Commitment Tracking via the Reactive Event Calculus*

Federico Chesani, Paola Mello, Marco Montali, and **Paolo Torroni**
DEIS, University of Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{federico.chesani, paola.mello, marco.montali, paolo.torroni}@unibo.it

Abstract

Runtime commitment verification is an important, open issue in multiagent research. To address it, we build on Yolum and Singh's formalization of commitment operations, on Chittaro and Montanari's cached event calculus, and on the SCIFF abductive logic programming proof-procedure. We propose a framework consisting of a declarative and compact language to express the domain knowledge, and a reactive and complete procedure to track the status of commitments effectively, producing provably sound and irrevocable answers.

1 Introduction

Since the introduction of social semantics of agent interaction [Singh, 1998], *social commitments* have become a central notion in multiagent research. One of the main reasons why they became popular is that they lend themselves to verification. Agents are treated as if they were black boxes, which modify the environment via actions, and send/receive messages. Interaction protocols are specified outside of agents, independently of their architecture. Whether agents interact properly or not, that depends on their external behaviour, but not on their internal status.

By interacting or by performing actions, agents manipulate commitments [Singh, 1999], whose status could be inspected to verify how agents are behaving, or what is expected of them. This form of verification should ideally be carried out at runtime, and it is what we call *commitment tracking*. Being verifiability one of social semantics' main motivations, one would expect that commitment tracking is a routine task which makes use of well-known procedures: but this is not the case, and runtime verification is an open issue to date.[1]

We thus propose a framework for commitment tracking based on SCIFF and on the *Event Calculus*. The framework features a flexible, declarative encoding of the Event Calculus and of the theory of commitments, which can easily be customized and extended. The commitment theory we implement is mainly based on Yolum and Singh's formalization [2002], but it also has temporal elements in the style of Mallya and Huhns [2003] since it features an explicit treatment of time. This is necessary for real-world applications, where temporal constraints and deadlines are ubiquitous.

After giving the necessary background notions, we informally discuss the requirements of a commitment tracking framework. We then present our language and procedure, discuss its formal properties, and show a working example.

2 The Event Calculus

The Event Calculus ($\mathcal{EC}$) was introduced by Kowalski and Sergot [1986] for representing and reasoning about events and their effects. Basic concepts are that of *event*, happening at a point in time, and *property* (or *fluent*), holding during time intervals. Fluents are initiated/terminated by occurring events. There are many different axiomatizations of the $\mathcal{EC}$. A simple one, taken from [Chittaro and Montanari, 2000], is the one below (P stands for *Property*, E for *Event*):

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

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

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

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

Axioms ec_1 and ec_2 are the general ones of $\mathcal{EC}$, whereas ec_3 and ec_4 are used-defined, domain-specific axioms. Sometimes *initially*(P) is used to define properties that hold at the beginning of time. Dual axioms and predicates, such as *declipped* can be added to define when properties do *not* hold and to model actions with duration [Shanahan, 1999].

The $\mathcal{EC}$ framework has been extensively used in the literature to carry out two main reasoning tasks: deductive *narrative verification*, to check whether a certain fluent holds, given a narrative (set of events) [Kowalski and Sergot, 1986], and abductive *planning*, to simulate a possible narrative which satisfies some requirements [Shanahan, 2000]. These tasks take place after or prior to execution, but not during execu-

*Work supported by the FIRB RBNE05BFRK project TOCAI.it.

[1]Notably, two EU projects, FP5's "SOCS" and FP6's "CONTRACT" are largely concerned with runtime verification.

tion. The reason why the $\mathcal{EC}$—in its above formulation—is not used at runtime is that each time an event occurs, it enables a straightforward update of the theory (it suffices to add *happens* fact), 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 such a powerful framework. For this reason, Chittaro and Montanari [1996] proposed a mechanism to cache the outcome of the inference process every time the knowledge base is updated by a new event. The *Cached Event Calculus* ($\mathcal{CEC}$) computes and stores the *maximum validity intervals* (MVIs) of fluents, i.e., the maximum time intervals in which fluents hold, according to the known events. The set of cached MVIs is then extended/revised as new events occur or get to be known.

