Specification and Dynamic 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
`piede@itn.liu.s`
[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 have defined semantic frameworks for agent approaches based on logic programming that account for: 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]; 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] and [4]) where moreover every agent is seen as the composition of a base-level (or ground-level or object-level) agent program and one or more meta-layers. 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 agents behavior against desired properties, or w.r.t. 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, with a certain frequency associated with the property itself depending upon its criticality. 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* actions, or simply *improvements*: 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 a temporal logic, and our choice has fallen on METATEM [8] [9]. Since properties should often be defined on certain intervals, we define a variant of METATEM, that we call A-IMETATEM, where operators are defined over intervals. We do not adopt the full power of METATEM rules, where operators 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 ([10], [2]), though the 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 very 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*. We then 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 (or, 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] and [4]. In this framework, an agent is considered to be composed of two distinct interacting layers: the BA (or *base layer*, or *ground layer*) and (one or more) Meta-level(s). The BA is a base level, whereas the MA (that stands for Meta-Agent) along with the IEA (Information Exchange Agent), constitutes the Meta-Level. Here we assume that the BA is a logic program and make an 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 suggest, e.g., the Answer Set Semantics proposed in [12].

The Meta-level, by means of both components MA and IEA, performs various kinds of meta-reasoning and is the responsible for supervising and coordinating the BA's activities. The MA meta-level is in charge of coordinating all activities and takes decisions over the BA layer. More precisely, the MA layer will be the one up to decide which modifications have to be undertaken onto the BA level, in order to correct (or improve) inadequacies or unexpected behavior. The IEA layer, instead, is the one deciding and evaluating 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 (one or more) group(s) called "society". Below we do not give a formal definition of the BA, MA and IEA as their actual form depends upon the various possible concrete instances of the meta-framework. Rather, we specify the requirements the have to obey. We also define the overall architecture, and outline a possible semantic account.

2.1 Agent Model: the knowledge base

Below we outline the requirements for the knowledge base of an agent in the proposed framework. In line with [13] and with the BDI (*beliefs*, *desires*, *intentions*) model (see [14] as an example of a BDI framework), the components forming the BA and/or the MA layers include the following.

1. Both BA and MA layer contain in general a *belief* component, enclosing modules for reasoning, planning, goal identification, reactivity and proactivity, etc.
2. The BA and MA layers also contain a set of *desires* (called SD) and *intentions* (called SI). SD includes all goals adopted or under considerations while the SI is composed by all plans both in execution and consideration.
3. In addition to the components proposed in [4], we consider the (optional) components of *ability* and *confidence* enclosing modules of: *trust*, *abilities* of agents in computing certain action and *confidence*. The latter component is responsible for reasoning about the confidence of the agent in doing something (an action, a goal, etc.) and can be influenced by other agents in the society.
4. In order to interact with the society, there will be a component encompassing the agent's interaction mechanisms.
5. In order to manage beliefs as well as confidence (and their alterations), a specific component may be provided, which should possibly include a learning mechanism.

All components mentioned in the above points have to be combined, exploited, and supervised by a control component. This component is based on control information aimed at improving control effectiveness.

2.2 Agent Model: operational behavior

To define the operational behavior of the agent meta-model we exploit our previous work reported in [3]. Each agent is considered as a logic program that will evolve by its interaction with the environment. In fact, the interaction triggers many agent activities such as response, goals identification, decisions on recording or pruning the gathered information, etc. Of course, these activities will be affected by the belief, desire and intention control that is part of the agent's 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 are going to 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.

Here is 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, $\mathcal{CI}$ 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 $\mathcal{P}_{\mathcal{M}}$ *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* $\mathcal{CI}$ *(optional) contains some kind of control information.*

In previous definition, we consider that 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. E.g., $\mathcal{CI}$ may contain directives may state the priorities among different events/goals that the agent has to cope with, and the frequency those properties have to be taken into consideration.

