Ensuring Trustworthy and Ethical Behavior in Intelligent Logical Agents*

Stefania Costantini

Università degli Studi di L'Aquila
Dipartimento di Ingegneria e Scienze dell'Informazione, e Matematica
Via Vetoio snc, Loc. Coppito, I-67010 L'Aquila - Italy
{stefania.costantini}@univaq.it

Abstract. Autonomous Intelligent Agents are employed in many important autonomous applications upon which the life and welfare of living beings and vital social functions may depend. Therefore, agents should be *trustworthy*. A-priori certification techniques can be useful, but are not sufficient for agents that evolve, and thus modify their epistemic and belief state. In this paper we propose/refine/extend techniques for run-time assurance, based upon introspective self-monitoring and checking. The aim is to build a 'toolkit' to allow an agent designer/developer to ensure trustworthy and ethical behavior.

1 Introduction

The development of methods for implementing Intelligent Agents so as to ensure transparent, explainable, reliable and ethical behavior is due to the fact that agent systems are being widely adopted for many important autonomous applications upon which the life and welfare of living beings and vital social functions may depend. Therefore, agents should be trustworthy in the sense that they could be relied upon to do what is expected of them, while not exhibiting unwanted behavior. Equally importantly, they should *not* behave in improper/unethical ways given the present context, and they should not devise their own new objectives in contrast with the user's interests. They should also be transparent, in the sense of being able to explain their actions and choices when required. Without that, it would be hard for human users to trust such systems. Agents should also report to their users in case the interaction with the environment leads them to indentify new objectives to pursue, as such objectives might not be in line to user's interests. We restrict ourselves to agent systems based upon computational logic, because they provide transparency and explainability 'by design', as logical rules can easily be transposed into natural-language explanations. Logic-based languages and architectures are discussed in the survey [1]. They are based (more or less directly) on the so-called BDI ('Belief, Desires, Intention') model of agency [2].

Most pre-deployment (or 'static' or 'a priori') verification methods for logical agents rely upon model-checking (cf. [3] and the references therein), and some (e.g., [4]) upon theorem proving. These techniques are able to certify 'a priori' that agents

* Copyright ©2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-2710/paper12.pdf

fulfill certain requisites of trustworthiness, that means that they *do* what is expected from them, and *do not violate* certain rules of behavior. However, they can be sufficient for agents that keep their epistemic state constant during their operation, and interact with the environment in a predefined way; so, they are not sufficient for agents that will revise their beliefs and objectives in consequence of the interaction with a changing not-always-predictable environment.

The motivation of the work presented in the present paper is in fact that an agent's behavior is in general affected by its interaction with the external world, i.e., by which events perceived by the agent and in which order. In many practical cases, the actual arrival order of events and the set of possible events is so large that computing all combinations would result in a combinatorial explosion, thus making 'a priori' verification techniques too heavy to apply and therefore unpractical. In agents that learn, it is not even possible to predict the set of events that will be observed and considered by an agent, that might devise new objective,s not necessarily in line with the expected agent's behavior. We believe in therefore that, in changing circumstances, agents should be able to observe and if necessary modify their own behavior, i.e. they should *reflect* on themselves. The methods that we propose are not alternative but rather complementary to a-priori existing verification and testing methodologies.

We find similarities between our approach and the point of view of *Self-aware computing*: quoting [5], *Self-aware and self-expressive computing describes an emerging paradigm for systems and applications that proactively gather information; maintain knowledge about their own internal states and environments; and then use this knowledge to reason about behaviors, revise self-imposed goals, and self-adapt.... Systems that gather unpredictable input data while responding and self-adapting in uncertain environments are transforming our relationship with and use of computers.* Reporting from [6], *From an autonomous agent view, a self-aware system must have sensors, effectors, memory (including representation of state), conflict detection and handling, reasoning, learning, goal setting, and an explicit awareness of any assumptions. The system should be reactive, deliberative, and* **reflective**.

In this paper we propose new contributions to an envisaged toolkit for run-time self-assurance of evolving agents. We specify techniques and tools for: (i) checking the immediate, "instinctive" reactive behavior in a context-dependent way, and (ii) checking and re-organizing an agent's operation at a more global level. In particular, we introduce meta-rules and meta-constraints for agents' run-time self-checking (i.e., checking occurs at run time at a certain –customizable– frequency), that can be exploited to ensure respect of machine ethics principles. The proposed meta-constraints are based upon a simple interval temporal logic (defined in previous work) particularly tailored to the agent realm, that we called A-ILTL ('Agent-Oriented Interval LTL', LTL standing as customary for 'Linear Temporal Logic'). As A-ILTL constraints and evolutionary expressions are defined over formulas of an underlying logic language $\mathcal{L}$, one contribution of this paper is to make A-ILTL independent of $\mathcal{L}$.

