A Logic-Based, Reactive Calculus of Events

Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni

DEIS, University of Bologna,
Viale Risorgimento 2, 40136 - Bologna, Italy
name.surname@unibo.it

Abstract. Since its introduction, the Event Calculus ($\mathcal{EC}$) has been recognized for being an excellent framework to reason about time and events, and it has been applied to a variety of domains. However, its use inside logic-based frameworks has been mainly a-posteriori, based on specific queries and backward reasoning. This has somehow limited its applicability in dynamic environments. We fill this gap by proposing a Reactive and logic-based implementation of the $\mathcal{EC}$, called $\mathcal{REC}$. We give an axiomatization of $\mathcal{REC}$ inside the SCIFF Abductive Logic Programming framework, and study its formal properties.

1 Introduction

More than 20 years ago, Kowalski and Sergot [8] introduced the Event Calculus ($\mathcal{EC}$) as a general framework to reason about time and events which overcomes limitations of other previous approaches, such as the situation calculus.

The event calculus has many interesting features. Among them: an extremely simple and compact representation, symmetry of past and future, generality with respect to time orderings, executability and direct mapping with computational logic frameworks, modeling of concurrent events, immunity from the frame problem, and explicit treatment of time and events. It has therefore been applied in a variety of domains. Among them, planning, via Abductive Logic Programming (ALP), as suggested by Eshghi [5], using backward, goal-oriented and hypothetical reasoning.

A decade later, following a different line of research, Kowalski and Sadri [7] proposed to use ALP as a way to reconcile backward with forward reasoning inside an intelligent agent architecture. However, beside planning, ALP has not been used in combination with the $\mathcal{EC}$. Nor are we aware of other logical frameworks that implement the $\mathcal{EC}$ in a reactive way: i.e., a logical based implementation of $\mathcal{EC}$ that dynamically reacts to happening events is missing.

For that reason, we only find reactive $\mathcal{EC}$ implementations outside of logical frameworks, or else logic-based implementations of the $\mathcal{EC}$ that do not exhibit any reactive feature. As a consequence, large application domains such as runtime monitoring and event processing have been tackled so far by $\mathcal{EC}$-inspired methods but only based on ad-hoc methods without a strong formal basis. In particular, it is very difficult to understand and prove the formal properties of current reactive $\mathcal{EC}$ implementations.

In this work, we show how to overcome this limitation. We identify a new feature, not encompassed by the original $\mathcal{EC}$ framework, named irrevocability. We deem such a feature necessary for monitoring applications. Building on Kowalski et al.'s work, we equip the $\mathcal{EC}$ framework with the reactive features of a powerful, general purpose ALP language and proof-procedure named SCIFF. We obtain a reactive version of the calculus, which we call Reactive Event Calculus ($\mathcal{REC}$).

$\mathcal{REC}$ exhibits irrevocability. It draws inspiration from Chittaro and Montanari's work [3], in which they introduce the concept of *maximum validity intervals* (MVIs). These are the maximum time intervals in which fluents hold, according to the known events. Chittaro and Montanari propose a mechanism, called *Cached Event Calculus* ($\mathcal{CEC}$), to cache the outcome of the inference process every time the knowledge base is updated by a new event. Also $\mathcal{REC}$ uses MVIs.

We demonstrate the advantage of $\mathcal{REC}$ with respect to existing $\mathcal{EC}$ implementations and investigate in depth the $\mathcal{REC}$'s formal properties. We identify a class of *well-formed* specifications, which ensure that $\mathcal{REC}$ is a viable method for runtime, event-based tracking of a system's properties.

2 The Event Calculus

The Event Calculus ($\mathcal{EC}$) was introduced as a logic programming framework for representing and reasoning about events and their effects [8]. Basic concepts are that of *event*, happening at a point in time, and *fluent*, holding during time intervals. Fluents are initiated/terminated by events. Given an event narrative (a set of events), the $\mathcal{EC}$ theory and domain-specific axioms together ("$\mathcal{EC}$ axioms") define what fluents hold at each time. There are many different formulations of these axioms [4]. One possibility is given by the following axioms ec_1, ec_2 (P stands for *Fluent*, E for *Event*, and T represents time instants):

$$\begin{aligned} holds\_at(P,T) \leftarrow\ & initiates(E,P,T_{Start}) \\ & \wedge\, T_{Start} < T \wedge \neg clipped(T_{Start},P,T). \end{aligned} \tag{ec_1}$$

$$\begin{aligned} clipped(T_1,P,T_3) \leftarrow\ & terminates(E,P,T_2) \\ & \wedge\, T_1 < T_2 \wedge T_2 < T_3. \end{aligned} \tag{ec_2}$$

$$\begin{aligned} initiates(E,P,T) \leftarrow\ & happens\_at(E,T) \wedge holds\_at(P_1,T) \\ & \wedge\, ... \wedge holds\_at(P_M,T). \end{aligned} \tag{ec_3}$$

$$\begin{aligned} terminates(E,P,T) \leftarrow\ & happens\_at(E,T) \wedge holds\_at(P_1,T) \\ & \wedge\, ... \wedge holds\_at(P_N,T). \end{aligned} \tag{ec_4}$$

Axioms (ec_3, ec_4) are schemas for defining the domain-specific axioms: a certain fluent P is initiated/terminated at a time instant T if an event E happened at the same time, and if some other fluents P_i hold at that time. Sometimes $initially(P)$ is used to define fluents that hold at the beginning of time. Dual axioms and predicates can be added to define when fluents *do*/*do not* hold [10]: e.g., an axiom can be added to define *declipped*/3 (an event has made a certain fluent holding).

