Runtime Verification of Agent Properties

Stefania Costantini[1], **Pierangelo Dell'Acqua**[2], **Luís Moniz Pereira**[3], and **Panagiota Tsintza**[1]

[1] Dip. di Informatica, Università di L'Aquila, Coppito 67100, L'Aquila, Italy
stefcost@di.univaq.it
[2] Dept. of Science and Technology - ITN, Linköping University, Norrköping, Sweden
pierangelo.dellacqua@itn.liu.se
[3] Departamento de Informática, Centro de Inteligência Artificial (CENTRIA), Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. In previous work, we have proposed a multi-level agent model with at least a meta-level aimed at meta-reasoning and meta-control. In agents, these aspects are strongly related with time and therefore we retain that they can be expressed by means of temporal-logic-like rules. In this paper, we propose an interval temporal logic inspired by METATEM, that allows properties to be verified in specific time interval situated either in the past or in the future. We adopt this logic for definition and run-time verification of properties which can imply modifications to the agent's knowledge base.

1 Introduction

Agents are by definition software entities which interact with an environment, and thus are subject to modify themselves and evolve according to both external and internal stimuli, the latter due to the proactive and deliberative capabilities of the agent themselves (whenever encompassed by the agent model at hand). In past work, we defined semantic frameworks for agent approaches based on logic programming that account for: (i) the kind of evolution of reactive and proactive agents due to directly dealing with stimuli, that are to be coped with, recorded and possibly removed [1]; and (ii) the kind of evolution related to adding/removing rules from the agent knowledge base [2]. These frameworks have been integrated into an overall framework for logical evolving agents (cf. [3, 4]) where every agent is seen as the composition of a base-level (or object-level) agent program and one or more meta-levels. In this model, updates related to recoding stimuli are performed in a standard way, while updates involving the addition/deletion of sets of rules, related to learning, belief revision, etc. are a consequence of meta-level decisions.

As agent systems are more widely used in real-world applications, the issue of verification is becoming increasingly important (see [5] and the many references therein). In computational logic, two common approaches to the verification of computational systems are model checking [6] and theorem proving. There are many attempts to adapt these techniques to agents (see again [5]). In this paper, we address the problem concerning the monitoring of agent behavior against desired properties, or with respect to a

certain specification, in a different way. We assume defined, possibly both at the object and at the meta-level, axioms that determine properties to be respected or enforced, or simply verified, whenever a property is desirable but not mandatory. We assume these properties to be verified at runtime. Upon verification of a property (which is evaluated within a context instantiated onto the present circumstances), suitable actions can be undertaken, that we call in general *improvement action*. Improvements can imply *revision* of the agent knowledge, or tentative *repair* of malfunctioning, or tentative improvement of future behavior, according to the situation at hand. Our approach is to some extent similar to that of [7] for evolving software.

As many of the properties to be defined and verified imply temporal aspects, we have considered to adopt the temporal logic METATEM [8, 9] as this logic is specially tailored to an agent context. Since properties should often be defined on certain intervals, we define a variant of METATEM, that we call I-METATEM, where connectives are defined over intervals. We do not adopt the full power of METATEM rules, where connectives are interpreted as modalities, and semantics provided accordingly. Instead, we remain within the realm of logic programming, and interpret the temporal axioms in the context of the above-mentioned semantic framework. Therefore, we should better call our axioms "temporal-logic-like" axioms. However, to simulate to some extent the power of modal logic, improvements can imply the removal/addition of new temporal-logic-like axioms. The addition of new ones determines their immediate operational use. In this way, we stay within our semantic framework, where we are able to provide a full declarative semantics and an efficient corresponding operational semantics, as demonstrated by the existing implementations ([2, 10]), though the whole proposed approach has not been fully implemented yet.

The plan of the paper is as follows. In Section 2 we summarize the features of the agent model our framework is based upon. This model is general, and many existing agent-oriented logic languages can be easily rephrased in terms of it. In Section 3 we shortly summarize the METATEM temporal logic, and then introduce the proposed extension. In Section 4 we show how we mean to use temporal-logic-like rules for defining properties, how these properties are meant to be verified, and we establish our notion of improvement. Then we conclude.

2 Layered Agent (Meta-)Model

We do not mean to restrict the proposed approach to one single agent model/language. Therefore, we refer to an abstract agent model (we might say, meta-model). In this way, the approach can be adapted to any specific agent formalism which can be seen as an instance of this meta-model. We therefore refer to the abstract multi-layer meta-framework for agents proposed in [3, 4]. In this framework, an agent is considered to be composed of two distinct interacting levels: the BA (standing for Base-Agent) and one or more meta-levels. BA is the base level, whereas MA (Meta-Agent) along with IEA (Information Exchange Agent) constitute the two meta-levels. Here we assume that BA is a logic program and make the additional assumption that its semantics may ascribe multiple models to BA in order to deal with "uncertainty". For the semantics of logic

programs we can adopt one of those reported in the survey [11] and for the semantics dealing with "uncertainty" we can adopt the Answer Set Semantics [12].

The meta-level, by means of both components MA and IEA, performs various kinds of meta-reasoning and is responsible for supervising and coordinating the BA's activities. MA is in charge of coordinating all activities and takes decisions over the BA. More precisely, the MA level will be the one up to decide which modifications have to be undertaken onto the BA level, in order to correct or improve inadequacies and unexpected behavior. The IEA level instead decides and evaluates when an interaction with the society is necessary in order to exchange knowledge: in fact, agents are in general not entities standing alone but, rather, are part of possibly several groups to form a society. Below we do not give any formal definition of BA, MA and IEA as their actual form depends upon the various possible concrete instances of the meta-framework. Rather, we specify the requirements they have to obey. We also define the overall architecture, and outline a possible semantic account.