In A-ILTL, properties can be defined that should hold in specific time instants and time intervals, according to past and future events. In fact, *Evolutionary A-ILTL Expressions* are a novel feature that we introduce in this paper, and that we have implemented and experimented. They are based upon specifying: (i) a sequence of events that are

supposed to have happened; represented via a notation obtained from regular expressions: one does not need to completely specify a finite sequence, but is allowed to define partially specified and even infinite sequences; (ii) a temporal-logic-like expression defining a property that should hold (in given interval); (iii) a sequence of events that are supposed to happen in the future, without affecting the property; (iv) optionally, "repair" countermeasure to be undertaken if the property is violated. Counter measures can be at the object-level, i.e., they can be related to the application, or at the meta-level, e.g., they can even result in replacing a software component by a diverse alternative. The act of checking temporal expressions can indeed be considered as an introspective act, as an agent suspends its current activities in order to envision and possibly self-modify its own state. We do not aim to continuously monitor the entire system's state, but rather to monitor only the activities that a designer deems to be relevant for keeping the system's behavior within a desired range.

Our work has a clear connection to the work of [7], which proposes to implement the "restraining bolt"[1] by conditioning reinforcement learning of reactive actions to obey LTL specifications defining known ethical principles. This (very promising) method is orthogonal to ours, because our checking is performed at run-time in order on the one hand to enact ethical rules not hard-wired but learned by the agent, and on the other hand to check the agent's overall BDI behavior (desires and intentions/plans should not be unethical).

A toolkit for logical agents' run-time self-assurance can be obtained by means of the synergy between the new features proposed here with those introduced in past work (notably [9, 10], [12], and [8])[2]. The proposed approach can be seen under two perspectives. On the one hand, as a means to define an enhanced "restraining bolt" capable to prevent agents to enact unwanted behavior unpredictable at design time, as it was learned later on (tipically via reinforcement learning). On the other hand, since "our" agents can be also able to learn rules of behavior over time, we can have a "disobeying robot" that can on occasion disallow behavior hardwired at design time, because in the present agent's context such behavior violates context-dependent learned ethical rules.

The paper is organized as follows: in Section 2 we introduce metarules for checking reactive behavior, and in Section 3 we introduce A-ILTL constraints in theory and in practice. In Section 4 we briefly discuss related work and then conclude. In the examples, we adopt a Prolog-like syntax of the form $Head\ :-Body$, where $Head$ is an atom, $Body$ is a conjunction of literals (atoms or negated atoms, that are often called 'subgoals') and the comma stands for $\wedge$. The specific syntax that we use is however purely aimed at illustration: the approach can be, 'mutatis mutandis', re-worked w.r.t. a different syntax.

[1] A "restraining bolt" as imagined in the Star Wars Science Fiction saga is a small cylindrical device that, when activated, restricts a droid's actions to a set of behaviors considered desirable/acceptable by its owners.

[2] The author acknowledges the co-authors of the aforementioned papers, that contributed to various extents to the development of this research.

2 Checking Agents' Reactive Behavior

In the BDI model, an agent will have objectives, and devise plan to reach these objectives. However, most agent-oriented languages and framework also provide mechanisms for 'pure' reactivity, i.e. 'instinctive' reaction to an event. The possible ethically acceptable reactions that an agent can enact are in general strictly dependent on the context, the agent's role and the situation. Assume for sake of example an agent (either human or artificial) which finds itself to face some other agent which might be, or certainly is, a criminal:

- if the context is that of playing a videogame, every kind of reaction is allowed, including beating, shooting or killing the 'enemy', except when small children are watching;
- if the context is a role game then the players can pretend to threaten, shout or kill the other players, where every action is simulated and thus harmless;
- in reality, a citizen can shout, call the police, and try to defend itself; a policemen can threathen, arrest, or in extreme cases shoot the suspect criminal.

The reaction to enact in each situation can be 'hardwired' by the agent's designer, or it can be learned, e.g., via reinforcement learning. While in the former case a-priori verification can suffice, in the latter case run-time checking of agent's behavior w.r.t. ethical rules is in order, as the results of learning are in general unpredictable and to some extent potentially unreliable. Even the method of conditioning reinforcement learning to obey some properties proposed in [7] may not suffice, as it can hardly consider contexts and roles.

We introduce a mechanism to verify and possibly enforce desired properties by means of metalevel rules. To define such new rules, we assume to augment the underlying language $\mathcal{L}$ at hand with a naming (or "reification") mechanism, and with the introduction of two distinguished predicates, $solve$ and $solve_not$. These are metapredicates, that can be employed to control the object-level behavior. In fact, $solve$, applied to (the name of) an atom which represents an action of an objective of an agent, may specify conditions for that action/goal to be enacted; vice-versa $solve_not$ specifies under which conditions they should be blocked.

Below is a simple example of the use of $solve$ to specify that execution of an action Act is allowed in the present agent's context of operation C and role R only if such action is allowed, and it is deemed to be ethical w.r.t. context and role. Any kind of reasoning can be performed via metalevel rules in order to monitor and assess base-level ethical behavior Below, lowercase syntactic elements such as p', c' are names of predicates and constants, and uppercase syntactic elements such as V' are metavariables (naming mechanisms have been widely studied, cf., among many, [13–15], where meta-level mechanisms such as those that we employ here are studied).

$$\begin{aligned}
&solve(execute_action'(Act')) :- \\
&\qquad present_context(C), agent_role(R), \\
&\qquad allowed(C, R, Act'), ethical(C, R, Act').
\end{aligned}$$

We assume that $solve(execute_action'(Act'))$ is automatically invoked whenever subgoal (atom) $execute_action(Act)$ is attempted at the object level. More generally,

given any subgoal A at the base level, if there exists an applicable $solve$ rule such rule is automatically applied, and A can succeed only if $solve(\uparrow A)$ succeeds, where $\uparrow A$ is the name of A according to the chosen naming mechanism. We assume, also, that the present context and the agent's role are kept in the agent's knowledge base. Since both parameters may change, the same action may be allowed in some circumstances and not in others.