The $\mathcal{EC}$ framework has been extensively used in the past to carry out two main reasoning tasks: deductive *narrative verification*, to check whether a certain fluent holds given a narrative (set of events), [8], and abductive *planning*, to simulate a possible narrative which satisfies some requirements [11]. These tasks take place after or prior to execution, but not during execution. The reason is that each time an event occurs, the $\mathcal{EC}$ enables a straightforward update of the theory (it suffices to add *happens_at* facts), but it incurs a substantial increase of the query time, since backward reasoning has to be restarted from scratch. However, runtime reasoning tasks, such as monitoring, would greatly benefit from the $\mathcal{EC}$'s expressive power. For this reason, some propose to cache the outcome of the inference process every time the knowledge base is updated by a new event. The *Cached Event Calculus* ($\mathcal{CEC}$) [3] computes and stores fluents' *maximum validity intervals* (MVIs), which are the maximum time intervals in which fluents hold, according to the known events. The set of cached validity intervals is then extended/revised as new events occur or get to be known.

3 The Reactive Event Calculus

The $\mathcal{EC}$ can be elegantly formalized in logic programming, but as we said above, that would be suitable for top-down, "backward" computation. Runtime monitoring, intended as checking the behaviour of interacting entities, capturing at run time (as soon as possible) violations, and possibly raising alarms, requires reactive $\mathcal{EC}$ implementations. For this reason, we resort to a framework which reconciles backward with forward reasoning: the SCIFF language and proof-procedure [1].

SCIFF is an extension of Fung an Kowalski's IFF proof-procedure for abductive logic programming [6]. It has two primitive notions: events (mapped as **H** atoms) and expectations (mapped as **E**/**EN** atoms). $\mathbf{H}(Ev, T)$ means that an event Ev has occurred at time T, and it is a ground atom. $\mathbf{H}(Ev, T)$ is similar to Chittaro and Montanari's *happens_at* atom. Instead $\mathbf{E}(Ev, T)$ and $\mathbf{EN}(Ev, T)$ can contain variables with domains and CLP constraints, and they denote in the first case that an event unifying with Ev is expected to occur at some time in the range of T (T existentially quantified), and in the second case that all events unifying with Ev are expected to not occur, at all times in the range of T (i.e., T is considered as universally quantified over its CLP range). SCIFF accommodates existential and universal variable quantification and quantifier restriction, CLP constraints, dynamic update of event narrative and it has a built-in runtime protocol verification procedure. A SCIFF specification is composed of a knowledge base $\mathcal{P}$, a set of ICs (integrity constraints) $\mathcal{IC}$, and a goal $\mathcal{G}$. $\mathcal{P}$ consists of backward rules $head \leftarrow body$ (see ax_1 below), whereas the ICs in $\mathcal{IC}$ are forward implications $body \rightarrow head$ (see ax_2). ICs are interpreted in a reactive manner; the intuition is that when the body of an IC becomes true (i.e., the involved events occur), then the rule fires, and the expectations in the head are generated by abduction. For example, $\mathbf{H}(a, T) \rightarrow \mathbf{EN}(b, T')$ defines a relation between events a and b, saying that if a occurs at time T, b should not

occur at any time; $\mathbf{H}(a,T) \rightarrow \mathbf{E}(b,T') \wedge T' \leq T + 300$ says that if a occurs, then an event b should occur no later than 300 time units after a.

To exhibit a correct behavior, given a goal $\mathcal{G}$ and a triplet $\langle \mathcal{P}, \mathcal{A}, \mathcal{IC} \rangle$, a set of abduced expectations must be *fulfilled* by corresponding events. The SCIFF semantics [1] is given for a given specifications and narrative, denoted by **HAP** (a set of **H** atoms), and it intuitively states that $\mathcal{P}$, together with the abduced literals, must entail $\mathcal{G} \wedge \mathcal{IC}$, **E** expectations must have a corresponding matching happened event, and **EN** expectations must not have a corresponding matching event. Moreover, **E** and **EN** atoms must be consistent and not contradictory (this is formalized in [1] via the **E**-consistency notion).

The SCIFF axiomatization of $\mathcal{EC}$ that follows draws inspiration from Chittaro and Montanari's $\mathcal{CEC}$ and on their idea of MVIs. Events and fluents are terms and times are integer (CLP) variables, 0 being the "initial" time. The main distinctive feature of our implementation is its reactivity: fluents are initiated and terminated by dynamically occurring events. Thus its name, "Reactive Event Calculus" ($\mathcal{REC}$). $\mathcal{REC}$ uses the abduction mechanism to generate MVIs and define their persistence. It has a fully declarative axiomatization (Axioms ax_1 through ax_7): no operational specifications are needed. It uses two special internal events (denoted by the reserved *clip/declip* words, differently from generic external events, taht are "incapsulated" into the reserved term *event*) to model that a fluent is terminated/initiated, respectively. The expressive power of $\mathcal{REC}$ is the same as the one of $\mathcal{CEC}$, specifically it enables the definition of a context. A use case will be shown below.

Axiom 1 (Holding of fluent) *A fluent F holds at time T if a MVI containing T has been abduced for F.*[1]

$$holds\_at(F,T) \leftarrow \quad \boldsymbol{mvi}(F,[T_s,T_e]) \wedge T > T_s \wedge T \leq T_e. \qquad (ax_1)$$

Axiom ax_1 is a backward rule (clause), as well as Axiom ax_7, whereas Axiom ax_2 through Axiom ax_6 are forward implications (ICs). Such a mixture of backward and forward inference rules is enabled by ALP [7] and it represents the backbone of $\mathcal{REC}$'s reactive behaviour.

