1 \ \wedge \ l=0) \ \vee \\
 & & \quad (\mathsf{me}(m,[t_1,t],s,l_1) \ \wedge \ \mathsf{me}(m^*,[t,t_2],s_2,l_2) \ \wedge \\
 & & \quad s+l_1\preceq t\preceq s_2 \ \wedge \ l=s_2+l_2-s)
\end{array}
$$

Figure 3. Definition of me

Before giving the exact semantics of macro-events, we update our formalization of *ECTD* of Section 3.2 to accomodate these entities. The changes to the definition of *ECTD*-structure turn out to be very modest.

Definition 4.2 (*MECTD-structure*)

A structure *for the* Macro-Event Calculus (MECTD-structure for short) *is a septuple* $\mathcal{H} = (E,\ P,\ M,\ [\cdot|\cdot|\cdot\rangle,\ \langle\cdot|\cdot|\cdot],\ T,\ \delta)$ *which differs from the definition of* ECTD*-structure only by the following points:*

- $M \subseteq \mathcal{M}_T$ *is a set of macro-events over* T*.*
- *The codomain of* $[\cdot|\cdot|\cdot\rangle$ *and* $\langle\cdot|\cdot|\cdot]$ *are redefined to be* $\mathbf{2}^M$*: indeed* $[\cdot|\cdot|\cdot\rangle : P\times\mathbf{2}^P\times\mathbf{2}^P\to\mathbf{2}^M$ *and* $\langle\cdot|\cdot|\cdot] : P\times\mathbf{2}^P\times\mathbf{2}^P\to\mathbf{2}^M$*. This implies that properties can be started and ended by generic macro-events, not just plain events.*
- *We assume that the temporal domain* $(\mathbb{T},\prec)$ *has a maximum element* ∞*, and that* $(\mathbb{T},+,0)$ *is a group, with* $-$ *the inverse operation of* $+$*.* □

Observe that we did not propagate the change to the duration function: only basic events are explicitly given a duration by means of δ. The duration of occurrences of macro-events will instead be computed on the basis of their structure and of the participating basic events.

We do not modify our notion of time-structure $\mathcal{T}$: only elementary events are recorded. Occurrences of macro-events will instead be inferred.

4.2 Monitoring and Evaluation

The occurrence of a macro-event is not explicitely recorded, but must be determined on the basis of its definition and of the time-structure at hand. In order to do so, we define the auxiliary predicate $\mathsf{me}(m,[t_1,t_2],s,l)$, which verifies whether a macro-event m has occurred over the interval $[t_1,t_2]$, and, if this is the case, computes its starting point s and duration l. These two arguments are necessary to correctly process the bounds of a delayed sequence of events. Given a *MECTD*-structure $\mathcal{H} = (E,\ P,\ M,\ [\cdot|\cdot|\cdot\rangle,\ \langle\cdot|\cdot|\cdot],\ T,\ \delta)$ and a time structure $\mathcal{T}$ on $\mathcal{H}$, this predicate is recursively defined in Figure 3, where we have promoted $\preceq$ to denote the sub-interval relation over T.

Observe that me defines the semantics of the macro-event constructors presented in Definition 4.1. In the base case of the recursion, i.e. if m is an event e, we verify if an ocurrence of e has been recorded in the time-structure $\mathcal{T}$. If so, we check that it takes place over a subinterval of $[t_1,t_2]$. In the case of a sequence, we must make sure that the endpoint of the first component and the starting time of the second are within the acceptable delay. Clearly, they must take place over sequentially disjoint intervals. In order to verify that an alternative macro-event has occurred, we look for the occurrence of either component. Parallel macro-events must have both occurred over the same interval. Observe that we do not require that the two branches mention distinct events; indeed $m\,||\,m$ is equivalent to m. Finally, iterated macro-events are essentially reduced to (possibly empty) sequences. Notice that we do not force any form of maximality: the empty iteration is always satisfied; its starting point is made to coincide with the beginning of the test interval, and it always has null duration.

We check whether a given macro-event has occurred over some interval by using the above definition, while abstracting from the starting time and duration.

Definition 4.3 (*Monitoring Macro-Events*)

Let $\mathcal{H}$ *be a MECTD-structure and* $\mathcal{T}$ *be a time-structure over* $\mathcal{H}$*. We say that a macro-event* m *has occurred over an interval* $[t_1,t_2]$*, written*

$\mathsf{check}(m, [t_1, t_2])$, *iff*

$$\exists s, l \in \mathbb{T}.\ \mathsf{me}(m, [t_1, t_2], s, l)$$

is valid. □

The starting point and duration of macro-events are useful in order to compute MVIs, and ultimately to check whether a given property holds at a certain point in time. Therefore, me offers a way to update the predicate happens from Definition 3.5 to operate over generic macro-events.