2.1 Agent Model: operational behavior

To define the operational behavior of the agent meta-model we exploit our previous work reported in [3, 4]. Each agent is considered as a logic program that will evolve interacting with the environment. In fact, the interaction triggers many agent activities such as response, goals identification, decisions on recording and pruning the gathered information. Of course, these activities will be affected by the belief, desire and intention control that is part of MA. Note that this component will itself evolve and change in time as a result of the interaction with the society. In this paper, we consider the evolution of the initial agent into subsequent related versions of the agent itself. Therefore, we consider that each interaction will, eventually, determine the evolution of the initial agent in terms of successive transformations.

We start by providing a more formal view of agent evolution. We consider a generic instance of our agent meta-model that we refer to as $\mathcal{M}$. The agent model $\mathcal{M}$ will have to provide an agent-oriented programming language and a control mechanism. For example, if $\mathcal{M}$ provides a prolog-like programming language, $\mathcal{C}$ may be a meta-interpreter, and $\mathcal{CI}$ may be a set of directives to the interpreter. Below we describe the operational behavior of our meta-model thus providing a specification to which $\mathcal{M}$ should conform, whatever its specific functioning, to be seen as an instance of our framework.

Definition 1. *An* agent program *is a tuple* $\langle BA, MA, \mathcal{C}, \mathcal{CI} \rangle$ *of software components where* BA *and* MA *are logic programs,* $\mathcal{C}$ *is the control component and* $\mathcal{CI}$ *is the component containing control information.*

Specifically, the control component $\mathcal{C}$ takes as input both the logic programs BA and MA and the control information $\mathcal{CI}$. The $\mathcal{CI}$ component has the role of customizing the *run-time* behavior of the agent. For example, $\mathcal{CI}$ may contain directives stating the priorities among different events/goals that the agent has to cope with.

In the following, we clarify how the control $\mathcal{C}$ and control information $\mathcal{CI}$ components are enabled to actually affect the operational behavior of agents. In fact, both components are taken as input by an underlying control mechanism $\mathcal{U}$ that implements

the operational counterpart of the agent model. For example, if the agent model provides a prolog-like programming language the underlying control mechanism may be either an interpreter or a virtual machine related to a compiler. The underlying control mechanism $\mathcal{U}$ that is able to put into operation the various components of an agent model $\mathcal{M}$, is a transformation function starting from A_0 and transforming it step by step into A_1,A_2, ... The transition from a generic step A_i into the next step A_{i+1} is defined as follows.

Definition 2. *Let $A_i = \langle BA_i, MA_i, \mathcal{C}_i, \mathcal{CI}_i \rangle$ be an agent program at step i. Then, the* underlying control mechanism *$\mathcal{U}$ is a binary function defined as:*

$$A_i \rightarrow^{\mathcal{U}(\mathcal{C}_i, \mathcal{CI}_i, w_i)} \langle BA_{i+1}, MA_{i+1}, \mathcal{C}_{i+1}, \mathcal{CI}_{i+1} \rangle.$$

It is important to notice that, given an initial step A_0, subsequent steps $A_i s$ in general do not follow deterministically. The reason is that each step depends both on the interaction with the society w_i (external environment) and on the internal choices of each agent that are based on its previous knowledge and "experience". The underlying control mechanism $\mathcal{U}$ can operate in two different ways by providing: (i) either different parallel threads for BA and MA, or (ii) an interleaving of control between the two levels. In the former case, MA continuously monitors BA. In the latter case control must somehow pass between the two levels, for instance as follows. Control will shift up from BA to MA periodically (and/or upon certain conditions) by means of an act called *upward reflection*. When controls shifts up, MA will revise the BA's activities, which may imply constraints and condition verification. Vice versa, control will shift down from MA to BA by performing an act called *downward reflection*. Forms of control based on reflection in computational logic are formally accounted for in [13]. The frequency as well as the conditions of each type of shift is defined in the control information component $\mathcal{CI}_i$ and therefore can be encoded as a subset of directives included in this component. A declarative semantics for evolving agents that fulfills the above-proposed meta-model is presented in [1]. Dynamic changes that MA can operate on BA can be semantically modeled by means of the approach of Evolving Logic Programs [14]. In the following, we will assume these formalizations as the semantic bases of the approach proposed here.

The meta-model and its operational behavior are consistent at least with the KGP ([15–17]) and DALI ([10, 18, 19]) agent-oriented languages.

3 I-METATEM: Temporal Logic in the proposed framework

In the previous section, we discussed the non determinism of states that can be reached by agents during their evolution. For defining temporal-logic-like rules while keeping the complexity under control, we are going to adapt the approach of METATEM, a propositional Linear-time Temporal Logic (LTL), that implicitly quantifies universally upon all possible paths. LTL logics are called linear because, in contrast to branching time logics, they evaluate each formula with respect to a vertex-labeled infinite path $s_0 s_1 \ldots$ where each vertex s_i in the path corresponds to a point in time (or "time instant" or "state"). In order to model the dynamic behavior of agents, we propose an extension

to the well-established METATEM logic called I-METATEM, an acronym standing for Interval METATEM.