As mentioned before, the initial agent is considered as a logic program, to which we associate the *initial state of the agent*. More formally, the initial agent is given by means of the following definition.

Definition 2. *The* initial agent A_0 *is an agent program* $\mathcal{P}_{0\mathcal{M}}$ *(or simply* $\mathcal{P}_0$ *when* $\mathcal{M}$ *is clear from the context), i.e.,* $\langle BA_0, MA_0, \mathcal{C}_0, \mathcal{CI}_0\rangle$*, where* BA_0*,* MA_0 *are the initial logic programs, and* $\mathcal{C}_0$*,* $\mathcal{CI}_0$ *are, respectively, the initial control and control information components.*

In the following, we are going to state that 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 $\mathcal{U}^{\mathcal{M}}$ 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.

Definition 3. *The* underlying control mechanism $\mathcal{U}^{\mathcal{M}}$ *(or* $\mathcal{U}$ *in short), able to put into operation the various components of an agent model* $\mathcal{M}$*, is a transformation function operating in terms of a set of distinguishable* steps*, starting from* A_0 *and transforming it step by step into* A_1*,* $A_2, \ldots,$ *given* $\mathcal{C}_i$ *and* $\mathcal{CI}_i$ *as defined in* A_0, A_1, A_2*,... respectively.*

Next, we consider that the transition from a generic step A_i into the next step A_{i+1} is defined as follows:

Definition 4. *Let $\mathcal{P}$ be an agent program. Then,* $\forall i \geq 0$, $A_i \rightarrow^{\mathcal{U}(\mathcal{C}_i, \mathcal{CI}_i)} A_{i+1}$.

It is important to notice that, given an initial step A_0, subsequent steps $A_j s$ in general do not follow deterministically. The reason is that each step depends on the both on the interaction with the society (external environment) and on the internal choices of each agent that are based on its previous knowledge and "experience".

The *underlying control* $\mathcal{U}$ can operate in two different ways: (i) $\mathcal{U}$ provides different parallel threads for the levels BA and MA or (ii) $\mathcal{U}$ provides an interleaving of control between the two layers.

In the former case, the MA level continuously monitors the BA. In the latter case instead, control must somehow pass between the two levels, e.g., 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, the MA will revise the BA's activities, which may imply constraints and condition verification.
– Vice versa, control will *shift down* from the MA to the BA by performing an act called *downward reflection*.

Forms of control based on reflection in computational logic are formally accounted for in [15]. 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 fulfils the above-proposed meta-model is presented in [1]. Dynamic changes that the MA level can operate on the BA level can be semantically modeled by means of the approach of Evolving Logic Programs (described in [16]). 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 ([17], [18],[19]) and DALI ([20], [21], [10]) agent-oriented languages.

2.3 Agent Model: specific aspects

To deal with dynamic verification of agent properties, both the BA and MA layers are supposed to include a component composed of a set of *constraints* (called SC): as we will see later, here we include all temporal constraints designed in order to induce or verify that certain actions or goals are performed in the correct order and in the allocated time. In addition, we consider constraints on the appropriate performance of actions that include what should happen and what should not happen.

In our setting, we assume that the new knowledge that an agent acquires from the society is recorded at the same time in two different ways: (i) together with meta-information that allows the agents to track the new knowledge and to store (an possibly later remove) it along with time-stamps and expectations; (ii) as plain knowledge added to the beliefs component.

The first way enables agents to reason about expectations and thus about goals that have not been accomplished yet. Therefore, the meta-information will help the agent "explore" the set of beliefs and to adequately update it (that is, remove/deactivate those beliefs that are deemed useless). Expectations, obligations and prohibitions have been coped with in logical agents by the approach (complementary to ours) of [22], where abducible predicates are used in the representation of obligations and prohibitions.