Symmetrically we can define metarules to forbid unwanted object-level behavior, e.g.:

$$solve_not(execute_action'(Act')) :- \\ present_context(C), ethical_exception(C, Act').$$

this rule prevents successful execution of its argument, in the example $execute_action(Act)$, whenever $solve_not(\uparrow A)$ succeeds. Then, action/goal A can succeed (according to its base-level definition) only if $solve(\uparrow A)$ (if defined) succeeds and $solve_not(\uparrow A)$ (if defined) does not succeed.

The outlined functioning corresponds to *upward reflection* when the current subgoal A is reified and applicable $solve$ and $solve_not$ metarules are searched; if applicable metarules are found, control in fact shifts from base level to metalevel (as $solve$ and $solve_not$ can rely upon a set of auxiliary metalevel rules). If no rule is found or whenever $solve$ and $solve_not$ metarules complete their execution, *downward reflection* returns control to the base level, to execution of A if confirmed or to subsequent goals/actions if A has been canceled by either failure of the applicable $solve$ metarule or success of the applicable $solve_not$ metarule.

Via $solve$ and $solve_not$ metarules, fine-grained activities of an agent can be punctually checked and thus allowed and disallowed, according to the context an agent is presently involved into with a certain role.

Semantics of the proposed approach can be sketched as follows (a full semantic definition can be found in [16]). According to [17], in general terms we understand a semantics SEM for logic knowledge representation languages/formalisms as a function which associates a theory/program with a set of sets of atoms, which constitute the intended meaning. When saying that P is a program, we mean that it is a program/theory in the (here unspecified) logic languages/formalism that one wishes to consider.

We introduce the following restriction on sets of atoms I that should be considered for the application of SEM. First, as customary we only consider sets of atoms I composed of atoms occurring in the ground version of P. The ground version of program P is obtained by substituting in all possible ways variables occurring in P by constants also occurring in P. In our case, metavariables occurring in an atom must be substituted by metaconstants, with the following obvious restrictions: a metavariable occurring in the predicate position must be substituted by a metaconstant denoting a predicate; a metavariable occurring in the function position must be substituted by a metaconstant denoting a function; a metavariable occurring in the position corresponding to a constant must be substituted by a metaconstant denoting a constant. According to well-established terminology, we therefore require $I \subseteq B_P$, where B_P is the *Herbrand Base* of P. Then, we pose some more substantial requirements: we restrict SEM

to determine acceptable sets of atoms only[3] where I is an *acceptable* set of atoms iff I satisfies the axiom schemata:

$$A \leftarrow solve(\uparrow A) \quad \neg A \leftarrow solve_not(\uparrow A)$$

So, we obtain the implementation of properties that have been defined via $solve$ and $solve_not$ rules without modifications to SEM for any formalism at hand. For clarity however, one can assume to filter away $solve$ and $solve_not$ atoms from acceptable sets. In fact, the *Base version* I^B of an acceptable set I can be obtained by omitting from I all atoms of the form $solve(\uparrow A)$ and $solve_not(\uparrow A)$. Procedural semantics and the specific naming relation that one intends to use remain to be defined, whereas the above-introduced semantics is independent of the naming mechanism. For approaches based upon (variants of) Resolution (like, e.g., Prolog and like many agent-oriented languages such as, e.g., AgentSpeak [18], GOAL [19], 3APL [20] and DALI [21, 22]) one can extend their proof procedure so as to automatically invoke metarules whenever applicable, to validate or invalidate success of A.

How to define the predicate $ethical(C, R, Act')$? Again, rules defining this predicate can be specified at design time, or they can be learned, or a combination of both options. In previous works [23–25], a hybrid logic-based approach was proposed for ethical evaluation of agents' behavior, with reference to dialogue agents (so-called 'chatbots') but easily extendable to other kinds of agents and of applications. The approach is based on logic programming as a knowledge representation and reasoning language, and on Inductive Logic Programming (ILP) for learning rules needed for ethical evaluation and reasoning, taking as a starting point general ethical guidelines related to a context; the learning phase starts from a set of annotated cases, but the system is then able to perform continuous incremental learning.

3 Checking Agent's Behavior over Time

The techniques illustrated in the previous section are "punctual", in the sense that they provide context-based mechanisms to allow/disallow agents' actions. However, it is necessary to introduce ways to monitor an agent's behavior in a more extensive way. In fact, properties that one wants to verify often depend upon which events have been observed by an agent up to a certain point, and which others are supposed to occur later. The definition of frameworks such as the one that we propose here, for checking agent's operation during its 'life' based on its experience and expectations, has not been widely treated so far in the literature.