3.1 METATEM

In this subsection, we present the basic elements of propositional METATEM logic [8, 9]. Its language is based both on the classical propositional logic enriched by temporal connectives and on the direct execution of temporal logic statements. The symbols used by METATEM are: (i) a set A_C of propositions controlled by the component which is being defined, (ii) a set A_E of propositions controlled by the environment (where $A_C \cap A_E = \emptyset$), (iii) the alphabet of propositional symbols $A_P = A_C \cup A_E$, (iv) a set of propositional connectives such as **true**, **false**, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\Leftrightarrow$, and (v) a set of temporal connectives. The set of temporal connectives is composed of a number of unary and binary connectives referring to future-time and past-time. A METATEM program is a set of temporal logic rules of the form:

$$\textit{past time antecedent} \rightarrow \textit{future time consequent}$$

where the *past time antecedent* is considered as a temporal formula concerning the past while the *future time consequent* is a temporal formula concerning the present and future time. Therefore, a temporal rule is the one determining how the process should progress through stages.

3.2 I-METATEM

The purpose of this extension is to allow properties and anomalous behavior in agent evolution to be checked at run-time. Since agent evolution can be considered as an infinite sequence of states, it is often not possible (and not suitable) to verify properties on the entire sequence. Sometimes it is not even desirable, since one needs properties to hold within a certain time interval. Thus we propose the extension I-METATEM to the METATEM logic. Specifically, we introduce the new connectives $F_{m,n}$, $G_{m,n}$, $G_{\langle m,n\rangle}$, $N_{m,n}$, $E_{m,n}$, $\widehat{G}_{m,n}$ and $\widehat{G}_{\langle m,n\rangle}$.

Future-time connectives of I-METATEM (Assume $m < n$)

- X (*next state*). $X\varphi$ states that φ will be true at next state.
- G (*always in future*). $G\varphi$ means that φ will always be true in every future state.
- F (*sometime in future*). $F\varphi$ states that there is a future state where φ will be true.
- W (*weak until*). $\varphi W \psi$ is true in a state s if ψ is true in a state t, in the future of state s, and φ is true in every state in the time interval [s,t) where t is excluded.
- U (*strong until*). $\varphi U \psi$ is true in a state s if ψ is true in a state t, in the future of state s, and φ is true in every state in the time interval [s,t] where t is included.
- N (*never*). $N\varphi$ states that φ should not become true in any future state.
- τ (*current state*). $\tau(i)$ is true if s_i is the current state.
- X_m (*future m-state*). $X_m\varphi$ states that φ will be true in the state s_{m+1}.

F_m (*bounded eventually*). $F_m\varphi$ states that φ eventually has to hold somewhere on the path from the current state to s_m.

$F_{m,n}$ (*bounded eventually in time interval*). $F_{m,n}\varphi$ states that φ eventually has to hold somewhere on the path from state s_m to s_n.

$G_{m,n}$ (*always in time interval*). $G_{m,n}\varphi$ states that φ should become true at most at state s_m and then hold at least until state s_n.

$G_{\langle m,n\rangle}$ (*strong always in time interval*). $G_{\langle m,n\rangle}\varphi$ states that φ should become true just in s_m and then hold until state s_n, and not in s_{n+1}.

$N_{m,n}$ (*bounded never*). $N_{m,n}\varphi$ states that φ should not be true in any state between s_m and s_n.

$E_{m,n}$ (*sometime in time interval*). $E_{m,n}\varphi$ states that φ has to occur one or more times between s_m and s_n.

Past-time connectives of I-METATEM (Assume $m < n$)

$\widehat{X}$ (*last state*). $\widehat{X}\varphi$ states that if there is a last state, then φ was true in that state.

$\widehat{F}$ (*some time in the past*). $\widehat{F}\varphi$ states that φ was true in some past state.

$\widehat{G}$ (*always in the past*). $\widehat{G}\varphi$ states that φ was true in all past states.

$\widehat{Z}$ (*weak since*). $\varphi\widehat{Z}\psi$ is true in a state s if ψ was true in a state t (in the past of state s), and φ was true in every state of the time interval [t,s).

$\widehat{S}$ (*since*). $\varphi\widehat{Z}\psi$ is true in a state s if ψ was true in a state t (in the past of state s), and φ was true at every state in the time interval [t,s].

$\widehat{X}_m$ (*last m-state*). $\widehat{X}_m\varphi$ states that φ was true in the past state s_{m-1}.

$\widehat{F}_m$ (*bounded some time in the past*). $\widehat{F}_m\varphi$ states that φ was true in some past state before s_m included.

$\widehat{G}_{m,n}$ (*always in time interval in the past*). $\widehat{G}_{m,n}\varphi$ states that φ was true in the past state s_m and then it remained true at least until the past state s_n.

$\widehat{G}_{\langle m,n\rangle}$ (*strong always in time interval in the past*). $\widehat{G}_{\langle m,n\rangle}$ states that φ became true just in the past state s_m and then remained true exactly until the past state s_n.

The syntax of I-METATEM temporal formulae is given by the following definition.

Definition 3. *Formulae of I-METATEM logic are defined inductively in the usual way:*

$$\varphi ::= p \mid \mathit{true} \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \tau(i)$$
$$\varphi ::= X\varphi_1 \mid \varphi_1 U \varphi_2 \mid \widehat{X}\varphi_1 \mid \varphi_1 \widehat{S} \varphi_2$$
$$\varphi ::= \varphi_1 U_{m,n} \varphi_2 \mid \varphi_1 \widehat{S}_{m,n} \varphi_2$$
$$\varphi ::= (\varphi_1)$$

where $p \in A_P$, and φ_1 and φ_2 are formulae of I-METATEM.