3 A-IMETATEM: Temporal Logic in the proposed framework

As already mentioned, the MA level is the one responsible for run-time monitoring of BA's activities over time. Therefore, in our perspective the MA will include rules inspired by temporal logic. Also the BA may include and take profit of this kind of rules: however, we mainly consider here meta-rules defined in the MA. Note that the MA is supposed to perform checks at *run-time* rather in advance like in model checking (which is however by no means excluded, but is not treated here). The basic aim of the checks is the detection of either fulfillment or violations of constraints that have to be worked out by some action of *improvement*. Those actions can not be decided "a priori" since they will depend on each specific context.

In 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 $p_0p_1...$, where: each vertex p_i in the path corresponds to a point in time (or "time instant" or "state"), p_0 is the present state and each p_i, with $i > 0$, are future states.

In order to model the dynamic behavior of agents, we propose an extension to the well-established METATEM logic called A-IMETATEM, an acronym standing for "Agent-Interval METATEM".

3.1 METATEM

In this subsection, we present the basic elements of propositional METATEM logic or PML ([9], [8]). The PML language is based both on the classical propositional logic enriched by temporal operators and on the direct execution of temporal logic statements.

First, we present the *syntax* of METATEM. The symbols used by this language 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 (note that $A_C \cap A_E = \emptyset$); (iii) an alphabet of propositional symbols A_P, obtained as the union of sets A_C and A_E ($A_P = A_C \cup A_E$); (iv) a set of propositional connectives such as **true**, **false**, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\Leftrightarrow$; (v) a set of temporal connectives; (vi) quantifiers, $\forall$ and $\exists$.

The set of temporal connectives is composed of a number of unary and binary connectives referring to future-time and past-time. Given a proposition $p \in A_p$ and the formulae φ and ψ, the syntax of connectives is given below. Note that if φ and ψ are formulae so is their combination.

Unary connectives referring to future time:

- $\bigcirc$ that is the "next state" symbol and $\bigcirc\varphi$ stands for: "the formula φ will be true at next state",
- $\Box$ that is the "always in future" symbol and $\Box\varphi$ means that the formula φ will always be true in the future,
- $\Diamond$ that is the "sometime in future" symbol and $\Diamond\varphi$ stands for: "there is a future state where the formula φ will be true".

Binary connectives referring to future time:

– $\mathcal{W}$ that is the “unless” (or “weak until”) symbol. The formula $\varphi\mathcal{W}\psi$ is true in a state s if the formula 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) (t excluded)
– $\mathcal{U}$ that is the “strong until” . The formula $\varphi\mathcal{U}\psi$ is true in a state s if the formula 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] (t included). In other worlds, from now on, φ remains true until ψ becomes true.

Unary connectives referring to past time:

– $\bullet$ is the “last state” operator and the formula $\bullet\varphi$ stands for “if there was a last state, then φ was true in that state”,
– $\blacklozenge$ is the “some time in past” operator and the formula $\blacklozenge\varphi$ means that formula φ was true in some past state,
– $\blacksquare$ is the “always in the past” and the formula $\blacksquare\varphi$ means that φ was true in all past states,
– $\odot$ is the strong last time operator, where $\odot\varphi \Leftrightarrow \neg \bullet \neg\varphi$

Note that the last state operator can determine the beginning of time by using the formula $\bullet$**false**.

Binary connectives referring to past time:

– $\mathcal{Z}$ is the “zince” (or “weak since”) operator. The formula $\varphi\mathcal{Z}\psi$ is true in a state s if the formula ψ is true in a state t (in the past of state s), and φ was true in every state of the time interval [t,s),
– $\mathcal{S}$ that is the “since” operator. The formula $\varphi\mathcal{Z}\psi$ is true in a state s if the formula ψ is true in a state t (in the past of state s), and φ was true in every state of the time interval [t,s]. That means that φ was true since ψ was true.

A METATEM program is a set of temporal logic rules in the form:

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

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.

The last part of this section is dedicated to the presentation of METATEM formulae semantics. For doing so, we first define the Model structures used in the interpretation of temporal formulae.