Axiom 2 (MVI semantics) *If $(T_s,T_e]$ is a MVI for F, then F must be declipped at time T_s and clipped at time T_e, and no further declipping/clipping must occur in between.*

$$\begin{aligned} &\boldsymbol{mvi}(F,[T_s,T_e]) \\ &\quad \rightarrow \mathbf{E}(declip(F),T_s) \wedge \mathbf{E}(clip(F),T_e) \\ &\qquad \wedge \mathbf{EN}(declip(F),T_d) \wedge T_d > T_s \wedge T_d \leq T_e \\ &\qquad \wedge \mathbf{EN}(clip(F),T_c) \wedge T_c \geq T_s \wedge T_c < T_e. \end{aligned} \qquad (ax_2)$$

[1] A fluent F does not hold at the time it is declipped, but holds at the time it is clipped, i.e., MVIs are left-open and right-closed.

Axiom 3 (Initial status of fluents) *If a fluent initially holds, a corresponding declipping event is generated at time 0.*

$$initially(F) \rightarrow \mathbf{H}(declip(F), 0). \qquad (ax_3)$$

Operationally, Axiom ax_3 enforces the generation of a set of **H** events that are needed for the correct extension of MVIs.

Axiom 4 (Fluents initiation) *If an event Ev occurs at time T which initiates fluent F, either F already holds or it is declipped.*

$$\begin{aligned} &\mathbf{H}(event(Ev), T) \wedge initiates(Ev, F, T) \\ \rightarrow &\mathbf{H}(declip(F), T) \\ &\vee \mathbf{E}(declip(F), T_d) \wedge T_d < T \\ &\quad \wedge \mathbf{EN}(clip(F), T_c) \wedge T_c > T_d \wedge T_c < T. \end{aligned} \qquad (ax_4)$$

(ax_4) does not use the *holds_at* predicate and it does not incur a new MVI.

Axiom 5 (Impact of initiation) *The happening of a* declip(F) *event causes fluent F to start to hold.*

$$\mathbf{H}(declip(F), T_s) \rightarrow \ \boldsymbol{mvi}(F, [T_s, T_e]) \wedge T_e > T_s. \qquad (ax_5)$$

Axiom 6 (Fluents termination) *If an event Ev occurs which terminates a fluent F, F is clipped.*

$$\begin{aligned} &\mathbf{H}(event(Ev), T) \\ \wedge &terminates(Ev, F, T) \rightarrow \mathbf{H}(clip(F), T). \end{aligned} \qquad (ax_6)$$

Axiom 7 (Final clipping of fluents) *All fluents are terminated by the special* complete *event.*

$$terminates(complete, \_, F). \qquad (ax_7)$$

4 $\mathcal{REC}$ illustrated: a personnel monitoring facility

The following real-world case study has been proposed to us by a local medium-sized enterprise. A company wants to monitor its personnel's time-sheets. Each employee punches the clock when entering or leaving the office. The system recognizes two events:

- *check_in*(E): employee E has checked in;
- *check_out*(E): employee E has checked out.

The following requirements on employee behaviour require monitoring:

(R0) after check in, an employee must check out within 8 hours;

(R1) as soon as a deadline expiration is detected, a dedicated alarm fires at an operator's desk. It reports the employee ID, and an indication of the time interval elapsed between deadline expiration and its detection. The alarm is turned off when the operator decides to handle it.

We assume that the following actions are available to the operator:

- $handle(E)$ states that the operator wants to handle the situation concerning the employee identified by E;
- *tic* is used to take a snapshot of the current situation of the system, by updating the current time.

We capture requirements (R0) and (R1), using three fluents:

- $in(E)$: E is currently in;
- $should\_leave(E, T_d)$: E is expected to leave her office by T_d;
- $alarm(delay(E, D))$: E has not left the office in time – D represents the difference between the time a deadline expiration is detected and the deadline expiration time itself.

It is possible to model such requirements declaratively using *initiates* and *terminates* predicate definitions. We assume hour time granularity.

Let us first focus on the $in(E)$ fluent. E is "in" as of the time she checks in. She ceases to be "in" as of the time she checks out:

$$initiates(check\_in(E), in(E), \_). \tag{1}$$

$$terminates(check\_out(E), in(E), T) \leftarrow holds\_at(in(E), T). \tag{2}$$

When E checks in at T_c, a *should_leave* fluent is activated, expressing that E is expected to leave the office by $T_c + 8$:

$$initiates(check\_in(E), should\_leave(E, T_d), T_c) \leftarrow T_d \text{ is } T_c + 8. \tag{3}$$

(note that T_c is ground at *body* evaluation time, due to ax_4).

Such a fluent can be terminated in two ways: either E correctly checks out within the 8-hour deadline, or the deadline expires. In the latter case, termination is imposed at the next *tic* action.

$$terminates(check\_out(E), should\_leave(E, T_d), T_c) \leftarrow holds\_at(should\_leave(E, T_d), T_c) \wedge T_c \leq T_d. \tag{4}$$

$$terminates(tic, should\_leave(E, T_d), T) \leftarrow holds\_at(should\_leave(E, T_d), T) \wedge T > T_d. \tag{5}$$

The same *tic* action also causes an alarm to go off:

$$initiates(tic, alarm(delay(E, D), T) \leftarrow holds\_at(should\_leave(E, T_d), T) \wedge D \text{ is } T - T_d. \tag{6}$$

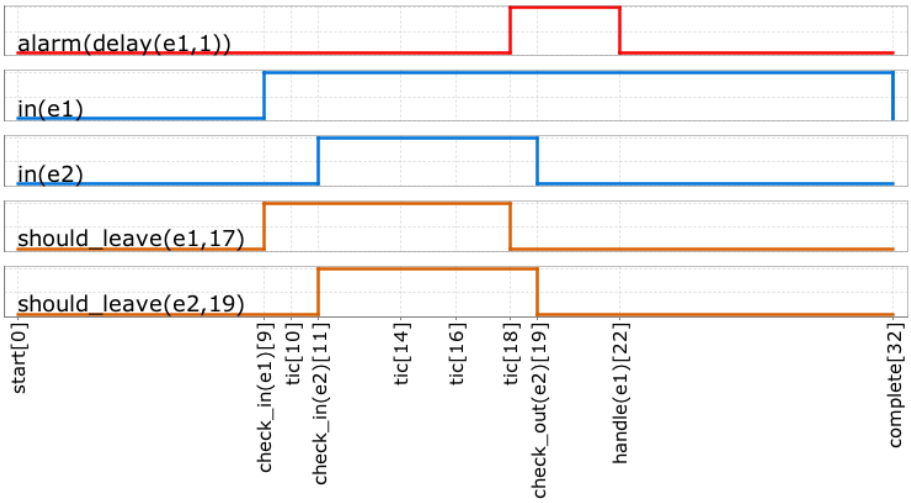

Fig. 1. Fluents tracking with $\mathcal{REC}$.