Semantics of I-METATEM After having introduced the syntax of I-METATEM, we present the semantics of I-METATEM formulae. To do so, we first recall the notion of model structures to be used in the interpretation of temporal formulae. In the following, let σ be a sequence of states $s_0 s_1 ...$ and i a time instant. A *structure* is a pair

$\langle \sigma, i \rangle \in (\mathbb{N} \rightarrow 2^{A_P})\ x\ \mathbb{N}$ where A_P is the alphabet of propositional symbols. Given some moment in time, represented by a natural number j, $\sigma(j)$ is the set of propositions drawn from the alphabet A_P and denotes all the propositions that are true at time j. The satisfaction relation $\models$ gives the interpretation to temporal formulae in the given model structure.

Definition 4. (Semantics of I-METATEM temporal formulae) *Let $\langle \sigma, i \rangle$ be a structure. The semantics of I-METATEM temporal logic is defined as follows.*

Propositions and propositional connectives

$$\begin{array}{ll}
\langle \sigma, i \rangle \models p & \textit{iff } p \in \sigma(i) \\
\langle \sigma, i \rangle \models \textit{true} & \\
\langle \sigma, i \rangle \models \neg\varphi & \textit{iff } \langle \sigma, i \rangle \not\models \varphi \\
\langle \sigma, i \rangle \models \varphi \wedge \psi & \textit{iff } \langle \sigma, i \rangle \models \varphi \textit{ and } \langle \sigma, i \rangle \models \psi \\
\langle \sigma, i \rangle \models \tau(i) &
\end{array}$$

Temporal connectives

$$\begin{array}{lll}
\langle \sigma, i \rangle \models X\varphi & \textit{iff} & \langle \sigma, i{+}1 \rangle \models \varphi \\
\langle \sigma, i \rangle \models \varphi U \psi & \textit{iff} & \exists k \in \mathbb{N}\ \langle \sigma, i{+}k \rangle \models \psi \textit{ and } \forall j\ (0 \leq j < k)\ \langle \sigma, i{+}j \rangle \models \varphi \\
\langle \sigma, i \rangle \models \widehat{X}\varphi & \textit{iff} & \textit{if } i > 0, \textit{ then } \langle \sigma, i{-}1 \rangle \models \varphi \\
\langle \sigma, i \rangle \models \varphi \widehat{S} \psi & \textit{iff} & \exists k\ (1 \leq k \leq i)\ \langle \sigma, i{-}k \rangle \models \psi \textit{ and } \forall j\ (1 \leq j < k)\ \langle \sigma, i{-}j \rangle \models \varphi
\end{array}$$

Temporal connectives in time intervals

$$\begin{array}{lll}
\langle \sigma, i \rangle \models \varphi\, U_{m,n}\, \psi & \textit{iff} & i \leq m \leq n, \exists k\,(m - i \leq k \leq n - i)\ \langle \sigma, i + k \rangle \models \psi \textit{ and} \\
 & & \forall j\ (0 \leq j < k - i)\ \langle \sigma, i + j \rangle \models \varphi \\
\langle \sigma, i \rangle \models \varphi\, \widehat{S}_{m,n}\, \psi & \textit{iff} & m \leq n < i, \exists k\,(i - m \leq k \leq i - n)\ \langle \sigma, i - k \rangle \models \psi \textit{ and} \\
 & & \forall j\ (1 \leq j < k)\ \langle \sigma, i - j \rangle \models \varphi
\end{array}$$

In addition, we use the following abbreviations:

$$\begin{array}{rcl}
\text{false} & \widehat{=} & \neg\text{true} \\
\varphi \vee \psi & \widehat{=} & \neg(\neg\varphi \wedge \neg\psi) \\
\varphi \Rightarrow \psi & \widehat{=} & \neg\varphi \vee \psi
\end{array}$$

$$\begin{array}{rcl}
F\varphi & \widehat{=} & \text{true}\, U \varphi \\
G\varphi & \widehat{=} & \neg F \neg\varphi \\
\varphi W \psi & \widehat{=} & \varphi U \psi \vee G\varphi \\
N\varphi & \widehat{=} & \neg F \varphi
\end{array}$$

$$\widehat{F}\varphi \mathrel{\widehat{=}} \text{true}\ \widehat{S}\ \varphi$$
$$\widehat{G}\varphi \mathrel{\widehat{=}} \neg\widehat{F}\neg\varphi$$
$$\varphi\widehat{Z}\psi \mathrel{\widehat{=}} \varphi\widehat{S}\psi \vee \widehat{G}\varphi$$

$$X_m\varphi \mathrel{\widehat{=}} G_{m+1,m+1}\varphi$$
$$F_m\varphi \mathrel{\widehat{=}} \bigvee_{i=1}^{m}(\ \tau(i) \wedge \text{true}\ U_{i,m}\ \varphi\ )$$
$$G_{m,n}\varphi \mathrel{\widehat{=}} \neg(\text{true}\ U_{m,n}\ \neg\varphi)$$
$$G_{\langle m,n\rangle}\varphi \mathrel{\widehat{=}} \neg F_m\varphi \wedge G_{m,n}\varphi \wedge G_{n+1,n+1}\neg\varphi$$
$$E_{m,n}\varphi \mathrel{\widehat{=}} \neg G_{m,n}\varphi \wedge \neg N_{m,n}\varphi$$