Definition 4.4 (*MECTD-model*)

Let $\mathcal{H}$ be a MECTD-structure and $\mathcal{T}$ be a time-structure over $\mathcal{H}$. We modify Definition 3.5 by overriding the definition of happens *given there with the following clause:*

$\mathsf{happens}(m, t, d)$ *iff* $\mathsf{me}(m, [0, \infty], t, d)$ □

This definition provides an elegant specification to the core of the system presented in [10]. We limited ourselves to treating the situation where macro-events can initiate or terminate properties. This is sufficient for many applications, and involves much simpler definitions than the general case. We did not include here the possibility of a macro-event that cancels effects caused by some of its components (be it an elementary event, or a macro-event). A complete specification can be found in [22], and will be included in an extended version of this paper.

An implementation of these specifications can be found in Section 5.3.

4.3 Example — Part II

Macro-events are a convenient tool to complete the specification of the example in Section 2. Continuing from Section 3.3, we can now express the validity of properties pilotOn, pilotOff, and burning:

$$\begin{aligned}
[\mathsf{pilotOn}|\cdot|\mathsf{power}, \mathsf{gas}\rangle &= \{\mathsf{prLighter} \,||\, \mathsf{prDisable}\} \\
[\mathsf{pilotOff}|\cdot|\cdot\rangle &= \{\mathsf{start}\} \\
[\mathsf{pilotOff}|\cdot|\mathsf{pilotOn}\rangle &= \{\mathsf{gasOff}\} \\
[\mathsf{burning}|\cdot|\mathsf{pilotOn}\rangle &= \{\mathsf{coolDown}\} \\
[\mathsf{burning}|\cdot|\mathsf{cold}, \mathsf{power}, \mathsf{gas}\rangle &= \{\mathsf{prLighter} \,||\, \mathsf{prDisable}\} \\
\langle\mathsf{pilotOn}|\cdot|\cdot] &= \{\mathsf{gasOff}\} \\
\langle\mathsf{pilotOff}|\cdot|\mathsf{power}, \mathsf{gas}] &= \{\mathsf{prLighter} \,||\, \mathsf{prDisable}\} \\
\langle\mathsf{burning}|\cdot|\mathsf{pilotOn}] &= \{\mathsf{warmUp}\} \\
\langle\mathsf{burning}|\cdot|\cdot] &= \{\mathsf{gasOff}\}
\end{aligned}$$

Property burning has two initiation clauses. The first applies when the pilot light is lit: if the temperature drops below the thermostatic threshold, gas will start burning. The second handles the case where the room is cold at the moment in which the pilot light is ignited. This property has an equally interesting termination behavior: it can always be ended by cutting the gas supply, but if the pilot light is on it is sufficient that the room temperature warms up above the threshold set through the thermostat.

This concludes the specification of the properties of interest.

It is worth observing that the overall specification of the gas heater could be considerably shortened by admitting the notions of *auto-initiation* and *auto-termination* [10] to our calculus. An auto-initiated property (here burning is a good candidate) is explicitly initiated not by the occurrence of events, but as soon as the validity periods of one or more other properties start overlapping (here pilotOn and cold). Auto-termination is defined in a dual way. We plan to include these constructs in a forthcoming version of the macro-event calculus.

5 Implementation

We will now give a *Prolog* [25] implementation of the calculi we have presented so far and an encoding of our case study. We assume the reader is familiar with this logic programming language.

5.1 EC with Explicit Time

We represent the initiation and termination maps, $[\cdot|\cdot\rangle$ and $\langle\cdot|\cdot]$ of an *ECTD*-structure by means of the *Prolog* predicates `init` and `term`, respectively. We use *Prolog*'s lists to represent the preconditions of the effect of an event. We adopt the integers as the temporal domain T. The precedence relation $\prec$ is then mapped to `<`. Each pair (e, t) in a time-structure is represented as the fact `happens(`$\ulcorner e \urcorner$`,`$\ulcorner t \urcorner$`)`, where $\ulcorner e \urcorner$ and $\ulcorner t \urcorner$ encode e and t respectively. For aesthetic reasons, we represent an interval $[t_1, t_2]$ with the two-element list `[`$\ulcorner t_1 \urcorner$`,`$\ulcorner t_2 \urcorner$`]`.