In the following, we consider σ to be a *state sequence* ($s_0 s_1$...), i the current time instant. We define a *structure* as a pair $(\sigma, i) \in (\mathbb{N} \rightarrow 2^{A_P})$ *x* $\mathbb{N}$ where A_P is the alphabet of propositional symbols. The relation $\models$ is the one giving the interpretation for temporal formulae in the given model structure. In general, a proposition $p \in A_P$ is true in a given model iff it is true in the current moment. As base case, we consider that formula **true** is true in any structure, while **false** is true in no model. Then we have:

Definition 5. Semantics *of temporal connectives is defined as follow:*

- $\sigma, i \vDash$ ***true***
- $\sigma, i \vDash \neg\varphi$ *iff* ***not*** $\sigma, i \vDash \varphi$
- $\sigma, i \vDash \varphi \wedge \psi$ *iff* $\sigma, i \vDash \varphi$ *and* $\sigma, i \vDash \psi$
- $\sigma, i \vDash \bigcirc\, \varphi$ *iff* $\sigma, i+1 \vDash \varphi$
- $\sigma, i \vDash \Box\, \varphi$ *iff* ***forall*** $k \in \mathbb{N}$ $\sigma, i+k \vDash \varphi$
- $\sigma, i \vDash \Diamond\, \varphi$ *iff* ***exists some*** $k \in \mathbb{N}$ $\sigma, i+k \vDash \varphi$
- $\sigma, i \vDash \varphi\, \mathcal{U}\, \psi$ *iff* ***exists some*** $k \in \mathbb{N}$ *such that* $\sigma, i+k \vDash \psi$ *and* ***forall*** $j \in 0..k-1$, $\sigma, i+j \vDash \varphi$
- $\sigma, i \vDash \varphi\, \mathcal{W}\, \psi$ *iff* $\sigma, i \vDash \varphi\, \mathcal{U}\, \psi$ *or* $\sigma, i \vDash \Box\, \varphi$
- $\sigma, i \vDash \bullet\varphi$ *iff* $i > 0$ *then* $\sigma, i-1 \vDash \varphi$
- $\sigma, i \vDash \blacksquare\, \varphi$ *iff* ***forall*** $k \in 1..i$ $\sigma, i-k \vDash \varphi$
- $\sigma, i \vDash \blacklozenge\, \varphi$ *iff* ***exist some*** $k \in 1..i$ *such that* $\sigma, i-k \vDash \varphi$
- $\sigma, i \vDash \varphi\, \mathcal{S}\, \psi$ *iff* ***exist some*** $k \in 1..i$ *such that* $\sigma, i-k \vDash \psi$ *and* ***forall*** $j \in 1..k-1$, $\sigma, i-j \vDash \varphi$
- $\sigma, i \vDash \varphi\, \mathcal{Z}$ *iff* $\sigma, i \vDash \varphi\, \mathcal{S}\, \psi$ *or* $\sigma, i \vDash \blacksquare\, \varphi$

3.2 A-IMETATEM

The purpose of this work 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 of the entire sequence. Sometimes it is not even desirable, since one needs properties to hold (or never to hold) within a certain time interval. This is why we propose an extension, called A-IMETATEM (acronym of "Agent-Interval METATEM"), to the METATEM logic.

Below are the future time interval operators of A-IMETATEM(where $m < n$).