$$\widehat{X}_m\varphi \mathrel{\widehat{=}} \widehat{G}_{m-1,m-1}\varphi$$
$$\widehat{F}_m\varphi \mathrel{\widehat{=}} \bigvee_{i=1}^{m-1}(\ \tau(i) \wedge \text{true}\ \widehat{S}_{m,i-1}\ \varphi\ )$$
$$\widehat{G}_{m,n}\varphi \mathrel{\widehat{=}} \neg(\text{true}\ \widehat{S}_{m,n}\ \neg\varphi)$$
$$\widehat{G}_{\langle m,n\rangle}\varphi \mathrel{\widehat{=}} \widehat{G}_{0,m}\neg\varphi \wedge \widehat{G}_{m,n}\varphi \wedge \widehat{G}_{n+1,n+1}\neg\varphi$$

Let $\sigma \vDash \varphi$ be an abbreviation for $\langle\sigma, 0\rangle \vDash \varphi$. We call σ a *model* of φ iff $\sigma \vDash \varphi$.

4 I-METATEM for defining and verifying properties in logical agents

In our framework, agents are supposed to live in an open society where they interact with each other and with the environment, and where they can learn either by observing other agents behavior or by imitation. Given the evolving nature of learning agents, their behavior has to be checked from time to time and not (only) "a priori". Model checking and other "a priori" approaches are static, since the underlying techniques require to write an ad-hoc interpreter and this operation cannot be re-executed whenever the agent learns a new piece of information. Note that in case of re-execution this operation would in principle be required a huge number of times, adding a further cost to the system. Moreover, an a priori full validation of agent's behavior would have to consider all possible scenarios that are not known in advance. These are the reasons why we propose (also) a run-time control on agent behavior and evolution, for checking correctness during agents activity, rather than a model checking control.

To do so, we add to the underlying logic programming agent-oriented language the possibility of specifying rules including I-METATEM connectives. These rules can be defined both at the object level and at the meta-level to determine properties to be respected. These rules will be attempted, and whenever verified they may determine suitable modifications to the program itself. In the rest of this section, we first define the syntax of I-METATEM connectives in the context of logic programs, and then we define I-METATEM basic rules, I-METATEM contextual rules, and I-METATEM rules with improvements. Along with the explanation we provide some examples.

In our framework, we consider I-METATEM rules to be applied upon universally quantified formulae. Note that the negation connective *not* is interpreted in our setting as negation by default. To increase readability in I-METATEM rules, we write any temporal connective of the form $O_{m,n}$ as $O(m, n)$. Sometimes, we omit the connective arguments when implied from the context, and in these cases we write O instead of $O(m, n)$. Also, as a special case, when we do not care about the starting point of an interval, we introduce the special constant $start$ where $O(start, n)$ means that O is checked since the "beginning of time" up to n, where the beginning of time coincides with the agent's activation time. We also introduce the shorthand now standing for the time t for which $\tau(t)$ holds.

4.1 I-METATEM rules

In this section we present rules with an associated remedial action to be performed any time the rule is violated. We start by introducing the notion of basic rule and contextual rule.

I-METATEM basic rules We start by first introducing the basic form of rules.

Definition 5. *Any I-METATEM formula φ is a rule.*

Whenever checked, an I-METATEM rule is verified (or *succeeds*) whenever φ holds, otherwise the rule is violated. In the case of I-METATEM connectives (or their negation), this means that the related property holds either in the specified interval (if elapsed) or up to now. According to the semantic framework of [1], where special formulas can be designated to be periodically executed, I-METATEM rules will be periodically attempted (we also say "checked"). We assume some default frequency whenever not explicitly defined. As a first example of an I-METATEM rule, consider the following:

$$N(\, goal(g) \wedge deadline(g, t) \wedge now(T) \wedge T \leq t \wedge not\ achieved(g) \wedge dropped(g)\,)$$

We assume predicates $goal$, $achieved$ and $dropped$ to be suitably defined in the agent's knowledge base. Informally: $goal(g)$ means that g is the goal that has to be achieved; $achieved(g)$ is deemed true when the plan for reaching the goal g has been successfully completed; $dropped(g)$ means that agent has dropped any attempt to achieve g. The rule states that it cannot be the case that a given goal not accomplished up to now but not expired yet (the deadline t for this goal has not been met), is dropped by the agent. There are in principle different ways to exploit this rule: (i) as an "a priori" check to be performed whenever a $drop$ action is attempted; if the check fails, then the action is not allowed and (ii) as an "a posteriori" check on the agent behavior; in case of violation, some repair action should presumably been undertaken, as discussed below.

Notice that for performing this kind of evaluation we have to consider ground rules. In the above rule in fact, the only variable is the present time T, which is however instantiated by the predefined connective now. Below we generalize to the non-ground case.

I-METATEM contextual rules For the sake of generality, and in view of a changing environment, we propose a further extension of rule syntax to include variables instantiated by an *evaluation context* associated to each rule.