3 Social commitments

Social commitments are commitments made from an agent to another agent to bring about a certain property. They are a well-known concept in Multi-Agent Systems (MAS) research [Castelfranchi, 1995; Singh, 1999]. Representing the commitments that the agents have to one another and specifying constraints on their interactions in terms of commitments provides a principled basis for agent interactions [Torroni *et al.*, 2009]. Commitments also serve as a natural tool to resolve design ambiguities. Finally, the formal semantics enables verification of conformance and reasoning about the MAS specifications to define core interaction patterns and build on them by reuse, refinement, and composition.

Central to the whole approach is the idea of manipulation of commitments: their creation, discharge, delegation, assignment, cancellation, and release. Commitments are stateful objects that change in time as events occur. Time and events are, therefore, essential elements. Some authors distinguish between *base-level* commitments, written $C(x, y, p)$, and *conditional* commitments, written $CC(x, y, p, q)$ (x, y are agents, called *debtor* and *creditor*; p, q are properties). $CC(x, y, p, q)$ signifies that if p is brought out, x will be committed towards y to bring about q.

The $\mathcal{EC}$ is a suitable formalism to specify the effects of commitment manipulation and reason upon such operations. Yolum and Singh [2002] have shown how commitments can be embedded in a logical framework based on the $\mathcal{EC}$. As a sample fragment of such a formalization, consider a *create* operation, whose purpose is to establish a commitment, and can only be performed by the debtor. To express that an event $e(x)$ carried out by x at time t creates a commitment c, Yolum and Singh define the operation *create*$(e(x), C(x, y, p))$ in terms of *happens*$(e(x), t) \wedge$ *initiates*$(e(x), C(x, y, p), t)$.

4 Tracking social commitments

Example 1. *A customer has signed a service agreement with a printer supplier: if a printer breaks down, the supplier guarantees to send a technician on site. The technician must intervene within three days from the call. Any delay in the intervention will incur from the supplier's side an obligation to pay a* \$10 *penalty per day of delay, as of the fourth day.*

The supplier's contractual obligations could be represented by commitments. *Commitment tracking* is the automated process of verifying the status of commitments. In the example above, it can serve to verify that the service agreement is respected and, in case of delays, that the corresponding penalty is paid. A commitment tracking framework should comprise a language and a procedure, implemented into a tool.

The *language* must be expressive enough to define (i) obligations (commitments), (ii) deadlines to be respected, and (iii) compensations actions, such as those arising from deadline expiration.

The *procedure* keeps track of the obligations at each given moment. It should lend itself to an efficient implementation, to enable early detection of expiring deadlines. Most importantly, it must provide some *guarantees*. First of all, its output must be provably sound. Moreover, it should be complete, in the sense that it should provide all the relevant information about the relevant commitments. A tool that sometimes "forgets" to indicate some duties cannot be trusted. Finally, it must provide stable, irrevocable answers.

5 The Reactive Event Calculus

The $\mathcal{EC}$ can be elegantly formalized using logic programming, but as we mentioned above, that would be suitable for top-down, backward computation, and not for runtime monitoring. For this reason, we resort to a framework which reconciles backward with forward reasoning: the SCIFF language and proof-procedure [Alberti *et al.*, 2008].

SCIFF is an extension of Fung an Kowalski's IFF proof-procedure for abductive logic programming [1997], in which abduction is adopted to enable forward reasoning via *integrity constraints*. SCIFF has two primitive notions: events (mapped as $\mathbf{H}$ atoms) and expectations (mapped as $\mathbf{E}$/$\mathbf{EN}$ atoms). $\mathbf{H}(E, T)$ means that an event E has occurred at time T, and it is a ground atom. Atoms $\mathbf{E}(E, T)$ and $\mathbf{EN}(E, T)$ instead are not necessarily ground. They can contain variables with domains and be associated with *constraint logic programming* (CLP) constraints. $\mathbf{E}(E, T)$ denotes that an event unifying with E is expected to occur at some time in the range of T. $\mathbf{EN}(E, T)$ denotes that all events unifying with E are expected to not occur, at all times in the range of T.

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. The verification features of SCIFF are discussed in relation with alternative temporal logic-based approaches by Montali *et al.* [2008].