The contents of Definition 3.3 are then transcribed as follows in *Prolog*, where we have kept the predicate name unchanged.

```
holdsAt(P, T) :-
     mvi(P, [T1,T2]),
     T1 =< T,  T < T2.

mvi(P, [T1,T2]) :-
     initiate(P, T1),  terminate(P, T2),
     T1 < T2,
     \+ broken(P, [T1,T2]).
```

```
mHoldsAt([],_).
mHoldsAt([P|C], T) :-
     holdsAt(P, T),
     mHoldsAt(C, T).

initiate(P, T) :-
     init(E, P, C),
     happens(E, T),
     mHoldsAt(C, T).

terminate(P,T) :-
     term(E, P, C),
     happens(E, T),
     mHoldsAt(C, T).

broken(P, [T1,T2]) :-
     (initiate(P, T)  ;  terminate(P, T)),
     T1 < T,  T < T2.
```

Whenever the syntactic conditions [8] mentioned in Section 3.1 are met, this program allows not only verifying the validity of a property at a given time point t, but also computing all the properties that hold at t. Using the technique in [9], it is possible to prove the soundness and completeness of this program with respect to Definition 3.3.

5.2 EC with Explicit Time and Event Duration

We add one argument to `init` and `term` to the previous representation to encode the constraints that distinguish $[\cdot|\cdot|\cdot\rangle$ and $\langle\cdot|\cdot|\cdot]$ in an *ECTD*-structure. Moreover, we rely on *Prolog*'s arithmetic to emulate the operation now available in the temporal domain. Finally, we rely on the predicate `lasts` to model the duration function δ.

The behavior of *ECTD* is captured by replacing the clauses for `initiate` and `terminate` with the following definitions, and adding the accessory predicates `happens`, `mHoldsDuring`, and `holdsDuring`.

```
initiate(P,TD) :-
     init(E, P, C, K),
     happens(E, T, D),
     TD is T + D,
     mHoldsAt(C, T),
     mHoldsDuring(K, [T,TD]).

terminate(P, TD) :-
     term(E, P, C, K),
     happens(E, T, D),
     TD is T + D,
     mHoldsAt(C, T),
     mHoldsDuring(K, [T,TD]).

happens(E, P, D) :-
     happens(E, P),  lasts(E, D).

mHoldsDuring([], _).
mHoldsDuring([P|C], I) :-
     holdsDuring(P, I),
     mHoldsDuring(C, I).

holdsDuring(P, [T1,T2]) :-
     mvi(P, [T0,T3]),
     T0 =< T1,  T2 =< T3.
```

Again, this implementation can be proved sound and complete with respect to Definition 3.4.

5.3 Macro-Event Calculus

We encode the process constructions $m_1 ;_d^D m_2$, $m_1 + m_2$, $m_1 \parallel m_2$ and m^* by means of the *Prolog* terms `seq(`$\ulcorner m_1 \urcorner$`,`$\ulcorner m_2 \urcorner$`,`$\ulcorner d \urcorner$`,`$\ulcorner D \urcorner$`)`, `alt(`$\ulcorner m_1 \urcorner$`,`$\ulcorner m_2 \urcorner$`)`, `par(`$\ulcorner m_1 \urcorner$`,`$\ulcorner m_2 \urcorner$`)`, and `it(`$\ulcorner m \urcorner$`)`, respectively. Lacking a better abstraction, we used `100000` for ∞.

We implement the predicate `me` and Definitions 4.3-4.4 by replacing the clause for `happens`/3 above with the following code. The convoluted definition for `subinterval` is due to the fact that it is used both for checking that an interval is contained in another interval, but also to set either endpoints of the former.

```
me(E, I, S, L) :-
     happens(E, S),
     lasts(E, L),
     T is S + L,
     subinterval([S,T], I).

me(seq(M1,M2,Min,Max), [T1,T2], S, L) :-
     me(M1, [T1,T11], S, L1),
     me(M2, [T22,T2], S2, L2),
     S + L1 =< T11,  T11 =< T22,  T22 =< S2,
     Min =< (S2 - S - L1),  (S2 - S - L1) =< Max,
     L is S2 + L2 - S.

me(alt(M1,M2), I, S, L) :-
     (me(M1, I, S, L) ;
      me(M2, I, S, L)).

me(par(M1,M2), I, S, L) :-
     me(M1, I, S1, L1),
     me(M2, I, S2, L2),
     S is min(S1,S2),
     L is max(S1+L1,S2+L2) - S.

me(it(M), [T1,T2], S, L) :-
     me(M, [T1,T], S, L1),
     me(it(M), [T,T2], S2, L2),
     S+L1 =< T, T  =< S2,
     L is S2 + L2 - S.
```

```
me(it(_), [T,T], T, 0) :- !.
me(it(_), [T,_], T, 0).


check(M, I) :-
     me(M, I, _, _).


happens(M, T, D) :-
     me(M, [0,100000], T, D).


subinterval([B1,E1], [B2,E2]) :-
      ((var(B2), B1 = B2, !)  ;  B2 =< B1),
      ((var(E2), E1 = E2, !)  ;  E1 =< E2).
```