Note that in these equations the time of event termination/start (T_c and T) is the same time present in their respective body's *holds_at* atoms. This is perfectly normal, but it is not a requirement. In particular, times could be different, as long as the times of *holds_at* atoms do not follow event termination/start times. That again would be allowed, but it would amount to define fluents that depend on future events: fluents that are thus not suitable for runtime monitoring. For that reason, we assume that *well-formed* theories do not contain such kind of clauses. More details on this matter will be given below when we discuss the *irrevocability* property in a formal way.

Finally, an alarm is turned off when the operator handles it:

$$terminates(handle(E), alarm(delay(E, D)), T) \leftarrow holds\_at(delay(E, D), T). \quad (7)$$

Based on such a theory, $\mathcal{REC}$ becomes able to dynamically reason from the employees' flow inside the company. In particular, $\mathcal{REC}$ tracks the status of each employee, and generates an alarm as soon as a *tic* action detects a deadline expiration.

Let us consider an event narrative (*execution trace*) involving two employees e_1 and e_2, where e_2 does respect the required deadline while e_1 does not:

$$\begin{array}{lll} \mathbf{H}(event(check\_in(e_1)), 9), & \mathbf{H}(event(tic), 10), & \mathbf{H}(event(check\_in(e_2)), 11), \\ \mathbf{H}(event(tic), 14), & \mathbf{H}(event(tic), 16), & \mathbf{H}(event(tic), 18). \end{array}$$

Figure 1 shows the global state of fluents at 18, when $\mathcal{REC}$ generates an alarm because e_1 was expected to leave the office no later than 17, but she has not left yet. The operator can check all pending alarms, and pick an employee to handle

in case. The execution now proceeds as follows:

$$\mathbf{H}(event(check\_out(e_2)), 19), \qquad \mathbf{H}(event(handle(e_1)), 22),$$
$$\mathbf{H}(event(check\_out(e_1)), 23).$$

e_2 correctly leaves the office within the deadline, bringing her corresponding *in* and *should_leave* fluents to termination. At 22 the operator handles an alarm involving e_1, who eventually leaves her office at 23.

In general, a monitoring application is usable only if it provides stable, deterministic outputs, which do not flutter due to algorithmic issues but only change as a sensible response to the inputs. The reasons of an undesired fluttering behaviour could be of two types: a bad set of specifications, or an unsuitable underlying reasoning machinery.

As an example of the former, imagine to replace Eq. (1) by

$$initiates(check\_in(e_1), in(E_2), \_), \tag{8}$$

in which E_2 represents a generic employee. This is an ambiguous specification since it does not clearly state which employee should change status as a consequence of e_1 checking in. Another different source of ambiguity could lay hidden in an alternative formulation of Eq. (6) such as the following:

$$\begin{aligned} initiates(tic, alarm(delay(E, D), T) \leftarrow & \\ holds\_at(should\_leave(E, T_d), T_1) \wedge D \text{ is } T - T_d, T_1 > T. & \end{aligned} \tag{9}$$

The meaning would be that an action is a consequence of an alarm which has not fired yet. Only speculations are possible in that case, and no runtime monitoring algorithm could provide deterministic answers (save freezing until the alarm fires, but in that case the application would no longer be called "runtime").

Thus we need to isolate "good" sets of specifications. Once we have them, we must ensure that the reasoning machinery does not make unjustified retractions. In other words, we must guarantee irrevocability.

5 Formal properties of $\mathcal{REC}$

$\mathcal{REC}$ is implemented on top of SCIFF. As a consequence, it inherits soundness and completeness properties of the SCIFF's operational semantics with respect to the declarative semantics.

Theorem 1 (Soundndess and completeness of $\mathcal{REC}$). *$\mathcal{REC}$ is sound and complete. Specifically, the SCIFF proof-procedure will derive all and only the answers defined by its corresponding declarative semantics, augmented with the $\mathcal{REC}$ axioms ax_1–ax_7.*

Proof. It follows from SCIFF's soundness and completeness properties [1]. □

Hence, the operational behaviour of the $\mathcal{REC}$ is faithful to its axiomatization. We are unaware of other implementations of the $\mathcal{EC}$ that provide such a guarantee. The next results concern uniqueness and irrevocability, and they are instead relative to the special kind of reasoning needed for the monitoring applications. We thus need to introduce some information about the SCIFF's operational behaviour.

5.1 Open, closed and semi-open reasoning

In general, SCIFF features two main forms of inference, called *open* and *closed derivation*. A derivation starts from a (possibly empty) goal and $\mathcal{IC}$, and it generates a list of nodes according to the operational semantics [1], i.e., by applying a SCIFF transition rule at a time. It terminates when no transition is applicable. Open and closed derivations differ by one such transition.