Definition 6. *A contextual I-METATEM rule is a rule of the form $\chi \Rightarrow \varphi$ where:*

- *χ is the evaluation context of the rule, and consists of a conjunction of logic programming literals;*
- *every variable occurring in φ must occur in the context χ.*

From Definition 6 it follows that the evaluation of a contextual rule becomes feasible only when grounded from the context. In order to clarify the syntax of a *contextual I-METATEM rule*, consider the following example:

$$[\ goal(Goal), priority(Goal, Pr), timeout(Pr,\ T\_out)\ ] \Rightarrow \\ F(T\_out)\ achieved(Goal)$$

In this rule, the goal $Goal$ is established by the context, which also contains the atoms $priority$ and $timeout$. Informally, the rule requires that a goal with timeout T_out (set according to the priority Pr of the goal itself), should actually be accomplished before the timeout. This contextual rule can be verified whenever instantiated to a specific goal g. In general, the rule will be repeatedly checked until it either succeeds, if g will be achieved in time, or it fails because the timeout will have elapsed.

I-METATEM rules with improvement Whenever an instance of an I-METATEM rule succeeds, it either expresses a desirable property or not. In the former case, some kind of "positive" action may be undertaken; in the latter case, a repair action will in general be required. We call the corresponding modification of the program an *improvement* or *remedial action*. Program modification/evolution is accounted for by the EVOLP semantics [2, 14].

We now extend the definition of contextual I-METATEM rules to specify a corresponding improvement action.

Definition 7. *Let A be an atom. An I-METATEM rule with improvement is a rule of the form $\chi \Rightarrow \varphi$:A where $\chi \Rightarrow \varphi$ is a contextual rule, and A is its improvement action.*

Given a rule $\chi \Rightarrow \varphi : A$, whenever its monitoring condition $\chi \Rightarrow \varphi$ holds, then the rule is checked and the corresponding improvement action A is attempted. The improvement action is specified via an atom that is executed as an ordinary logic programming goal. Consider again the previous example which monitors the achievement of goals, but extended to specify that, in case of violation, the current level of commitment of the agent to its objectives has to be increased. This can be specified as:

$$[goal(Goal), deadline(Goal,\ T), now(T2), T2 \leq T] \Rightarrow \\ G\,(\neg\, achieved(Goal) \wedge dropped(Goal)\,) : inc\_comt(T2)$$

$$incr\_comt(T2) \leftarrow commitment\_level(T, L), \\ increase\_level(L, L2), \\ assert(not\ commitment\_level(T, L)), \\ assert(commitment\_level(T2, L2))$$

Suppose that at a certain time t the monitoring condition

$$G\,(\neg\, achieved(Goal) \wedge dropped(Goal)\,)$$

holds for some specific goal g. Upon detection, the system will attempt the improvement action consisting in executing the goal $inc\_comt(t)$. Its execution will allow the system to perform the specified run-time re-arrangement of the program that attempts to cope with the unwanted situation: in the example, the module defining the rules that specify the level of commitment to which the agent obeys is retracted and substituted by a new one corresponding to a higher level.

Semantically, in our agent meta-model the execution of the improvement action will determine the update of the current agent program $\mathcal{P}_i$, returning a new agent program $\mathcal{P}_{i+1}$. The I-METATEM rules with improvements are to some extent similar to METATEM rules, though here one does not state properties of the future but rather specifies actions to be undertaken.

Based on this definition, we are able for instance to define rules that aim at detecting various kinds of anomalous behavior of an agent (for a discussion of run-time anomalies see, e.g., [20]). For example, we can introduce a rule for checking an unexpected behavior such as **omission**, which occur whenever an agent fails to perform the desired action/goal. The rule:

$$[\,goal(Goal), not\ achieved(Goal), dropped(Goal, T3), confidence(G, T3, C3)\,] \Rightarrow$$
$$G\,(\,confidence(G, now, C) \wedge C3 \leq C\,) : re\_exec(Goal)$$

states how the agent has to behave in the case of a dropped goal. If, after dropping the goal (because it has not been achieved in a given interval), the agent's confidence in being able to achieve the goal has increased, then the goal will be re-attempted.

To detect an anomalous behavior consisting of **duplication** or **incoherence**, i.e., an agent performs more than once the same action/goal when not necessary, we introduce the following rule:

$$[\,goal(Goal)\,] \Rightarrow \widehat{F}_0\ times\_exec(Goal) > k : disable(Goal)$$

with the role of checking if a goal/plan has been executed more times than a given threshold: if so, further execution of the goal will be disabled.

The following example outlines the so-called anomaly of **intrusion**, i.e., the case of an unexpected behavior, or unwanted consequence, arisen from the execution of a goal. Whenever the constraint defined below succeeds, as a repair a new constraint is asserted establishing that G cannot be further pursued, at least until a certain time has elapsed. As soon as asserted, the new constraint will start being checked.

$$[\,now(T), goal(Goal), executed(Goal), consequence(Goal, C)\,] \Rightarrow \widehat{F}_{0,T}\,\neg desired(C)\ :$$
$$assert(\,[\,now(T), threshold(T1)\,] \Rightarrow N(T, T1)\ exec(Goal)\,)$$

I-METATEM connectives can be used to check the past behavior and knowledge of the agent but also to influence its future behavior. The agent evolution entails also an evolution of recorded information, which in turn may affect the evaluation of social factors such as trust and confidence. Consider for instance the following example, where the level of trust is increased for agents that have proved themselves reliable in communication during a test interval. The increase of the level of trust is modeled as an

improvement. Notice that the improvement is determined based on recorded information, that is, every agent that will pass the test will have its trust level increased as soon as the rule with improvement is executed.

$$[\, agent(Ag), now(T)\,] \Rightarrow G(m,n)\ reliable(Ag) : assert(rel\_ag(Ag,T))$$
$$rel\_ag(Ag,\, T) \Rightarrow\ true : increase\_trust\_level(Ag)$$

5 Related Work

In this section we discuss other related approaches to verification starting by those using declarative programming.