- τ where the proposition $\tau(s_i)$ is true if s_i is the current state. I.e., we introduce the possibility of accessing the current state;
- $\bigcirc_m$, i.e., φ should be true at state s_{m+1};
- $\Diamond_m$ stands for "bounded eventually", i.e., $\Diamond_m\varphi$ means that φ eventually has to hold somewhere on the path from the current state to s_m;
- $\Diamond_{m,n}$ stands for "bounded eventually in a time interval", i.e., $\Diamond_{m,n}\varphi$ means that φ eventually has to hold somewhere on the path from state s_m to s_n;
- $\Box_{m,n}$ stands for "always in a given interval", i.e., $\Box_{m,n}\varphi$ means that φ should become true at most at state s_m and then hold at least until state s_n;
- $\Box_{\langle m,n\rangle}$ means that φ should become true just in s_m and then hold until state s_n, and not in s_{n+1}, where nothing is said for the remaining states;
- N stands for "never", i.e., $N\varphi$ means that φ should not become true in any future state;
- $N_{m,n}$ stands for "bounded never", i.e. $N_{m,n}\varphi$ means that φ should not be true in any state between s_m and s_n, included.

The past time interval operators instead are:

– $\blacklozenge_m$, i.e., given the current state s_i then φ was at state s_m, with $m < i$;
– $\blacksquare_{m,n}$ is the "always in past" operator where, given the current state s_i and $m \leq n \leq i$ then φ was true in the entire time interval m, n. I.e., $\blacksquare_{m,n}\varphi$ means that if φ was true at state s_m and then it remained true at least until state s_n;
– $\blacksquare_{\langle m,n \rangle}$ is the strict version of $\blacksquare_{m,n}$, where φ was true only in the time interval m, n. I.e., $\blacksquare_{\langle m,n \rangle}$ means that φ became true just in state s_m and then remained true exactly until state s_n

After having introduced the syntax of A-IMETATEM, we present the semantics of A-IMETATEM formulae.

Definition 6. (Semantics of A-IMETATEM formulae) *Let σ be a* state sequence $s_0s_1...$, *i the* current moment *in time, and φ, ψ* METATEM-formulae. *The* semantics *of A-IMETATEM is defined as:*

– *all basic METATEM operators are defined as in Definition 5*
– *we define $\sigma \models \tau(s_0)$ (for $i = 0$), where $s_0 \equiv \blacklozenge$**false**, in order to track down the initial state (or time) of the σ-sequence;*
– *$\sigma \models \bigcirc_m\varphi$ iff $\sigma, m + 1 \models \varphi$;*
– *$\sigma \models \Diamond_m\varphi$ iff **exist some** $j, j \leq m$: $\sigma, j \models \varphi$;*
– *$\sigma \models \Box_{m,n}\varphi$ iff **forall** $m \leq j \leq n$: $\sigma, j \models \varphi$;*
– *$\sigma \models \Box_{\langle m,n \rangle}\varphi$ iff **forall** $j, m \leq j \leq n$: $\sigma, j \models \varphi$* and ***forall** r: $r < m$: $\sigma, r \models \neg\varphi$* and *$\sigma, n + 1 \models \neg\varphi$;*
– *$\sigma \models N\varphi$ iff **forall** $j, j \geq 0$: $\sigma, j \models \neg\Diamond\varphi$;*
– *$\sigma \models \blacklozenge_m\varphi$ iff for $m < i$, where i is the current state: $\sigma, m \models \varphi$;*
– *$\sigma \models \blacksquare_{m,n}\varphi$ iff **forall** $j, m \leq j \leq n \leq i$: $\sigma, j \models \varphi$;*
– *$\sigma \models \blacksquare_{\langle m,n \rangle}\varphi$ iff **forall** $m \leq j \leq n \leq i$: then $\sigma, j \models \varphi$* and ***forall** r: $r < m$ then $\sigma, r \models \neg\varphi$* and *$\sigma, n + 1 \models \neg\varphi$*

Based on previous definition of A-IMETATEM semantics, we propose a run-time control of goals/plans performed by agents during their evolution and learning process. We remark that verification of properties is not meant to occur at every state but, rather, with a frequency associated to each property. In such a way, a crucial property for agent evolution can be tested more often than a less relevant one. For doing so, we need a further extension to define subsequences and refine the semantics accordingly.