A SCIFF specification is composed of a knowledge base $\mathcal{P}$, a set of ICs (integrity constraints) $\mathcal{IC}$, a set of abducible expectations $\mathcal{A}$, 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 events in its $body$ 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 [Alberti *et al.*, 2008] is given for a given specification and a narrative, denoted by $\mathcal{H}$, i.e., a set of **H** atoms. Intuitively, it states that $\mathcal{P}$, together with the abduced expectations, 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. The distinguishing feature of our SCIFF implementation of the $\mathcal{EC}$ is that, thanks to SCIFF, it is not goal-directed but event-driven, thus *reactive*. Thus its name, *reactive event calculus* ($\mathcal{REC}$). The status of fluents is updated at runtime as events occur. In this sense, we draw inspiration from Chittaro and Montanari's idea of MVIs.

The basic predicates of the calculus are presented below (Axioms ax_1 through ax_7). Events and fluents are terms and times are integer (CLP) variables, 0 being the "initial" time. $\mathcal{REC}$ uses the abduction mechanism to generate MVIs and define their persistence. As opposed to $\mathcal{CEC}$, which is implemented by a special-purpose algorithm, $\mathcal{REC}$ has a fully declarative axiomatization, and thanks to the SCIFF framework no ad-hoc implementation is needed. $\mathcal{REC}$ uses two special internal events (*clip*/*declip*) to model that a a fluent is initiated/can be terminated. 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 later below.

Axiom 1. *A fluent F holds at time T if an MVI containing T has been abduced for F:*[2]

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

Axiom 2. *If $(T_s, T_e]$ is an 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:*

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

Axiom 3. *If a fluent initially holds, a corresponding declipping event is generated at time 0:*

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

Axiom 4. *If an event E initiating a fluent F occurs at time T, either F already holds or it is declipped:*

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

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

Axiom 5. *The happening of a* declip(F) *event causes fluent F to start holding:*

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

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

Axiom 6. *If an event E terminates a fluent F, F is clipped:*[3]

$$\begin{aligned} &\mathbf{H}(\mathit{event}(E),T) \wedge \mathit{terminates_at}(E,F,T) \\ &\rightarrow \mathbf{H}(\mathit{clip}(F),T). \end{aligned} \quad (ax_6)$$

Axiom 7. *A (special)* complete *event terminates all fluents:*

$$\mathit{terminates}(\mathit{complete},F). \quad (ax_7)$$

6 Formal properties of $\mathcal{REC}$

$\mathcal{REC}$ is implemented on top of SCIFF, it thus inherits its soundness and completeness results of the declarative semantics with respect to the SCIFF's operational 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 declarative semantics, augmented with the $\mathcal{REC}$ axioms ax_1–ax_7.*

These are important results. The $\mathcal{REC}$ operational behaviour of is faithful to its specifications, i.e., it returns all and only correct answers with respect to the $\mathcal{EC}$ axiomatization given in Section 5. We are unaware of other reactive implementations of the $\mathcal{EC}$ that provide such a guarantee.

The next results concern uniqueness and irrevocability, and they are especially significant for the commitment tracking application. We thus need to introduce some information about the operational behaviour of SCIFF.

6.1 Open, closed and semi-open reasoning

SCIFF features two main forms of derivation, called open and closed. Given a specification $\mathcal{S}$ and two narratives $\mathcal{H}^i$ and $\mathcal{H}^f \supseteq \mathcal{H}^i$, if there exists an *open successful derivation* [Chesani, 2007] 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.[4] 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 to be violated but only *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 upon narratives known to be complete, or to close the reasoning process when the flow of events comes to an end. In that case, both **E** and **EN** atoms are evaluated: a *closed world assumption* is made about the collected execution trace, and pending expectations are considered to be violated, because no further event will occur to fulfill them.

SCIFF is sound and complete independently of the order in which it acquires and processes 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

[3]Although (ax_6) could be defined symmetrically to (ax_4), the present (equivalent) formulation produces a better performance.

[4]Below we omit $\mathcal{H}^i$ since $\mathcal{H}^f \supseteq \mathcal{H}^i$ and Δ only depends on $\mathcal{H}^f$.

on the past can thus be evaluated immediately. This form of *semi-open* reasoning is achieved by a rule, inside the SCIFF proof-procedure, which states that if an execution trace has reached time t, then all pending expectations must be fulfilled by some time $t' \geq t$. *Semi-open* derivation is denoted by $\models\!\!\sim$.