Showing the soundness and completeness of this implementation with respect to the specifications given in Section 4.2 is complicated by the non-logical implementation of `subinterval`. However, once this predicate has been processed in isolation, standard techniques from [9] can be successfully applied.

5.4 Example — Part III

We will now complete the treatment of the gas-heater example by displaying the clauses that encode the associated *MECTD*-structure and time structure. The latter is immediately rendered by the following facts:

```
happens(start,      0).   happens(relDisable, 6).
happens(coolDown,   0).   happens(warmUp,     8).
happens(gasOn,      1).   happens(coolDown,   10).
happens(powerOn,    2).   happens(warmUp,     11).
happens(prDisable,  3).   happens(powerOff,   18).
happens(prLighter,  3).   happens(gasOff,     19).
happens(relLighter, 5).   happens(coolDown,   25).
```

where we have kept the name of the events (and below properties) as in the body of this paper.

In our example, all events are instantaneous. Therefore, the clause

```
lasts(E, 0).
```

summarizes all the relevant information about event duration.

The initiation and termination maps relative to the gas heater problem are given by the following code, where we have grouped these facts by the property being initiated or terminated:

```
init(gasOn,      gas,         [], []).
term(gasOff,     gas,         [], []).

init(powerOn,    power,       [], []).
term(powerOff,   power,       [], []).

init(coolDown,   cold,        [], []).
term(warmUp,     cold,        [], []).

init(coolDown,   thermoVOpen, [], []).
term(warmUp,     thermoVOpen, [], []).

init(prLighter,  sparking,    [], [power]).
term(relLighter, sparking,    [], []).
term(powerOff,   sparking,    [], []).

init(prDisable,  safetyVOpen, [], []).
term(gasOff,     safetyVOpen, [], [pilotOn]).
term(relDisable, safetyVOpen, [], [pilotOff]).

init(coolDown,   burning,     [], [pilotOn]).
init(par(prLighter,prDisable),
                 burning,     [], [cold,power,gas]).
term(warmUp,     burning,     [], [pilotOn]).
term(gasOff,     burning,     [], []).

init(par(prLighter,prDisable),
                 pilotOn,     [], [power,gas]).
term(gasOff,     pilotOn,     [], []).

init(start,      pilotOff,    [], []).
init(gasOff,     pilotOff,    [], [pilotOn]).
term(par(prLighter,prDisable),
                 pilotOff,    [], [power,gas]).
```

This completes the formalization of the gas heater. We can now use it to extract information that is implicit in the example. The following query retrieves the maximum validity intervals of all the properties that appear in this case study.

```
?- mvi(M, I).

M = gas            I = [1, 19]     ;

M = power          I = [2, 18]     ;

M = cold           I = [0, 8]      ;
M = cold           I = [10, 11]    ;

M = thermoVOpen    I = [0, 8]      ;
M = thermoVOpen    I = [10, 11]    ;

M = sparking       I = [3, 5]     ;

M = safetyVOpen    I = [3, 19]     ;

M = burning        I = [10, 11]    ;
M = burning        I = [3, 8]      ;

M = pilotOn        I = [3, 19]     ;
```

```
M = pilotOff       I = [0, 3]      ;

No
```

We can also inquire whether a given macro-event has occurred, when it started, and how long it lasted. For example, if we want to know if it ever happened that the room first got cold and then warm within 20 time units, the following query will provide three answers:

```
?- happens(seq(coolDown,warmUp,0,20), T,D ).

T = 0     D = 8     ;
T = 0     D = 11    ;
T = 10    D = 1     ;

No
```

Observe that the first interval embeds the second. This is acceptable since, differently from the evaluation of *MVI*-goals, queries about macro-events do not involve maximality checks.