Below we propose in fact a method for checking the agent's behavior during the agent's activity, based on maintaining information on its past behavior. In particular, we assume an agent to record what has happened in the past (events perceived, conclusions reached and actions performed). These records (that we call *past events*) encode relevant aspects of an agent's *experience*: possibly augmented by time stamps, they form the (subjective) *history* of the agent's activity. The set of past events evolves in time, and can be managed and kept up-to-date. Past events constitute the ground upon which an

[3] modulo bijection: i.e., SEM can be allowed to produce sets of atoms which are in one-to-one correspondence with acceptable sets of atoms

agent can check its own behavior with respect to what has happened in the external environment (more precisely, with respect to what the agent has perceived about what has happened). To perform such checks, we introduce a new kind of constraints, that we call *Evolutionary Expressions*, that define properties that should hold and countermeasures that should be taken if they are violated.

3.1 A-ILTL

For defining properties that are supposed to be respected by an evolving system, a well-established approach is that of Temporal Logic, and in particular of Linear-time Temporal Logics (LTL). These logics evaluate each formula with respect to a vertex-labeled infinite path (or "state sequence") $s_0 s_1 \ldots$ where each vertex s_i in the path corresponds to a point in time (or "time instant" or "state"). In what follows, we use the standard notation for the best-known LTL operators.

In [10] we formally introduced an extension to LTL based on *intervals*, called A-ILTL for 'Agent-Oriented Interval LTL'. A-ILTL is useful because the underlying discrete linear model of time and the complexity of the logic remains unchanged with respect to LTL. This simple formulation can be efficiently implemented, and is sufficient for expressing and checking a number of interesting properties of agent systems. Formal syntax and semantics of A-ILTL operators (also called below "Interval Operators") are fully defined in [10]. Some among the A-ILTL operators are the following, where φ is an expression in an underlying agent-oriented language $\mathcal{L}$, and m, n are positive integer numbers used to (optionally) specify the interval where the formula must hold; if the interval is not specified, then the meaning is the same as for LTL.

$F_{m,n}$ (*eventually (or "finally") in time interval*). $F_{m,n}\varphi$ states that φ has to hold sometime on the path from state s_m to state 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. Can be customized into G_m, *bounded always*, where φ should become true at most at state s_m.

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

In practical use, as seen below A-ILTL operators will allow one to construct useful run-time constraints.

3.2 A-ILTL and Evolutionary Semantics

In this section, we refine A-ILTL so as to operate on a sequence of states that corresponds to the Evolutionary Semantics [26]. This is a meta-semantic approach, as it is independent of the underlying agent-oriented logic languages/formalism $\mathcal{L}$. It assumes that, during agent's execution, the agent can evolve: at each evolution step i the agent's program (that initially will be P_0) may change (e.g., by learning and via interaction with other agents), with a transformation of P_i into P_{i+1}, thus producing a Program Evolution Sequence $PE = [P_0, \ldots, P_n, \ldots]$. The program evolution sequence will imply a corresponding Semantic Evolution Sequence $ME = [M_0, \ldots, M_n, \ldots]$ where

M_i is the semantic account of P_i at step i according to the semantics of $\mathcal{L}$. In parallel, the agent' *history* H, including agent's *memories* as recorded by the agent itself, will evolve as well (for a recent formal approach to memory management in logical agents cf. [27, 28]) . The Evolutionary Semantics ε^{Ag} of Ag is thus the tuple $\langle H, PE, ME \rangle$, with $n = \infty$ (i.e., over a potentially infinite evolution). The *snaphot at stage i*, indicated with ε_i^{Ag}, is the tuple $\langle H_i, P_i, M_i \rangle$

Notice that, states in our case are not simply intended as time instants. Rather, they correspond to stages of the agent evolution. Time in this setting is considered to be local to the agent, where with some sort of "internal clock" is able to time-stamp events and state changes. We borrow from [29] the following definition of *timed state sequence*, that we tailor to our setting.

Definition 1. *Let σ be a (finite or infinite) sequence of states, where the ith state e_i, $e_i \geq 0$, is the* semantic snaphots at stage i ε_i^{Ag} *of given agent Ag. Let T be a corresponding sequence of time instants t_i, $t_i \geq 0$. A* timed state sequence for agent Ag *is the couple $\rho_{Ag} = (\sigma, T)$. Let ρ_i be the i-th state, $i \geq 0$, where $\rho_i = \langle e_i, t_i \rangle = \langle \varepsilon_i^{Ag}, t_i \rangle$.*

We in particular consider timed state sequences which are *monotonic*, i.e., if $t_{i+1} > t_i$ then $e_{i+1} \neq e_i$. In fact, there is no point in semantically considering a static situation: as mentioned, a transition from e_i to e_{i+1} will in fact occur when something happens, externally or internally, that affects the agent.

Then, in the above definition of A-ILTL operators, it is immediate to let $s_i = \rho_i$ (with a refinement, cf. [10], to make states correspond to time instants).