6.2 Irrevocability of $\mathcal{REC}$

The commitment tracking domain enables semi-open derivation. It would be desirable that MVIs once generated are never retracted. If that is the case, the inference process is said to be *irrevocable*.

We restrict ourselves to cases in which tracking makes sense. To this end, we define "well-formed" theories, which capture the notion of causality (today's events have no impact on yesterday's status of fluents). We then show that semi-open reasoning from such theories is irrevocable.

Definition 1 (Well-formed $\mathcal{REC}$ theory). *A well-formed $\mathcal{REC}$ theory $\mathcal{T}$ is a set of clauses*[5]

$$\textit{initiates}(E, F, T) \leftarrow \textit{body}.$$
$$\textit{terminates}(E, F, T) \leftarrow \textit{body}.$$

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, $T_2 \leq T$.*

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 following SCIFF specification:*[6]

$$\mathcal{R}^{\mathcal{T}} \equiv \langle \mathrm{KB}_{\mathcal{REC}} \cup \mathcal{T}, \{\mathbf{E}, \mathbf{EN}, \textbf{\textit{mvi}}\}, \mathrm{IC}_{\mathcal{REC}} \rangle \qquad (1)$$

where $\mathrm{KB}_{\mathcal{REC}} = \{(ax_1), (ax_7)\}$, $\{\mathbf{E}, \mathbf{EN}, \textbf{\textit{mvi}}\}$ *is the set of all possible expectations and MVIs, and* $\mathrm{IC}_{\mathcal{REC}} = \{(ax_2), (ax_3), (ax_4), (ax_5), (ax_6)\}$.

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

Lemma 1. *For each well-formed $\mathcal{REC}$ theory $\mathcal{T}$ and execution trace $\mathcal{H}$, given the goal true, the abduced MVIs always have a ground starting time, i.e.,*

$$\forall\ \Delta,\ \mathcal{S} \models\!\!\sim^{\mathcal{H}}_{\Delta} true \Rightarrow \forall\ \textbf{\textit{mvi}}(F, [T_s, T_e]) \in \Delta, T_s \in \mathbb{N}.$$

Lemma 2. *The expectation about the clipping of an MVI can be fullfilled by exactly one happened event, in particular the nearest one occurring after the declipping of the MVI.*

Lemma 3. *In order for a fluent to be declipped by two distinct events, at least one clipping event must occur in between.*

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. $\mathcal{S} \models\!\!\sim^{\mathcal{H}}_{\Delta} true$.*

[5]The *body* can be omitted when *true*.

[6]Below, $\mathcal{R}$ denotes a generic $\mathcal{REC}$ specification. For brevity, we will say that a $\mathcal{REC}$ specification is well-formed when its theory is.

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 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 defines irrevocability.

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

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

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

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

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 MVIs whose termination is a ground time (*closed MVIs*):*

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

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

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

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* progessive 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 elements, these are declipped after $ct(\mathcal{H}^1)$:*
   $\forall\ \textbf{\textit{mvi}}(f, [s, t]) \in \mathrm{MVI}(\mathcal{R}_{\mathcal{H}^2}) \backslash \mathrm{MVI}(\mathcal{R}_{\mathcal{H}^1}), s > ct(\mathcal{H}^1)$
3. $\forall\ \textbf{\textit{mvi}}(f, [s, T_e]) \in \mathrm{MVI}_{\sqcap}(\mathcal{R}_{\mathcal{H}^1})$*, either*
   (a) *it remains untouched in the new state:*
   $\textbf{\textit{mvi}}(f, [s, T_e]) \in \mathrm{MVI}_{\sqcap}(\mathcal{R}_{\mathcal{H}^2})$*, or*
   (b) *it is clipped after $ct(\mathcal{H}^1)$:*
   $\textbf{\textit{mvi}}(f, [s, e]) \in \mathrm{MVI}_{\sqcap\!\!\sqcup}(\mathcal{R}_{\mathcal{H}^2}), e > ct(\mathcal{H}^1)$.