Given a specification $\mathcal{S}$ and two *execution traces* (sets of **H** events) $\mathcal{H}^i$ and $\mathcal{H}^f \supseteq \mathcal{H}^i$, if there exists an *open successful derivation* [2] for a goal $\mathcal{G}$ that leads from $\mathcal{H}^i$ to $\mathcal{H}^f$ we write $\mathcal{S}_{\mathcal{H}^i}\!\!\mid\!\!\sim_{\Delta}^{\mathcal{H}^f}\mathcal{G}$, where Δ is the computed abductive explanation.[2] If $\mathcal{S}$ is a $\mathcal{REC}$ specification, Δ includes the abduced MVIs. When SCIFF executes an open derivation, it assumes that the acquired execution trace is partial. Thus **E** atoms without a matching **H** atom are not considered as violated but only as *pending*: further events may still occur to fulfill them. **EN** atoms can instead be evaluated, because they must never have a matching **H** atom. This approach is used when SCIFF is used for runtime verification, with events occuring dynamically, and the narrative is incomplete.

SCIFF can also perform *closed derivations*, to reason from narratives known to be complete, or to close the inference process when a dynamic execution comes to an end. In that case, both **E** and **EN** atoms are evaluated: a CWA is made about the collected execution trace, and pending expectations are considered as violated, because no further event will occur to fulfill them.

SCIFF is sound and complete independently of the order of events. However, there are many important domains in which we can safely assume that events are acquired in increasing order of time. In that case, reasoning is *partially open*: open on the future, when events may still occur, but closed on the past. Expectations on the past can thus be evaluated immediately. To enable this form of *semi-open* reasoning, the SCIFF proof-procedure is equipped with an additional rule, which states that if the execution trace has reached time t, then all pending expectations must be fulfilled at a time $t' \geq t$. We denote such a *semi-open* derivation by $\mid\!\!\sim$.

5.2 Irrevocability of $\mathcal{REC}$

Monitoring applications enable semi-open derivation. It is required that the generated MVIs are never retracted, but only extended or terminated as new events

[2] Note that Δ only depends on $\mathcal{H}^f$, because $\mathcal{H}^f$ includes $\mathcal{H}^i$. Therefore, in the following we will omit $\mathcal{H}^i$ when possible.

occur. If that is the case, the reasoning process is called *irrevocable*. Some target applications need irrevocable tracking procedures, which can give a continuously updated view of the status of all fluents. Fluttering behaviours must be by all means avoided. This is true, e.g., when the modeled fluents carry a normative meaning. It would be undesirable, for instance, to attach a certain obligation to an agent at runtime, and see it disappear later only because the calculus revises its computed status.

In the reminder of this section, we first define a class of $\mathcal{REC}$ theories, then we show that semi-open reasoning on the resulting $\mathcal{REC}$ specifications is irrevocable. Note that when monitoring the execution, $\mathcal{REC}$ is used with goal *true*.

Definition 1 (Well-formed $\mathcal{REC}$ theory). *A well-formed $\mathcal{REC}$ theory $\mathcal{T}$ is a set of clauses of the type* [3]

$$initiates(Ev, F, T) \leftarrow body.$$
$$terminates(Ev, F, T) \leftarrow body.$$

which satisfies the following properties:

1. *negation is not applied to holds_at predicates;*
2. *for initiates/3 clauses, fluent F must always be resolved with a ground substitution.*
3. *$\forall holds\_at(F_2, T_2)$ predicate used in body, $T_2 \leq T$.*

In the previous section our illustration was modeled by a well-formed $\mathcal{REC}$ theory. Using the same example, we have discussed the consequences of ill-formed specifications in the $\mathcal{REC}$ theory in a concrete case. Definition 1 identifies in the general case the three possible sources of non irrevocability. In particular, 1. ensures monotonicity, 2. prevents non-determinism due to case analysis and 3. restricts us to reasoning on stable, past conditions. $\mathcal{REC}$ theories that violate 2. and 3. would introduce choice points that hinder irrevocability.

Definition 2 ($\mathcal{REC}$ specification). *Given a well-formed $\mathcal{REC}$ theory $\mathcal{T}$, the corresponding $\mathcal{REC}$ specification $\mathcal{R}^{\mathcal{T}}$ is defined as the SCIFF specification:*[4]

$$\mathcal{R}^{\mathcal{T}} \equiv \langle \mathrm{KB}_{\mathcal{REC}} \cup \mathcal{T}, \{\mathbf{E}, \mathbf{EN}, \ \boldsymbol{mvi}\}, \mathrm{IC}_{\mathcal{REC}} \rangle \tag{10}$$

where $\mathrm{KB}_{\mathcal{REC}} = \{(ax_1), (ax_7)\}$ *and* $\mathrm{IC}_{\mathcal{REC}} = \{(ax_2), (ax_3), \ldots, (ax_6)\}$.

The following three lemmas establish interesting properties of $\mathcal{REC}$, defining the link between MVIs and the internal events used to clip and declip them.

[3] The *body* is a conjunction of *holds_at* predicates and CLP constraints. It can be omitted when *true*.

[4] Throughout the paper, we will simply use $\mathcal{R}$ to identify a generic $\mathcal{REC}$ specification. We will also state that a $\mathcal{REC}$ specification is well-formed as a shortcut to state that its theory is.

Lemma 1 (Groundedness of MVIs' starting times). *For each well-formed $\mathcal{REC}$ theory $\mathcal{T}$, for each execution trace $\mathcal{H}$ and given the goal true, the abduced MVIs always have a ground starting time, i.e.*

$$\forall\ \Delta,\ \mathcal{T} \overset{\mathcal{H}}{\underset{\Delta}{\models}} true \Rightarrow \forall\ \ \boldsymbol{mvi}(F, [T_s, T_e]) \in \Delta, T_s \in \mathbb{N}$$

Proof. Axiom (ax_5) is the only IC in which the abducible representing a MVI appears in the head. The starting time T_s is bound to the time at which the *declip* event in the body happens. This event is not contained in the execution trace, but is instead generated by applying axiom (ax_3) or (ax_4). By axiom (ax_3), a *declip* event is generated at time 0, whereas by axiom (ax_4), a *declip* event is generated using the same time of the body's external **H** event. Since all happened events contained in an execution trace are ground, then also the *declip* event is abduced to happen at a ground time. Therefore, also the starting time T_s of the corresponding MVI is ground. □