We need to adapt the interpretation function $\mathcal{I}$ of LTL to our setting. In fact, we intend to employ A-ILTL within agent-oriented languages, where we restrict ourselves to logic-based languages for which an evolutionary semantics and a notion of logical consequence can be defined. Thus, given agent-oriented language $\mathcal{L}$ at hand, the set Σ of propositional letters used to define an A-ILTL semantic framework will coincide with all ground expressions of $\mathcal{L}$ (an expression is *ground* if it contains no variables, and each expression of $\mathcal{L}$ has a possibly infinite number of ground versions). A given agent program can be taken as standing for its (possibly infinite) ground version, as it is customarily done in many approaches. Notice that we have to distinguish between logical consequence in $\mathcal{L}$, that we indicate as $\models_{\mathcal{L}}$, from logical consequence in A-ILTL, indicated above simply as $\models$. However, the correspondence between the two notions can be quite simply stated by specifying that in each state s_i the propositional letters implied by the interpretation function $\mathcal{I}$ correspond to the logical consequences of agent program P_i:

Definition 2. *Let $\mathcal{L}$ be a logic language. Let $Expr_{\mathcal{L}}$ be the set of ground expressions that can be built from the alphabet of $\mathcal{L}$. Let ρ_{Ag} be a timed state sequence for agent Ag, and let $\rho_i = \langle \varepsilon_i^{Ag}, t_i \rangle$ be the ith state, with $\varepsilon_i^{Ag} = \langle H_i, P_i, M_i \rangle$. An A-ILTL formula τ is defined over sequence ρ_{Ag} if in its interpretation structure $\mathcal{M} = \langle \mathbb{N}, \mathcal{I} \rangle$, index $i \in \mathbb{N}$ refers to ρ_i, which means that $\Sigma = Expr_{\mathcal{L}}$ and $\mathcal{I} : \mathbb{N} \mapsto 2^{\Sigma}$ is defined such that, given $p \in \Sigma$, $p \in \mathcal{I}(i)$ iff $P_i \models_{\mathcal{L}} p$. Such an interpretation structure will be indicated with $\mathcal{M}^{Ag}$. We will thus be consequently able to state whether τ holds/does not hold w.r.t. ρ_{Ag}.*

A-ILTL properties will be verified at run-time, and thus they can act as *constraints* over the agent behavior[4]. In an implementation, verification may not occur at every state (of the given interval). Rather, sometimes properties need to be verified with a certain frequency, that can be specific for each property. To model a frequency k, we have introduced a further extension that consists in defining subsequences of the sequence of all states: if Op is any of the operators introduced in A-ILTL and $k > 1$, Op^k is a semantic variation of Op where the sequence of states ρ_{Ag} of given agent is replaced by the subsequence $s_0, s_{k_1}, s_{k_2}, \ldots$ where for each $k_r, r \geq 1$, $k_r \ mod \ k \ = 0$, i.e., $k_r = g \times k$ for some $g \geq 1$.

A-ILTL formulas to be associated to an agent to establish the properties it has to fulfill can be defined within the agent program, though they constitute an additional separate layer. Agent evolution can be considered to be "satisfactory", or "coherent", if it obeys all these properties. An "ideal" agent will have a coherent evolution. Instead, violations will occasionally occur, and actions should be undertaken so as to attempt to regain coherence for the future.

3.3 A-ILTL in Practice

In this section, we adopt the *pragmatic* syntax that we have adopted in DALI, where an A-ILTL formula is represented as $OP(m, n; k)\,\varphi$. Integers m, n define the time interval where (or since when, if n is omitted) expression $OP\,\varphi$ is required to hold, and k (optional) is the frequency for checking whether the expression actually holds. E.g., $EVENTUALLY(m, n; k)\,\varphi$ states that φ should become true at some point between time instants m and n. In rule-based logic programming languages like DALI, we restrict φ to be a conjunction of literals. In pragmatic A-ILTL formulas, φ must be ground when the formula is checked. Variables occurring in an A-ILTL formula must be instantiated prior to check via a *context* χ: so, we have *contextual A-ILTL formulas* of the form $OP(m, n; k)\,\varphi :: \chi$. For the evaluation of φ and χ, we rely upon the procedural semantics of the 'host' language.

In the following, a contextual A-ILTL formula τ will implicitly stand for the ground A-ILTL formula obtained via evaluating the context. In [10] we have specified how to *operationally* check whether such a formula holds. The following formulation deals with monitoring a condition and performing, if necessary, suitable corrections to agent's belief state or behavior.

Definition 3. *An* evolutionary *A-ILTL rule (or "constraint") is of the form (where M, N, K can be either variables or constants)*

$\epsilon \ : \ OP(M, N; K)\,\varphi \ :: \ \chi \ ::: \delta \ :::: \ \kappa \ \div \ \rho$

where: (i) ϵ is an (optional) event sequence, to be satisfied by agent's history for the rule to be checked; (ii) $OP(M, N; K)\,\varphi :: \chi$ is a contextual A-ILTL formula, called the monitoring condition*, that may involve the evaluation of observed events, and possibly of agent's internal conditions; (iii) δ is a sequence of events (optional) that are expected to happen in the future; (iv) is a sequence of events (optional) that are expected* not *to*

[4] By abuse of notation we will indifferently talk about A-ILTL rules, expressions, or constraints.

happen in the future; (v) ρ (optional) is the active part *of the rule, as it specifies the actions of repair/improvement to be performed on the agent's state and functioning.*

All past events are assumed to be stored in a ground form. All variables occurring in evolutionary A-ILTL expressions are implicitly universally quantified, in the style of Prolog-like logic languages. To define partially known sequences of any length, we admit for event sequences the syntax of regular expressions so as to specify irrelevant/unknown events, and repetitions. Whenever the monitoring condition (automatically checked at frequency K) is violated (i.e., it does not hold in present agent's state), then the active part ρ is executed.