We consider σ to be an infinite sequence of states $s_0, s_1, ...$ of a system and σ^k to be the subsequence $s_0, s_{k_1}, s_{k_2}, ...$ where for each k_r ($r \geq 1$), k_r mod $k = 0$, i.e., $k_r = g \times k$ for some g. Note that we have $\sigma^1 = \sigma$, $\sigma^2 = s_0, s_2, s_4, ...$ and so on. All operators introduced above can be redefined for subsequences.

Definition 7. *Let O_p be any of the operators introduced in A-IMETATEM and $k \in \mathbb{N}$ with $k > 1$. Then O_{p^k} is an operator whose semantics is a variation of the semantics of O_p where the sequence σ_s is replaced by the subsequence ${\sigma_s}^k$.*

4 A-IMETATEM 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 can not be re-executed whenever the agent learns a new fact/rule/action. 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.

In fact, we will add to the underlying logic programming agent-oriented language the possibility of specifying rules including A-IMETATEM operators. These rules will be attempted at a certain frequency, and whenever verified may determine suitable modifications to the program itself. In the rest of this section, we first define the syntax of A-IMETATEM operators in the context of logic programs, and introduce some useful notation; next, we define A-IMETATEM basic rules, A-IMETATEM contextual rules, and A-IMETATEM rules with improvements. Along with the explanation we provide some examples.

In our framework, we consider A-IMETATEM rules to be applied upon universally quantified formulae. Note that the negation operator (*not*) is interpreted in our setting as "negation-as-failure". For introducing A-IMETATEM rules in logic-programming based languages, we first have to represent the A-IMETATEM operators within this kind of languages. This representation is shown in Figure 1.

In the following, we omit the operator arguments when implied from the context, and in these cases we write OP instead of $OP(m, n; k)$. We often omit frequency: we assume in fact that there exists a default frequency, defined in the component $\mathcal{CI}$ including control information (cf. Section 2). Also, as a special case, when we do not care about the starting point of an interval, we introduce the special constant $start$ where $OP(start, n; k)$ means that OP 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 $NOW(t)$ holds.

In addition to the basic operators, we introduce here two useful derived operators. The first one is related to the issue, that often occurs in practice, of defining a "normal" occurrence of an event, such as, e.g., a reaction to a certain external stimulus or the triggering of an internal process of an agent trying to achieve a goal. We say that an event's occurrence is "normal" when it occurs sufficiently often. For performing this type of control, it is necessary to define a new operator of A-IMETATEM, called $USUALLY$.

Definition 8. *Given a sentence φ and a related frequency $f_\varphi \in \mathbb{N}$ we define the operator $USUALLY$ as follows: $USUALLY(M, N)\,\varphi \equiv ALWAYS(M, N; f_\varphi)\,\varphi$.*
The shortcoming $USUALLY\,\varphi$ stands for $ALWAYS(start, now; f_\varphi)\,\varphi$.

A-IMETATEM Op^k	**OP(m,n;k)**
$\tau(t)$	$NOW(t)$
$\bigcirc^k$	$NEXT(1;k)$
$\bigcirc_j{}^k$	$NEXT(j;k)$
$\Diamond^k$	$FINALLY(1;k)$
$\Diamond^k_m$	$FINALLY(m;k)$
$\Box^k$	$ALWAYS(1;k)$
$\Box_{m,n}{}^k$	$ALWAYS(m,n;k)$
$\Box_{\langle m,n\rangle}{}^k$	$ALWAYS\_2(m,n;k)$
N^k	$NEVER(1;k)$
$N_{m,n}{}^k$	$NEVER(m,n;k)$
$\bullet^k$	$LAST(1;k)$
$\bullet^k_m$	$LAST(j;k)$
$\blacksquare^k$	$P\_ALWAYS(1;k)$
$\blacksquare^k_{m,n}$	$P\_ALWAYS(m,n;k)$
$\blacksquare^k_{\langle m,n\rangle}$	$P\_ALWAYS\_2(m,n;k)$

Fig. 1. Representation of A-IMETATEM operators

-

According to this definition, the new operator $USUALLY$ is a shortcoming for a check with frequency f_φ, where $ALWAYS$ has the usual role of checking the proposition in all given states. Therefore, $USUALLY\,\varphi$ holds as far as this periodical check is successful.