Progressive extensions capture the intuitive notion that a state extends another one if it keeps the already computed closed MVIs as they are, and it affects only the status that fluents assume after the latest time of the first state. The extension is determined by adding new MVIs and by clipping fluents which used to hold at the previous state. We can state the main result related to irrevocability: extending a trace results in a progressive extension of the state of MVIs.

Lemma 4. *Given a well-formed $\mathcal{REC}$ specification $\mathcal{R}$ and two execution traces $\mathcal{H}^1$ and $\mathcal{H}^2$,*

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

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

Therefore $\mathcal{REC}$, used in combination with a theory of commitments, fulfills all the requirements identified in Section 4. The language offers a declarative, intuitive language for representing obligations, deadlines, and compensation actions. The inference procedure provides sound, complete and irrevocable answers at runtime.

7 Commitment tracking via $\mathcal{REC}$

We demonstrate the features of $\mathcal{REC}$ using the example introduced in Section 4. We adopt the formalization of commitments in $\mathcal{EC}$ given by Yolum and Singh [2002]. The resulting framework accommodates conditional commitments, although for lack of space we do not illustrate them. We focus instead on deadlines and compensations. To enable runtime monitoring, we assume that an external clock is available, and that special *tic* events signal the passing of time.

7.1 Detecting deadline expiration

To illustrate flexibility and expressiveness, we extend the commitments theory—twice: by adding temporal constraints, and by introducing a finer-grained notion of violation. This notion distinguishes between a "partial" violation (a deadline has expired but there may be a belated make-up action) and a "full" violation (too late). We consider a very common temporal constraint: a relative deadline about the discharging of the commitment. The idea is that the user can specify that a certain commitment must be fulfilled within a certain interval T_D, as of the time the commitment has been established. The user specifies T_D, and if the commitment has been established at (absolute) time T, then it should be satisfied within the (absolute) time $(T + T_D)$.

We first introduce the fluents representing the status of commitments.

For each $c(X, Y, P)$, a fluent *waiting*$(c(X, Y, P))$ holds if $c(X, Y, P)$ has been established, and the deadline has not expired yet. An commitment discharging event has two effects: it terminates the *waiting*$(c(X, Y, P))$ fluent, and it instantiates a new *satisfied*$(c(X, Y, P))$ fluent, meaning that the commitment has been successfully discharged.

$$\begin{aligned} &\mathit{initiates}(E, waiting(c(X, Y, P)), T) \\ &\quad \leftarrow \mathit{create}(E, X, c(X, Y, P)). && (ex_1) \\ &\mathit{terminates}(E, waiting(c(X, Y, P)), T) \\ &\leftarrow \mathit{holds_at}(\mathit{waiting}(c(X, Y, P)), T), \\ &\quad \mathit{discharge}(E, X, c(X, Y, P)). && (ex_2) \\ &\mathit{initiates}(E, \mathit{satisfied}(c(X, Y, P)), T) \\ &\leftarrow \mathit{holds_at}(\mathit{waiting}(c(X, Y, P)), T), \\ &\quad \mathit{discharge}(E, X, c(X, Y, P)). && (ex_3) \end{aligned}$$

We then introduce a fluent *d_check*(F, T_D), meaning that a commitment F should be satisfied by T_D.

The fluent can be instantiated as follows:

$$\begin{aligned} &\mathit{initiates}(E, \mathit{d_check}(c(X, Y, P), \mathit{When}), T) \\ &\quad \leftarrow \mathit{create}(E, X, c(X, Y, P)), \\ &\quad \mathit{deadlines}(c(X, Y, P), \mathit{Delay}), && (ex_4) \\ &\quad \mathit{When}\ is\ T + \mathit{Delay}. \end{aligned}$$

where *deadlines*$(c(X, Y, P), \mathit{Delay})$ is a user-defined fact stating that commitment c should be satisfied within *Delay* time units from its instantiation.