Finally, we show a use of the predicate `check`, which verifies whether a given macro-event has occurred in a given interval.

```
?- check(seq(coolDown,warmUp,1,5), [3,19]).

Yes
```

6 Conclusions and Future Work

In this paper, we presented a preliminary attempt at defining a Calculus of Macro-Events that extends Kowalski and Sergot's Event Calculus with primitives (process constructors) for modeling discrete processes. In particular, we showed how to infer the occurrence of macro-events from the occurrence of their atomic components (monitoring) as well as how to derive the maximal validity intervals of properties initiated and/or terminated by a given set of macro-events (evaluation). Furthermore, the expressive power of the Macro-Event Calculus has been demonstrated through the encoding of a simple real-world example (a more complex example, namely the formalization of the Dagstuhl steam-boiler control specification problem [1], has been implemented using a superset of our proposal in [22]).

We are investigating ways to extend this work to naturally capture finer process constructors, in particular definitions, non-occurrence, and exclusive alternatives. This would allow, for example, a complete specification of the gas heater problem. We are also interested in formalizing synergetic effects and interference [22]. Finally, we are currently working at properly establishing the expressiveness and complexity of variants of the Macro-Event Calculus and at systematically comparing them with other formalisms for discrete process modeling proposed in the literature.

References

[1] J. Abrial. Steam-boiler control specification problem. In *Proc. of the Dagstuhl Seminar on Methods for Semantics and Specifications*, Dagstuhl, Germany, 1995.

[2] J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–580, 1994.

[3] S. Andler. Predicate path expressions. In *Proceedings of the 6th ACM Symposium on Principles of Programming Languages*, 1979.

[4] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proc. of 13th International Joint Conference on Artificial Intelligence — IJCAI*, pages 866–871, Chambéry, France, 1993. Morgan Kaufmann.

[5] C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1–3):85–117, 1997.

[6] P. Belegrinos and M. Georgeff. A model of events and processes. In *Proc. IJCAI'91*, Sydney, Australia, 1991.

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

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

[9] 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):111–164, Feb. 1999.

[10] L. Chittaro and A. Montanari. Reasoning about discrete processes in a logic programming framework. In D. Saccà, editor, *Proceedings of the Eight Conference on Logic Programming — GULP'93*, pages 407–421, Gizzieria Lido, Italy, 1993. Mediterranean Press.

[11] L. Chittaro and A. Montanari. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence*, 12(3):359–382, 1996.

[12] C. Evans. The macro-event calculus: Representing temporal granularity. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence — PRICAI'90*, Nagoya, Japan, 1990. IOS Press.

[13] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the limitations of the situation calculus? In R. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Kluwer, 1991.

[14] J. Gustafsson and L. Karlsson. Reasoning about actions in a multi-agent environment. *Linköping Electronic Articles in Computer and Information Science*, 1997. `http://www.ep.liu.se/ea/cis/1997/014`.

[15] C. Hoare. *Communicating Sequencial Processes*. Prentice-Hall, 1985.

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

[17] Y. Lespérance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. A logical approach to high-level robot programming — a progress report. In *Control of the Physical World by Intelligent Systems, WorkNotes of the 1994 AAAI Fall Symposium*, 1994.

[18] F. Lin and Y. Shoham. Concurrent actions in the situation calculus. In *Proc. of the 10th National Conference of the American Association for Artificial Intelligence — AAAI*, pages 590–695. AAAI Press/MIT Press, 1992.

[19] S. Lin and T. Dean. Localized temporal reasoning using subgoals and abstract events. *Computational Intelligence*, 12(3):423–449, 1996.

[20] R. Miller and M. Shanahan. Narratives in the situation calculus. *Journal of Logic and Computation*, 4(5):513–530, 1994.

[21] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1980.

[22] M. Mizzaro. La modellazione di sistemi complessi nel calcolo degli eventi: Analisi di un caso di studio (in italian). Tesi di Laurea in Scienze dell'Informazione, Università di Udine, 1997. Under the supervision of A. Montanari.

[23] J. Pinto. Concurrent actions and interacting effects. In *Proc. of the 6th International Conference on Principles of Knowledge Representation and Reasoning — KR'98*, pages 292–303. Morgan Kauffman, 1998.

[24] M. Shanahan. *Solving the Frame Problem*. The MIT Press, 1997.

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

[26] C. Yi. Reasoning about concurrent actions with features and fluents. In *Proceedings of the Third International Workshop on Temporal Representation and Reasoning — TIME'96*, pages 6–13, Key West, FL, 1996.