Alberti et al [21] addressed the problem of verifying system's specifications by using abductive logic programming. In particular, they proposed the SCIFF framework, an abductive logic programming language and proof system for the specification and runtime verification of interaction protocols. The authors developed an abductive proof-procedure (called g-SCIFF) which is used to prove properties at design time and to generate counter-examples of properties that do not hold. g-SCIFF is proven to be sound and complete with respect to its declarative semantics [21]. In [22] the g-SCIFF proof procedure is proved to be a valid alternative to other verification methods by experimentally comparing its performance with the one of some well-known model checker. The SCIFF proof-procedure is based on the notion of events, for example, sending a message or starting an action. Events are associated with time points. SCIFF uses integrity constraints ICs to model relations among events and expectations about events. Expectations are abducibles identified by the functors E (for positive expectations) and EN (for negative expectations). Events and time variables are constrained by ICs taking the form $body \Rightarrow head$. The $body$ is a conjunction of happened events, expectations and constraints, whereas the $head$ is a disjunction of conjunctions of positive and negative expectations. For example, the rule $H(a,T) \Rightarrow E(b,T2) \wedge T2 \leq T + 300$ states that if a occurs, then the event b should occur no later than 300 time units after a. The SCIFF approach is related to our proposed approach in a number of ways. Most of our interval temporal connectives can be simply expressed by SCIFF rules except for example the *until* connective. Moreover, in our logic (as well as in METATEM logic) temporal connectives can be composed in several ways with both temporal and logical connectives, while this is not possible with E and EN abducibles. For example, $E\,(a \wedge\ (EN(b,T), T2)\,)$ cannot be expressed.

McIlraith et al consider the problem of planning and monitoring in dynamic and uncertain domains (cf. [23–25]). To address real world planning problems, they extend classical planning to incorporate procedural control knowledge and temporally extended user preferences into the specification of the planning problem. These preferences are somewhat similar to the METATEM connectives. For example, if a preference establishes $eventually(\varphi)$, they want the planner to choose actions that will lead to the satisfaction of φ. However, their approach substantially differs from our since we consider intervals over METATEM formulae.

Bauer et al [26, 27] consider runtime verification for realtime systems emitting events at dedicated time points. A logic suitable for expressing properties of such a

system is *timed lineartime temporal logic* (TLTL), which is a timed variant of LTL (originally introduced by Raskin in [28]). TLTL allows one to express typical bounded response properties, for example requiring that an event a occurs within three time units. Note that such a property can be expressed in LTL with then formula $\varphi \equiv XXX\, a$. However, as discussed in [26], this formulation presumes a direct correspondence of discrete time delays with incoming events (that is, every time point corresponds to an incoming event). In contrast, what we want to express is that the event a occurs after three time units regardless of how many other events occur in between. Following [29] the authors consider an TLTL logic that is a timed variant of LTL, called LTL_{ec}. Its syntax includes the two new atomic formulae: $\triangleleft_a \in I$ and $\triangleright_a \in I$. The first asserts that the time since the event a has occurred the last time lies within then interval I. Analogously, $\triangleright_a \in I$ asserts that the time until a occurs again lies within I. For example, the formula:

$$G\,(\,request \rightarrow \triangleright_{acknowledge} \in [0,5]\,)$$

means that if a request event arrives, then it must be handled with an acknowledge event within 5 time units. These formulae can be expressed in I-METATEM by means of its temporal operators in time intervals. The formula above can be expressed as:

$$[\,now(T)\,] \Rightarrow G(\,request \Rightarrow F_{0,T+5}\; acknowledge).$$

Thus, I-METATEM logic is at least as expressive as the TLTL logic. Our conjecture is that the two logics have the same expressive power. For example, the I-METATEM connective $U_{m,n}$ can be represented in TLTL as follows:

$$\langle \sigma, i \rangle \vDash \varphi\, U_{m,n}\, \psi \qquad (i \leq m \leq n)$$

$$\equiv$$

$$\langle \sigma, i \rangle \vDash \bigvee_{x=m}^{n} (\triangleright_\psi \in [x,x] \wedge \Gamma)$$

where:

$$\Gamma = \begin{cases} true & \text{if } x-1 < i \\ \bigwedge_{y=i}^{x-1} (\triangleright_\varphi \in [y,y]) & \text{otherwise} \end{cases}$$

6 Concluding Remarks

We have introduced an approach to the definition and the run-time verification of properties of agent behavior that has elements of novelty: in fact, we adopt a temporal logic with connectives defined on intervals in order to define and verify the run-time behavior of agents evolution; we are able to undertake suitable repairing actions based on the verification of properties and, as the underlying abstract agent model includes meta-level(s), these actions may imply modifications to the agent's knowledge base.

At the moment, the implementation of the approach has not been completed yet. Thus, we do not make any claim on its performance and complexity. Future work includes a full implementation of the approach, the development of suitable case-studies in significant application realms such as, e.g., ambient intelligence, and theoretical developments aimed at coping with challenging contexts, e.g., learning.