If the deadline expires and the commitment is still *waiting*, the *status* of the commitment becomes *partially violated*: partially, because the deadline has expired, but something discharging the commitment could still happen. A fluent *p_viol*$(c(X, Y, P), \mathit{When})$ indicates that a deadline for c has expired, while c should have been satisfied by time *When*.

$$\begin{aligned} &\mathit{initiates}(\mathit{tic}, \mathit{p_viol}(c(X, Y, P), \mathit{When}), T) \\ &\quad \leftarrow \mathit{holds_at}(\mathit{d_check}(c(X, Y, P), \mathit{When}), T), \\ &\quad \mathit{holds_at}(\mathit{waiting}(c(X, Y, P)), T). && (ex_5) \end{aligned}$$

Axioms ex_1–ex_5 represent this new "theory of deadlines" for social commitments. It is a fully customizable theory, which a user can define and apply to many problems.

A domain-specific knowledge base specifying the example discussed in Section 4 is instead specified as follow:

$$\mathit{create}(\mathit{printer_broken}, \mathit{shop}, c(\mathit{shop}, \mathit{us}, \mathit{repair})). \quad (ex_6)$$

$$\mathit{deadlines}(c(\mathit{shop}, A, \mathit{repair}), 3). \quad (ex_7)$$

$$\mathit{fulfills}(\mathit{work_on_printer}, \mathit{repair}). \quad (ex_8)$$

ex_6 states that the event *printer_broken* establish the commitment of the supplier, towards us, to repair the printer; ex_7 adds the information that such type of commitment should fulfilled within 3 days form the commitment establishment; finally, ex_8 specifies that the event *work_on_printer* satisfies any commitment about the *repair* action.

7.2 Compensation

Users can also define *compensation* axioms. Compensation mechanisms come in hand when commitments are violated. They are important to tackle undesired situations, and to add robustness to the overall system. In our example, a compensation consists of a new commitment for the supplier to pay a penalty fee, whose amount depends on how many days have passed since the deadline expired at the time the printer is repaired. Note that until the technician is on site, it is not possible to correctly evaluate the extent of the new commitment. Such a situation is captured by the following axiom:

$$\begin{aligned} &\mathit{initiates}(E, c(\mathit{shop}, Y, \mathit{pay_penalty}(M)), T) \\ &\leftarrow \mathit{holds_at}(\mathit{p_viol}(c(\mathit{shop}, Y, \mathit{repair}), \mathit{When}), T), \\ &\quad \mathit{discharge}(E, \mathit{shop}, c(\mathit{shop}, Y, \mathit{repair})), && (ex_9) \\ &\quad M1\ \mathit{is}\ \mathit{When} - T,\ M\ \mathit{is}\ M1 * 10. \end{aligned}$$

where the new commitment *pay_penalty* is instantiated by any event E discharging the repair commitment, if such a repair commitment has a deadline expired at time *When*. The penalty is calculated based on a difference between two variables.

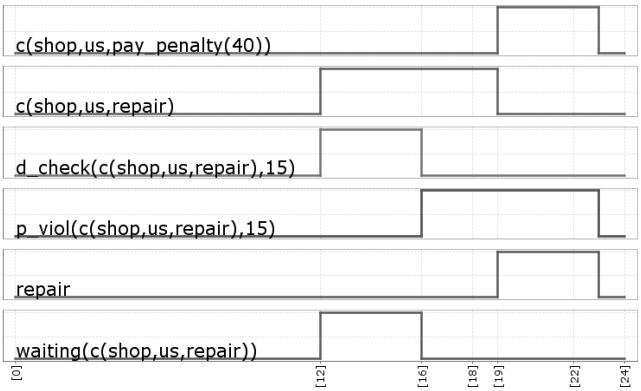

Figure 1: Output of the $\mathcal{REC}$-based tool

7.3 Running a monitoring process

Let the following events be observed at runtime:

$h(event(printer_broken), 12)$ $h(event(work_on_printer), 19)$
$h(event(tic), 16)$ $h(event(end_monitoring), 22)$
$h(event(tic), 18)$

In this narrative, the printer breaks down, and the technician arrives on site seven days later. Our example includes a penalty for each day passed after the third one. We then expect the printer supplier to be charged \$40 as a compensation.

Figure 1 shows the runtime output of $\mathcal{REC}$+SCIFF. At time 12 the printer breaks down (*printer_broken*). Three fluents are thus instantiated: $c(shop, us, repair)$, meaning that the shop gets committed to repair the printer, following ex_6; $waiting(c(\ldots))$, meaning that the commitment is waiting to be satisfied (ex_1); and $d_check(c(\ldots), 15)$, meaning that the commitment c must be satisfied by time 15 (ex_4).