A sample evolutionary A-ILTL rule is the following (where N stands for operator "never"):

$$supply_P{}^{+}(r, _s)\ :\ N(quantity(r, V), V < th)\ :::\\ consume_A{}^{+}(r, Q)$$

The expression specifies that, after a supply of resource r for a certain unknown quantity ($supply_P{}^{+}(r, _s)$ stands for a sequence of of supply actions occurred in the past, each one with a certain quantity $_s$, recorded as past events, postfix P), the agent is expected to consume quantities of the resource itself (postfix A indicates actions). The expression states in particular a constraint requiring, in the τ part, that the available quantity of resource r must remain over a certain threshold th. Such expression is supposed to be verified at run-time whenever new events are perceived. A violation may occur if in some state the A-ILTL formula τ does not hold, i.e., in the example, if the available quantity V of resource r runs too low. This constraint is very significant from an ethical point of view, where in fact a very common unethical behavior concerns the improper use/waste of limited resources, think for instance to the excessive and/or improper use of environmental resources.

The variation listed below states that no more consumption can take place if the available quantity of resource r is scarce; thus, in this case, the a repair measure is specified.

$$supply_P{}^{+}(r, s)\ :\ N(quantity(r, V), V < th)\ :::\\ consume_A{}^{+}(r, Q)\ \div block(consume_A(r, Q))$$

We might instead opt for another (softer) formulation, that forces the agent to limit consumption to small quantities (say less than quantity q) if the threshold is approaching (say that the remaining quantity is $th + f$, for some f).

$$supply_P{}^{+}(r, s)\ :\ N(quantity(r, V), V < th + f)\ :::\\ consume_A{}^{+}(r, Q)\ \div allow(consume_A(r, Q), Q < q)$$

The above example demonstrates that the proposed approach to dynamic verification is indeed needed: one can easily realize that static verification is not suitable for the verification task at hand. In fact, one cannot in general provide a static checker with any possible input configuration, i.e., any sequence of performed 'supply' and 'consume' actions, as the number of potential configurations is not limited. In alternative, one might provide total supply and consumption figures: however, one would just

draw the quite pleonastic conclusion that the desired property holds whenever supply is sufficiently generous and consumption prudentially limited. Instead, in the proposed approach we are able to verify the target property "on the fly", whatever the sequence of performed actions and the involved quantities. Moreover, we are also able to try to repair an unwanted situation and regain a satisfactory state of affairs.

Below we provide another example that, though simple, is in our opinion significant as it is representative of many others. Namely, we assume that an agent manages a FIFO queue, thus accepting operations of pushing and popping elements on/from the queue. The example thus models in an abstract way the very general and widely used architecture where an agent provides a service to a number of 'consumers'. We establish the constraint, represented below in our formalism, that the queue must not contain any duplicated elements e_1 and e_2. From an ethical point of view, this means that customers cannot reiterate a request of service if their previous one have not been processed. This in order to ensure fairness in the satisfaction of different customers' requests. The possible agent's actions are: $push_A(Req, Q)$, that inserts a generic value Req in the queue, representing (in some format) a request of service from another agent (each inserted element is given an index e_i); $pop_A(e, R)$, that extracts the oldest element from the queue, i.e., the request to be presently satisfied. The constraint considers an unknown number of pushing actions happened in the past (and thus are now recorded as past events) and can foresee an unknown number of future popping actions.

$$push_P{}^{+}(Req, Q) : \quad N(in_queue(e_1, R1), in_queue(e_2, R1)) \ ::: \ pop_A{}^{+}(eRQ)$$

3.4 Experiments

We implemented and we have been experimenting the proposed approach within the DALI multi-agent system [30], by exploiting the *internal event* feature that allows automated execution at a (predefined or user-defined) frequency. In this section we present basic experiments, aimed at establishing the effectiveness of the approach not w.r.t. competitors, that basically do not exist, but w.r.t. a correspondent solution developed in pure Prolog[5]. DALI is in fact an agent-oriented extension to Prolog whose interpreter is implemented in Prolog itself. The experiments have been performed on the queue constraint illustrated above. We did not consider the frequency of constraint-checking, that could not be implemented in an acceptably simple way in Prolog.

The instance size (number of elements to push and pop on the queue) can be established by the user when starting a test session. The items to pop/push are, in the experiments, randomly-generated numbers. In Figure 1 and Figure 2 we show the execution time of the two solutions at the growth of the instance size. In Figure 3 we show the difference in percentage between the execution times.

All figures refer to a dataset of up to 500 elements to push and pop. This has been sufficient to identify an initial "unstable" stage and then a trend that further consolidates with the growth of the instance size.

[5] We thank former Ph.D. student Dr. Alessio Paolucci who has practically performed the experiments.