The second derived operator holds if the given event has occurred at some point of an interval, one or more times:

Definition 9. *Given a sentence φ we define the operator $SOMETIMES$ as follows:* $SOMETIMES(M,N)\,\varphi \equiv \neg ALWAYS(M,N)\,\varphi \wedge \neg NEVER(M,N)\,\varphi$.

For each A-IMETATEM operator, we define its negated counterpart.

Definition 10. *Given an A-IMETATEM formula $OP(m,n;k)$, we define $N-OP(m,n;k)$ standing for $\neg OP(m,n;k)$.*

4.1 A-IMETATEM basic rules

We can now define the form of a basic A-IMETATEM rule.

Definition 11. *An A-IMETATEM rule ρ is a writing of the form $\alpha\ :\ \beta$ or simply β where we have the following. β is a conjunction including logic programming literals and/or (possibly negated) A-IMETATEM operators; α (if specified) is an atom of the form $p(t_1,\ldots,t_n)$ which is called the rule* representative.

Once attempted, an A-IMETATEM rule is verified (or *succeeds*, or *holds*) whenever all its conjuncts succeed (which implies that all the A-IMETATEM operators hold and all the logic programming literals are provable). In the case of A-IMETATEM operators (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, A-IMETATEM rules will be periodically attempted (we will also say "checked"). Whenever we should have a conjunction including A-IMETATEM operators with different frequencies, it is up to the implementation to choose one: here, we assume a random choice. We also assume a default frequency whenever not explicitly defined. As a first example of an A-IMETATEM rule, we are going to comment the following:

$NEVER(goal(g),\ deadline(g,t),$
$\quad NOW(T1),\ T1 \leq\ t,$
$\quad not\ achieved(g),\ dropped(g)$

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 $T1$, which is however instantiated by the predefined operator NOW. Below we generalize to the non-ground case.

4.2 A-IMETATEM 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 12. *Let* ρ *be an* A-IMETATEM *rule.*
The corresponding contextual A-IMETATEM rule *is a rule of the form* $\rho :: \chi$ *where:*

- χ *is called the* evaluation context *of the rule, and consists of a conjunction of logic programming literals;*
- *every variable occurring in* ρ *must occur in an atom (non-negated literal) of the context* χ*.*

From Definition 12 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 A-IMETATEM rule*, we propose the following example:

$FINALLY\ (N;F)\ achieved(G)\ ::$
$\qquad goal(G), priority(G,P), timeout(P,N), frequency(P,F)$

In this rule, the goal G is established by the context, which also contains the atoms $priority$, $timeout$ and $frequency$ (that we assume to be suitably defined in the agent's knowledge base). Informally, the rule requires that a goal with timeout N (set according to the priority P 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 have actually been achieved in time, or it fails because the timeout will have elapsed.

4.3 A-IMETATEM rules with improvement

Whenever an instance of an A-IMETATEM 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 *improvement*. Program modification/evolution is accounted for by the EVOLP semantics [2], [16].

In order to make the improvement possible (either immediately or later), we record successful instances of A-IMETATEM rules. In fact, according to the semantic approach of [1] which encompasses lemma assertion, the representative α of a successful rule is recorded in the form $\alpha P\ : t$ where postfix P stands for "past", and t is the time-stamp of the record (which can be omitted if not useful, but is needed to distinguish among different "versions" of the same record).

We now extend the definition of contextual A-IMETATEM rules to specify a corresponding *improvement* action, that can be a repair or other according to the situation at hand.