Lemma 2 (Relationship between clipping events and MVIs). *The expectation about the clipping of a MVI can be fullfilled by exactly one happened event, in particular the nearest which occurs after the declipping of the MVI.*

Proof. Let us consider by abdsurdum that there exists $\boldsymbol{mvi}(f, [t_s, T_e])$ triggering Axiom (ax_2) and generating an expectation about $clip(f)$ which can potentially be fulfilled by two distinct event happening, say, at time $t_{e1} > t_s$ and $t_{e2} > t_{e1}$, are able to fulfill it. If $\mathbf{E}(clip(f), T_e)$ is fulfilled at time t_{e2}, then Axiom (ax_2) also imposes $\mathbf{EN}(clip(f), T_c)$ between t_s and t_{e2}. However, this time interval also includes t_{e1}, leading to inconsistency. □

Lemma 3 (Interleaving between declipping and clipping events). *For each fluent, between two declipping events at least one clipping event must occur.*

Proof. Let us suppose that there exists a fluent f for which two $declip(f)$ events occur, say, at time t_1 and $t_2 > t_1$, without having a $clip(f)$ inbetween. An MVI is generated for f starting at t_1 (thank to Axiom ax_5); this MVI will be clipped by the first consequent time at which a $clip(f)$ occurs (see Lemma 2). The hypothesis which states that this $clip(f)$ event must occur after t_2 is however inconsistent, because Axiom ax_2 would state that between t_1 and this time (thus t_2 included) no further $declip(f)$ event can occur. □

We are now ready to state the following:

Theorem 2 (Uniqueness of derivation). *For each well-formed $\mathcal{REC}$ theory $\mathcal{T}$ and for each execution trace $\mathcal{H}$, there exists exactly one successful semi-open derivation computed by SCIFF for the goal true, i.e. $\exists 1 \Delta$ s.t. $\mathcal{T} \overset{\mathcal{H}}{\underset{\Delta}{\models}} true$.*

Proof. First, we prove that at most one computed explanation exists. Different explanations are computed when inclusive disjunctions are contained in the head of some integrity constraint, or if there are different ways to fulfill a positive expectation.[5]

[5] The other possibilities of having multiple explanations are ruled out by the fact that *initiates* predicate are resolved with a ground fluent, thus MVIs are always ground.

Let us first consider the case of disjunctions in the head. The only integrity constraint containing a disjunctive head is (ax_4);[6] however, we prove that these two disjuncts are mutually exclusive. Let us consider the second disjunct. It states that fluent F has been already declipped at a certain past time (let us denote it with t_d), and that between t_d and T no clipping event has been generated: thus F still holds at time T. In fact, when $declip(F)$ happened at time t_d, a ***mvi*** abducible was generated by applying rule (ax_5); due to the negative expectation contained in the second disjunct of (ax_5)'s head, this maximal validity interval must be clipped at a time greater than T (say, t_c). The application of rule (ax_2), in turn, states that it is forbidden to declip F between t_d and t_c, i.e., also at time T. Therefore, the first disjunct of (ax_5)'s head cannot be true at time T. A further important observation concerns the semi-open nature of the derivation. Event if the first disjunct of Axiom (ax_5) did not lead to violation, an open derivation would open a choice point, waiting for a suitable past declipping event to fulfill the expectation in the second disjunct. As we have just proven, this second possibility is impossible, because the two disjuncts are mutually exclusive. A semi-open derivation is immediately able to detect this situation, because the second disjunct refers to the past, and the first one to the present. Therefore, no choice point is left open.

Let us now consider the reasons to fulfill positive expectations, which are present in axioms (ax_2) and (ax_4). $\mathbf{E}(declip(F), T_s)$ in Axiom (ax_2) can be fulfilled in one way, because both F and T_s are ground, the former because $\mathcal{T}$ is well-defined by hypothesis, the latter as stated in Lemma 1. By Lemma 2, also the expectation about the clipping of the fluent can be fulfilled by exactly one event, in particular by the first clipping occurring after *declip*. Finally, the positive expectation in Axiom (ax_4) can be fulfilled by only one *declip* event; the proof is obtained by combining the negative expectation about the clipping of the fluent in Axiom ax_4 and the result proven in Lemma 3.

Now we prove that there always exists a computed explanation, i.e. that, when the goal is *true*, all execution traces comply with the $\mathcal{REC}$ specifications. The axioms imposing expectations are (ax_2) and (ax_4). If the positive expectation in the second disjunct of (ax_4)'s head cannot be fulfilled, then the involved fluent is not holding, and therefore it can be declipped by choosing the first disjunct. When the goal is *true*, Axiom (ax_2) only fires if Axiom (ax_5) fires, and therefore the positive expectation about the declipping of the fluent has a corresponding matching event (exactly the one which has caused Axiom (ax_5) to fire). The positive expectation about the clipping of fluent is eventually fulfilled by the special *complete* event, which is able to terminate all fluents (see Axiom (ax_7)). Finally, the negative expectations contained in Axioms (ax_2) and (ax_4) are simply used to select the "nearest" declipping/clipping, but are not used to rule out executions. □

Theorem 2 ensures that exactly one Δ is produced by a semi-open derivation of SCIFF. This, in turn, means that there exists exactly one "configuration" for

[6] The presence of negated abducibles in the body of a rule would also produce inclusive disjunctive head [6], but Definition 1 forbids negated *holds_at* predicates.

the MVIs of each fluent. We give a precise definition of this notion of state, which is the one of interest when evaluating the irrevocability of the reasoning process, and define the notion of progressive extension between states, which formally define irrevocability.