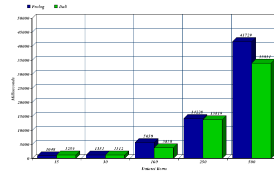

Fig. 1. x axis: instance size; y axis: execution time, blue bars pure Prolog green bars DALI

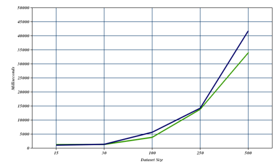

Fig. 2. Interpolation average values, blue line pure Prolog green line DALI

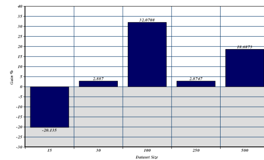

Fig. 3. x axis: instance size; y axis: gain (in percentage) when using DALI

What we can see is that, when the number of events that we consider is small, then the two solutions are more or less equivalent, the Prolog one a bit better as it involves no overhead (while the DALI implementation itself and the DALI events management necessarily involves some). But, as soon as the instance size grows, the DALI solution becomes better and better in a very significant way, despite the overhead of the DALI implementation[6]. We can thus conclude that the new constructs that we propose are not only expressive and then useful for expressing problem features in a compact and declarative way, but they also improve efficiency and thus effectiveness of solutions.

4 Related Work and Concluding Remarks

In this paper we have extended past work so as to devise a toolkit for run-time self-checking of logic-based agents. The proposed toolkit is able to detect and correct behavioral anomalies by using special meta-rules, and via dynamic constraints that are also able to consider partially specified sequences of events that happened, or that are expected to happen. The experiments, performed in the DALI language, have shown that the proposed approach is computationally affordable.

There are relationships between our approach and event-calculus formulations [31], e.g., the ones presented in [32] and [33] where however the temporal aspects and the correction of violations are not present. Deontic logic has been used for building well-behaved agents (c.f., e.g., [34]). However, expressive deontic logics are undecidable[7]. Therefore, although our approach cannot compete in expressivity with deontic logics, it can be usefully exploited in practical applications.

Future work includes making self-checking constraints adaptable to changing conditions, and devising a useful integration and synergy with declarative a-priori verification techniques.

References

1. Bordini, R.H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A.E., Gómez-Sanz, J.J., Leite, J., O'Hare, G.M.P., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica (Slovenia) **30**(1) (2006) 33–44
2. Rao, A.S., Georgeff, M.: Modeling rational agents within a BDI-architecture. In: Proc. of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Morgan Kaufmann (1991) 473–484
3. Kouvaros, P., Lomuscio, A.: Verifying fault-tolerance in parameterised multi-agent systems. In Sierra, C., ed.: Proceedings of the Twenty-Sixth Intl. Joint Conference on Artificial Intelligence, IJCAI2017. ijcai.org (2017) 288–294
4. Shapiro, S., Lespérance, Y., Levesque, H.: The cognitive agents specification language and verification environment In: AAMAS '02: First International Joint conference on Autonomous agents and multiagent systems, Proceedings (2002) 19–26

[6] The reader wishing to reproduce this experiment can refer to us to obtain the the code to run.

[7] We acknowledge Dr. Abeer Dyoub for the thorough investigation of the applications of deontic logic to build ethical agents during the development of her Ph.D. Thesis [35].

5. Tørresen, J., Plessl, C., Yao, X.: Self-aware and self-expressive systems. IEEE Computer **48**(7) (2015) 18–20
6. Amir, E., Andreson, M.L., Chaudri, V.K.: Report on DARPA workshop on self aware computer systems. Technical Report, SRI International Menlo Park United States (2007) Full Text : `http://www.dtic.mil/dtic/tr/fulltext/u2/1002393.pdf`.
7. De Giacomo, G., Iocchi, L., Favorito, M., Patrizi, F.: Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In: Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, AAAI Press (2019) 128–136
8. Costantini, S., Gasperis, G.D., Dyoub, A., Pitoni, V.: Trustworthiness and safety for intelligent ethical logical agents via interval temporal logic and runtime self-checking. In: 2018 AAAI Spring Symposia, Stanford University, AAAI Press (2018)
9. Costantini, S., Dell'Acqua, P., Pereira, L.M., Tocchio, A.: Ensuring agent properties under arbitrary sequences of incoming events. In: Informal Proceedings of 17th RCRA Intl. Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion (2010)
10. Costantini, S.: Self-checking logical agents. In: Proceedings of LA-NMR 2012. Volume 911, CEUR Workshop Proceedings (CEUR-WS.org), Invited paper (2012)
11. Costantini, S., Dyoub, A., Pitoni, V.: Reflection and introspection for humanized intelligent agents. In: Proceedings of the fourth Workshop on Bridging the Gap between Human and Automated Reasoning, co-located with the 27th International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence (IJCAI-ECAI 2018). Also appeared in: Proceedings of the 33rd Italian Conference on Computational Logic CILC2018, CEUR Workshop Proceedings, volume 2214, CEUR-WS.org, (2018)
12. Costantini, S.: Self-checking logical agents. In: AAMAS'13, International Conference on Autonomous Agents and Multi-Agent Systems, Proceedings. IFAAMAS (2013) 1329–1330
13. Perlis, D., Subrahmanian, V.S.: Meta-languages, reflection principles, and self-reference. In: Handbook of Logic in Artificial Intelligence and Logic Programming, Volume2, Deduction Methodologies. Oxford University Press (1994) 323–358
14. Barklund, J., Dell'Acqua, P., Costantini, S., Lanzarone, G.A.: Semantical properties of encodings in logic programming. In Lloyd, J.W., ed.: Logic Programming, Proceedings of the 1995 International Symposium. MIT Press (1995) 288–302
15. Costantini, S.: Meta-reasoning: a Survey. In: Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A.Kowalski, Part II. Lecture Notes in Computer Science 2408. Springer (2002) 253–288
16. Costantini, S., Formisano, A.: Augmenting knowledge representation and reasoning languages with customizable metalogic features. In: Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019. Volume 2396 of CEUR Workshop Proceedings., CEUR-WS.org (2019) 14–29
17. Dix, J.: A classification theory of semantics of normal logic programs: I. Strong properties. Fundam. Inform. **22**(3) (1995) 227–255
18. Rao, A.S.: Agentspeak(l): BDI agents speak out in a logical computable language. In: Agents Breaking Away, 7th European Works. on Modelling Autonomous Agents in a Multi-Agent World, Proceedings. Lecture Notes in Computer Science 1038. Springer (1996) 42–55
19. Hindriks, K.V.: Programming rational agents in goal. In: Multi-Agent Programming. Springer US (2009) 119–157
20. Dastani, M., van Riemsdijk, M.B., Meyer, J.J.C.: Programming multi-agent systems in 3APL In: Multi-Agent Programming: Languages, Platforms and Applications. Multiagent Systems, Artificial Societies, and Simulated Organizations, Volume 15. Springer (2005) 39–67