References

1. Costantini, S., Tocchio, A.: About declarative semantics of logic-based agent languages. In Baldoni, M., Torroni, P., eds.: Declarative Agent Languages and Technologies. LNAI 3904. 106–123
2. Alferes, J.J., Brogi, A., Leite, J.A., Pereira, L.M.: Evolving logic programs. In: Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002. LNAI 2424, Springer-Verlag, Berlin (2002) 50–61
3. Costantini, S., Tocchio, A., Toni, F., Tsintza, P.: A multi-layered general agent model. In: AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, 10th Congress of the Italian Association for Artificial Intelligence. LNCS 4733, Springer-Verlag, Berlin (2007)
4. Costantini, S., Dell'Acqua, P., Pereira, L.M.: A multi-layer framework for evolving and learning agents. In M. T. Cox, A.R., ed.: Proceedings of Metareasoning: Thinking about thinking workshop at AAAI 2008, Chicago, USA. (2008)
5. Fisher, M., Bordini, R.H., Hirsch, B., Torroni, P.: Computational logics and agents: a road map of current technologies and future trends. Computational Intelligence Journal **23**(1) (2007) 61–91
6. Clarke, E.M., Lerda, F.: Model checking: Software and beyond. Journal of Universal Computer Science **13**(5) (2007) 639–649
7. Barringer, H., Rydeheard, D., Gabbay, D.: A logical framework for monitoring and evolving software components. In: TASE '07: Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, Washington, DC, USA, IEEE Computer Society (2007) 273–282
8. Barringer, H., Fisher, M., Gabbay, D., Gough, G., Owens, R.: MetateM: A framework for programming in temporal logic. In: Proceedings of REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. LNCS 430, Springer-Verlag (1989)
9. Fisher, M.: Metatem: The story so far. In Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E., eds.: PROMAS. LNCS 3862, Springer (2005) 3–22
10. Costantini, S., Tocchio, A.: The DALI logic programming agent-oriented language. In: Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004. LNAI 3229, Springer-Verlag, Berlin (2004)
11. Apt, K.R., Bol, R.: Logic programming and negation: A survey. The Journal of Logic Programming **19-20** (1994) 9–71
12. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Logic Programming, Proc. of the Fifth Joint Int. Conf. and Symposium, MIT Press (1988) 1070–1080
13. Barklund, J., Dell'Acqua, P., Costantini, S., Lanzarone, G.A.: Reflection principles in computational logic. J. of Logic and Computation **10**(6) (2000) 743–786
14. J.Alferes, J., Brogi, A., Leite, J.A., Pereira, L.M.: An evolvable rule-based e-mail agent. In: Procs. of the 11th Portuguese Intl.Conf. on Artificial Intelligence (EPIA'03). LNAI 2902, Springer-Verlag, Berlin (2003) 394–408
15. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP model of agency: Computational model and prototype implementation. In: Global Computing: IST/FET International Workshop, Revised Selected Papers. LNAI 3267. Springer-Verlag, Berlin (2005) 340–367
16. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of agency. In: Proc. ECAI-2004. (2004)
17. Stathis, K., Toni, F.: Ambient Intelligence using KGP Agents. In Markopoulos, P., Eggen, B., Aarts, E.H.L., Crowley, J.L., eds.: Proceedings of the 2nd European Symposium for Ambient Intelligence (EUSAI 2004). LNCS 3295, Springer Verlag (2004) 351–362

18. Tocchio, A.: Multi-Agent systems in computational logic. PhD thesis, Dipartimento di Informatica, Università degli Studi di L'Aquila (2005)
19. Costantini, S., Tocchio, A.: A logic programming language for multi-agent systems. In: Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf.„JELIA 2002. LNAI 2424, Springer-Verlag, Berlin (2002)
20. Costantini, S., Tocchio, A.: Memory-driven dynamic behavior checking in logical agents. In: Electr. Proc. of CILC'06, Italian Conference of Computational Logic. (2006) URL: http://cilc2006.di.uniba.it/programma.html.
21. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The SCIFF framework. ACM Trans. Comput. Log. **9**(4) (2008)
22. Montali, M., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verification from declarative specifications using logic programming. In Banda, M.G.D.L., Pontelli, E., eds.: 24th Int. Conf. on Logic Programming (ICLP). LNCS 5366, Udine, Italy, Springer Verlag (December 2008) 440–454
23. Baier, J., Bacchus, F., McIlraith, S.: A heuristic search approach to planning with temporally extended preferences. In: Proc. 20th Int. J. Conf. on Artificial Intelligence (IJCAI-07). (2007) 1808–1815
24. Baier, J.A., Fritz, C., Bienvenu, M., McIlraith, S.: Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In: Proc. 23rd AAAI Conf. on Artificial Intelligence (AAAI), Nectar Track. (2008) 1509–1512
25. Fritz, C., McIlraith, S.A.: Monitoring policy execution. In: Proc. 3rd Workshop on Planning and Plan Execution for Real-World Systems. (2007) (at ICAPS07).
26. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. Technical Report TUM-I0724, Institut für Informatik, Technische Universität München (December 2007)
27. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In Arun-Kumar, S., Garg, N., eds.: Proc. 26th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LNCS 4337, Berlin, Heidelberg, Springer-Verlag (2006)
28. Raskin, J.F.: Logics, Automata and Classical Theories for Deciding Real Time. PhD thesis, Institut d'Informatique, FUNDP, Namur, Belgium (1999)
29. D'Souza, D.: A logical characterisation of event clock automata. Int. J. Found. Comput. Sci. **14**(4) (2003) 625–640