Nothing happens until the clock signals time 16 (*tic*), when the deadline check fluent is clipped (it stops holding), because of the deadline expiring (the axiom about such fluent's termination has not been reported here). At the same time the fluent $p_viol(c(\ldots), 15)$ is initiated (and *waiting* terminated), meaning that commitment $c(shop, us, repair)$ has been partially violated, i.e., it has not been discharged within the deadline (following ex_5). Note that the commitment is not clipped. That is an arbitrary choice we made in this example, and implementing a different behaviour could be achieved by simply adding a termination axiom.

At time 19, a technician arrives on site and repairs the printer, which terminates $c(shop, us, repair)$. A *repair* fluent is initiated and indicates that such a property has been achieved. The *waiting* fluent is instead terminated. Since the commitment to repair the printer was partially violated, a new commitment about paying the fee is created (ex_9). Note that since the technician arrived at time 19, while he was supposed to intervene by time 15, *shop* must pay a four-day, \$40 fee.

8 Conclusions

We identified the problem of commitment tracking in multi-agent systems. We observed that there is no solution to it in the state of the art. Specifically, we are not aware of any other work directly related to commitment tracking which satisfies some fundamental requirements of the language and of the formal implement. Related work on runtime verification of commitments and contracts mainly consist of ad-hoc, tailored procedures that are not easily modifiable and whose formal properties are not easy to determine. We therefore provided the first formal and operational approach to the problem. We showed that in order to address it one can use a $\mathcal{REC}$ implementation in SCIFF. 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

[Alberti *et al.*, 2008] M. Alberti, F. Chesani. M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable Agent Interaction in Abductive Logic Programming: The SCIFF framework. *ACM Trans. Comp. Log.*, 9(4):1–43, 2008.

[Castelfranchi, 1995] C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proc. 1st ICMAS*:41–48. The MIT Press, 1995.

[Chesani, 2007] F. Chesani. *Specification, execution and verification of interaction protocols*. PhD dissertation, University of Bologna, Italy, 2007.

[Chittaro and Montanari, 1996] L. Chittaro and A. Montanari. Efficient temporal reasoning in the cached event calculus. *Comp. Int.*, 12:359–382, 1996.

[Chittaro and Montanari, 2000] L. Chittaro and A. Montanari. Temporal representation and reasoning in AI: Issues and approaches. *Ann. Math. & AI*, 28(1-4):47–106, 2000.

[Fung and Kowalski, 1997] T. Fung and R. Kowalski. The IFF proof procedure for abductive logic programming. *J. Log. Prog.*, 33(2):151–165, Nov. 1997.

[Kowalski and Sergot, 1986] R. Kowalski and M. Sergot. A logic-based calculus of events. *NGC*, 4(1):67–95, 1986.

[Mallya and Huhns, 2003] A. Mallya and M. Huhns. Commitments among agents.*IEEE Int.Comp.*7(4):90–93, 2003.

[Montali *et al.*, 2008] M. Montali *et al*. Verification from declarative specifications using logic programming. In *Proc. 24th ICLP*, *LNCS* 5366:440–454. Springer, 2008.

[Shanahan, 1999] M. Shanahan. The event calculus explained. In *AI Today*, *LNAI* 1600:409–430. Springer, 1999.

[Shanahan, 2000] M. Shanahan. An abductive event calculus planner. *J. Log. Prog.*, 44(1-3):207–240, 2000.

[Singh, 1998] M. Singh. Agent communication language: rethinking the principles. *IEEE Comp.*:40–47, Dec 1998.

[Singh, 1999] M. Singh. An ontology for commitments in multiagent systems. *AI & Law*, 7:97–113, 1999.

[Torroni *et al.*, 2009] P. Torroni, P. Yolum, M. Singh, M. Alberti, F. Chesani. M. Gavanelli, E. Lamma, and P. Mello Modelling interactions via commitments and expectations. In *Handbook of Research on MAS: Semantics and Dynamics of Organizational Models*, Chap. XI. IGI Global, 2009.

[Yolum and Singh, 2002] P. Yolum and M. Singh. Flexible protocol specification and execution. *Proc. AAMAS*, 2002.