Definition 3 (Current time). *The* current time *of an execution trace $\mathcal{H}$, $ct(\mathcal{H})$, is the latest time of its events:*

$$ct(\mathcal{H}) \equiv \max(t \mid \mathbf{H}(event(\_), t) \in \mathcal{H})$$

Definition 4 (MVI State). *Given a $\mathcal{REC}$ specification $\mathcal{R}$ and an execution trace $\mathcal{H}$ the resulting* MVI state *at time $ct(\mathcal{H})$ is the set of* ***mvi*** *abducibles contained in the computed explanation generated by SCIFF with goal true:*

$$\mathrm{MVI}(\mathcal{R}_{\mathcal{H}}) \equiv \{\ \boldsymbol{mvi}(F, [T_s, T_e]) \in \Delta\},\ \textit{where}\ \mathcal{R} \mid\!\!\sim^{\mathcal{H}}_{\Delta} true$$

Definition 5 (State sub-sets). *Given a $\mathcal{REC}$ specification $\mathcal{R}$ and a (partial) execution trace $\mathcal{H}$, the current state* $\mathrm{MVI}(\mathcal{R}_{\mathcal{H}})$ *is split into two sub-sets:*

- $\mathrm{MVI}_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}})$*, is the set of (closed) MVIs, terminating at a ground time:*

$$\mathrm{MVI}_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}}) = \{\ \boldsymbol{mvi}(F, [s, e]) \in \mathrm{MVI}(\mathcal{R}_{\mathcal{H}}) \mid s, e \in \mathbb{N}\};$$

- $\mathrm{MVI}_{\sqcap}(\mathcal{R}_{\mathcal{H}})$*, is the set of (open) MVIs, terminating at a variable time:*

$$\mathrm{MVI}_{\sqcap}(\mathcal{R}_{\mathcal{H}}) = \{\ \boldsymbol{mvi}(F, [s, T]) \in \mathrm{MVI}(\mathcal{R}_{\mathcal{H}}) \mid s \in \mathbb{N}\}.$$

Definition 6 (Trace extension). *Given two execution traces $\mathcal{H}^1$ and $\mathcal{H}^2$, $\mathcal{H}^2$ is an* extension *of $\mathcal{H}^1$, written $\mathcal{H}^1 \prec \mathcal{H}^2$, iff*

$$\forall\ \mathbf{H}(e, t) \in \mathcal{H}^2/\mathcal{H}^1,\ t > ct(\mathcal{H}^1)$$

Definition 7 (State progressive extension). *Given a well-formed $\mathcal{REC}$ specification $\mathcal{R}$ and two execution traces $\mathcal{H}^1$ and $\mathcal{H}^2$, the state of $\mathcal{R}_{\mathcal{H}^2}$ is a* progressive extension *of the state of $\mathcal{R}_{\mathcal{H}^1}$, written* $\mathrm{MVI}(\mathcal{R}_{\mathcal{H}^1}) \trianglelefteq \mathrm{MVI}(\mathcal{R}_{\mathcal{H}^2})$*, iff*

1. *the set of closed MVIs is maintained in the new state:* $\mathrm{MVI}_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}^1}) \subseteq \mathrm{MVI}_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}^2})$
2. *if the set of MVIs is extended with new MVIs, these are declipped after the maximum time of $\mathcal{H}^1$:* $\forall\ \ \boldsymbol{mvi}(f, [s, t]) \in \mathrm{MVI}(\mathcal{R}_{\mathcal{H}^2})/\mathrm{MVI}(\mathcal{R}_{\mathcal{H}^1}), s > ct(\mathcal{H}^1)$
3. $\forall\ \ \boldsymbol{mvi}(f, [s, T_e]) \in \mathrm{MVI}_{\sqcap}(\mathcal{R}_{\mathcal{H}^1})$*, either*
   (a) *it remains untouched in the new state:*
   $\boldsymbol{mvi}(f, [s, T_e]) \in \mathrm{MVI}_{\sqcap}(\mathcal{R}_{\mathcal{H}^2})$*, or*
   (b) *it is clipped after the maximum time of $\mathcal{H}^1$:*
   $\boldsymbol{mvi}(f, [s, e]) \in \mathrm{MVI}_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}^2}), e > ct(\mathcal{H}^1)$.

Progressive extensions capture the intuitive notion that a state extends another one if it keeps the already computed closed MVIs and affects the status of fluents only at later times w.r.t. the time the first state was recorded. The extension is determined by adding new MVIs and by clipping fluents which held at the previous state. We can state the main result leading to irrevocability, namely that extending a trace results in a progressive extension of the MVI state.

Lemma 4 (Trace extension leads to a state progressive extension). *Given a well-formed $\mathcal{REC}$ specification $\mathcal{R}$ and two execution traces $\mathcal{H}^1$ and $\mathcal{H}^2$,*

$$\mathcal{H}^1 \prec \mathcal{H}^2 \Rightarrow \text{MVI}(\mathcal{R}_{\mathcal{H}^1}) \trianglelefteq \text{MVI}(\mathcal{R}_{\mathcal{H}^2})$$

Proof. Let us consider Definition 7, showing that $MVI(\mathcal{R}_{\mathcal{H}^1})$ and $MVI(\mathcal{R}_{\mathcal{H}^2})$ obey to its three requirements:

1. Let us consider an element of $MVI_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}^1})$, say, $\boldsymbol{mvi}(f,[s,e])$. The presence of this element is evidence that an event $\in \mathcal{H}^1$ occurring at time e exists s.t. fluent f is declipped. Since this event belongs to $\mathcal{H}^1$, $e < ct(\mathcal{H}^1)$. From the hypotheses that $\mathcal{H}^1 \prec \mathcal{H}^2$, all the events that belong to $\mathcal{H}^2/\mathcal{H}^1$ happen at a time greater than $ct(\mathcal{H}^1)$. Lemma 2 thus ensures that none of these events can change $\boldsymbol{mvi}(f,[s,e])$, which is maintained untouched in $MVI_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}^2})$.
2. As pointed out in the proof of Lemma 1, a happened event can start a new MVI at the time it happens. Since $\mathcal{H}^1 \prec \mathcal{H}^2$, each MVI generated by events belonging to $\mathcal{H}^2/\mathcal{H}^1$ will always have a starting time greater than $ct(\mathcal{H}^1)$. The only unfortunate case would be that there exists an event associated to an *initiates* predicate that can be evaluated as true $after$ the time at which the event occurs. This case arises when the *initiates* predicate is defined in terms of *holds_at* predicates which refer to the future.[7] However, Definition 1 rules out theories of this kind.
3. Axiom ax_6 is the rule which regulates the way fluents are clipped; a happened event can cause the termination of a MVI exactly at the time at which it occurs. From the hypotheses that $\mathcal{H}^1 \prec \mathcal{H}^2$, if a MVI is open at time $ct(\mathcal{H}^1)$ (i.e., it belongs to $MVI_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}^1})$), it is impossible for an event belonging to $\mathcal{H}^2/\mathcal{H}^1$ to clip it before time $ct(\mathcal{H}^1)$. Indeed, as in the case of *initiates* predicates, theories which lead to violate this property are not well-formed. Two possible cases for such an MVI may then arise:
   (a) no event $\in \mathcal{H}^2/\mathcal{H}^1$ is able to terminate the fluent associated to the MVI, which is therefore maintained untouched in $MVI_{\sqcap\!\sqcup}(\mathcal{R}_{\mathcal{H}^1})$;
   (b) there exists at least one event $\in \mathcal{H}^2/\mathcal{H}^1$ able to terminate the fluent associated to the MVI; the first one (see Lemma 2) will shift the MVI from the open to the closed set and respect the requirement that the MVI termination time must be greater than $ct(\mathcal{H}^1)$. □

Theorem 3 (Irrevocability of $\mathcal{REC}$). *Given a well-formed $\mathcal{REC}$ specification with goal true and a temporally ordered narrative, each time SCIFF processes a new event, the new MVI state is a progressive extension of the previous one.*

Proof. Let us suppose that the current execution trace is $\mathcal{H}^1$, and that a new happened event $\mathbf{H}(e,t)$ is acquired by SCIFF. Let us denote the new execution trace with $\mathcal{H}^2$, having $\mathcal{H}^2 = \mathcal{H}^1 \cup \{\mathbf{H}(e,t)\}$. If the execution of the system always grows by increasing times, then $t > ct(\mathcal{H}^1)$. Therefore, from Definition 6 it holds that $\mathcal{H}^1 \prec \mathcal{H}^2$ and, in turn, Lemma 4 ensures that $MVI(\mathcal{R}_{\mathcal{H}^1}) \trianglelefteq MVI(\mathcal{R}_{\mathcal{H}^2})$, i.e. that the new state is a progressive extension of the previous one. □

[7] For example, if the user states that "event e initiates fluent f at time T if fluent f_2 holds at time $T+2$", then it could be the case that at time $T+2$ f_2 indeed holds, causing f to be declipped in the past and violating condition 2 of Definition 7

6 Conclusions

The $\mathcal{EC}$ is a powerful framework for reasoning about time and events. We identified the problem applying the $\mathcal{EC}$ to runtime monitoring and tracking systems. We observed that there is no satisfactory solution to it in the state of the art. Specifically, related approaches mainly boil down to ad-hoc, tailored procedures that are not easily modifiable and whose formal properties are not easy to determine. Moreover, they do not guarantee properties that we deem fundamental in this domain, namely irrevocability. We therefore provided the first formal and operational approach to the problem, a $\mathcal{REC}$ implementation in SCIFF.

We proved that $\mathcal{REC}$ enjoys soundness, completeness and irrevocability: in other words, it is a reactive version of the $\mathcal{EC}$ which matches the domain requirements. Irrevocability holds for a class of well-formed $\mathcal{REC}$ theories. Soundness and completeness instead hold for the broader class of theories defined by well-formed SCIFF theories [1]. Thus $\mathcal{REC}$ can also reason on non well-formed $\mathcal{REC}$ theories, but in that case the value of fluents in the past may change as events in the future are processed, which is not at all surprising.

In a companion paper [9] we show how to realize a commitment tracking framework for multi-agent systems using $\mathcal{REC}$. Future work will focus on performance evaluation and on the integration of $\mathcal{REC}$ with the other forms of reasoning enabled by SCIFF, mainly with abduction.

References

1. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logic*, 9(4):1–43, 2008.
2. F. Chesani. *Specification, execution and verification of interaction protocols: an approach based on computational logic*. PhD thesis, University of Bologna, 2007. Available at `http://amsdottorato.cib.unibo.it/392/`.
3. L. Chittaro and A. Montanari. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence*, 12:359–382, 1996.
4. L. Chittaro and A. Montanari. Temporal representation and reasoning in artificial intelligence: Issues and approaches. *AMAI*, 28(1-4):47–106, 2000.
5. K. Eshghi. Abductive planning with event calculus. In *Proc. 5th ICSLP*:562–579. MIT Press, 1988.
6. T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, Nov. 1997.
7. R. A. Kowalski and F. Sadri. Towards a unified agent architecture that combines rationality with reactivity. *Logic in Databases*, LNCS 1154:137–149. Springer, 1996.
8. R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
9. M. Montali, F. Chesani, P. Mello, and P. Torroni. Commitment tracking via the reactive event calculus, 2009. Accepted at IJCAI 2009. Available upon request.
10. M. Shanahan. The event calculus explained. In *Artificial Intelligence Today*, LNAI 1600:409–430, Springer, 1999.
11. M. Shanahan. An abductive event calculus planner. *J. Log. Program.*, 44(1-3):207–240, 2000.