Definition 13. *An* A-IMETATEM *rule with a improvement is a rule of the form:*
$\rho :: \chi \div \psi$*, or* $\alpha P \div \psi$ *where:*

- $\rho :: \chi$ *is a contextual* A-IMETATEM *rule;*
- αP *is the recorded representative of a contextual* A-IMETATEM *rule;*
- ψ *is called the* improvement action *of the rule, and it consists of an atom* ψ*.*

the left-hand-side is called the monitoring condition *of the rule, its negation is called the* check condition *of the rule.*

If the monitoring condition of an *A-IMETATEM* holds when the rule is checked (or, symmetrically, the check condition is violated), then the improvement action ψ 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 present level of commitment of the agent to its objectives has to be increased. This can be specified as:

$N - NEVER\,(not\ achieved(G), dropped(G)\,) ::$
$\qquad (goal(G), deadline(G, T), NOW(T1), T1 \leq T) \div\ inc\_comt(T1)$

$incr\_comt(T)\ \leftarrow\ level(commitment, L),$
$\qquad increase\_level(L, L1),$

$assert(neg(commitment\_mod(L))$,
$assert(commitment\_mod(L1))$

Suppose that at a certain time t the check condition

$NEVER\,(not\ achieved(G), dropped(G))$

is violated for some specific goal g, i.e, its negation $N - NEVER$ holds. Upon detection of the violation, the system will attempt the improvement (in this case a repair) 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 repair action will determine the update of the current agent program $\mathcal{P}_i$, returning a new agent program $\mathcal{P}_{i+1}$. The A-IMETATEM 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., [23]). 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:

$ALWAYS(T1; now)\ ::\ goal(G), not\ achieved(G), dropped(G, T3),$
$now > T3, confidence(G, now) > confidence(G, T3)\ \div\ re - exec(G).$

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

$FINALLY(start, T) :: NOW(T), goal(G), times\_exec(G) > K \div disable(G)$

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.

$SOMETIMES(start, T) ::$
$NOW(T), goal(G), executed(G), consequence(G, C), not\ desired(C) \div$
$assert(NEVER(T, T1)exec(G) :: NOW(T), threshold(T1))$

A-IMETATEM operators can be used to check the past behavior and knowledge of the agent but also to influence its future behavior. The agent evolution in fact entails also an evolution of recorded information, which in turn may affect the evaluation of social factors such as trust, confidence, etc. 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 representatives. I.e., each agent which will have passed the test will have its trust level increased as soon as the rule with repair is executed.

$Rel\_Ag(Ag)\ :\ ALWAYS(m,n;k)\ reliable(Ag)$
$Rel\_Ag(A)P\ \div\ increase\_trust\_level(A)$

5 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 operators 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. Then, we do not make any claim on its performance and complexity. However, experiments performed in DALI show that a timely verification of properties at a suitable frequency not only does not worsen, but even improves, the agent performance (w.r.t. the same checks be integrated in the base-level agent program). 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 3229. Springer-Verlag, Berlin (2006)
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. Volume 430 of Lecture Notes in Computer Science., 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. Volume 3862 of Lecture Notes in Computer Science., 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. Fisher, M., Ghidini, C.: The abc of rational agent modelling. In: AAMAS ’02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM (2002) 849–856
14. Rao, A.S., Georgeff, M.P.: Modeling agents within a BDI-architecture. In Fikes, R., Sandewall, E., eds.: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR), Cambridge, Massachusetts, Morgan Kaufmann (1991)
15. 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
16. 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
17. 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
18. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of agency. In: Proc. ECAI-2004. (2004)
19. 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
20. Tocchio, A.: Multi-Agent systems in computational logic. PhD thesis, Dipartimento di Informatica, Università degli Studi di L’Aquila (2005)
21. 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)
22. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P., Sartor, G.: Mapping deontic operators to abductive expectations. Computational & Matematical Organization Theory **12**(2-3) (October 2006) 205–225
23. 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.