21. Costantini, S., Tocchio, A.: A logic programming language for multi-agent systems. In: Logics in Artificial Intelligence, European Conference, JELIA 2002, Proceedings. Lecture Notes in Computer Science 2424. Springer (2002)
22. Costantini, S., Tocchio, A.: The DALI logic programming agent-oriented language. In: Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Proceedings. Volume 3229 of Lecture Notes in Computer Science. Springer (2004) 685–688
23. Dyoub, A., Costantini, S., Lisi, F.A.: Learning Answer Set Programming Rules for Ethical Machines. In: Proceedings of the Thirty Fourth Italian Conference on Computational Logic CILC2019, CEUR-WS.org, Volume 2396 (2019) 300–315
24. Dyoub, A., Costantini, S., Lisi, F.A.: Towards an ILP Application in Machine Ethics. In: Proceedings of the 29th International Conference on Inductive Logic Programming - ILP2019, Lecture Notes in Computer Science 11770. Springer (2019) 26–35
25. Dyoub, A., Costantini, S., Lisi, F.A.: Towards ethical machines via logic programming. In: Proceedings 35th International Conference on Logic Programming (Technical Communications), ICLP 2019 Technical Communications. Volume 306 of EPTCS, (2019) 333–339
26. Costantini, S., Tocchio, A.: About declarative semantics of logic-based agent languages. In: Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Selected and Revised Papers. Lecture Notes in Computer Science 3904. Springer (2006) 106–123
27. Costantini, S., Pitoni, V.: Reasoning about memory management in resource-bounded agents. In: Proceedings of the 34th Italian Conference on Computational Logic. Volume 2396 of CEUR Workshop Proceedings, CEUR-WS.org (2019) 217–228
28. Pitoni, V., Costantini, S.: A temporal module for logical frameworks. In: Proceedings 35th International Conference on Logic Programming (Technical Communications), ICLP 2019 Technical Communications. Volume 306 of EPTCS, (2019) 340–346
29. Henzinger, T.A., Manna, Z., Pnueli, A.: Timed transition systems. In: Real-Time: Theory in Practice, REX Works., Lecture Notes in Computer Science 600 Springer (1992) 226–251
30. Costantini, S., De Gasperis, G., Pitoni, V., Salutari, A.: DALI: A multi agent system framework for the web, cognitive robotic and complex event processing. In: Joint Proceedings of the 18th Italian Conference on Theoretical Computer Science and the 32nd Italian Conference on Computational Logic co-located with the 2017 IEEE International Workshop on Measurements and Networking (2017 IEEE M&N). Volume 1949 of CEUR Workshop Proceedings., CEUR-WS.org (2017) 286–300 Download at `https://github.com/AAAI-DISIM-UnivAQ/DALI`.
31. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing **4** (1986) 67–95
32. Berreby, F., Bourgne, G., Ganascia, J.: A declarative modular framework for representing and applying ethical principles. In: AAMAS'17, 16th Conference on Autonomous Agents and MultiAgent Systems, Proceedings. ACM Press (2017) 96–104
33. Tufis, M., Ganascia, J.: A normative extension for the BDI agent model. In: Proceedings of the 17th INternational Conference on Climbing an Walking Robots and the Support Technologies for Mobile Machines. (2014) 691–702
34. Bringsjord, S., Arkoudas, K., Bello, P.: Toward a general logicist methodology for engineering ethically correct robots. IEEE Intelligent Systems **21**(4) (2006) 38–44
35. Dyoub, A.: Multi-Agent systems in Comp. logic. PhD thesis, Department of Information Engineering, Computer Science and Mathematics, Supervisor Prof. Stefania Costantini. University of L'Aquila, (2019)