Ensuring Agent Properties under Arbitrary Sequences of Incoming Events

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

[1] Dip. di Informatica, Università di L'Aquila, Coppito 67010, L'Aquila, Italy
stefania.costantini@univaq.it

[2] Dept. of Science and Technology - ITN, Linköping University, Norrköping, Sweden
pierangelo.dellacqua@itn.liu.se

[3] Centro de Inteligência Artificial (CENTRIA), Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract

This paper deals with run-time detection and possible correction of erroneous and/or anomalous behavior in agents. We augment our previous approaches by allowing an agent to explicitly observe and record its past behavior so as to be able to decide its best actions, and avoid errors performed in previous similar situations.

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.

In previous work, we defined semantic frameworks for agent approaches based on computational logic that account for: (i) the kind of evolution of reactive and proactive agents due to directly dealing with stimuli, that are to be coped with, recorded and possibly removed; and (ii) the kind of evolution related to adding/removing rules from the agent knowledge base. These frameworks have been integrated into an overall framework for logical evolving agents (cf. [7, 2]), where every agent is seen as the composition of a base-level (or object-level) agent program and one or more meta-levels. In this model, updates related to recoding stimuli are performed in a standard way, while updates involving the addition/deletion of sets of rules, related to learning, belief revision, etc. are a consequence of meta-level decisions. As agent systems are more widely used in real-world applications, the issue of verification is becoming increasingly important (see [9] and the many references therein).

The motivation of the work presented in the present paper is that the agent behavior is affected by its interaction with the external world, i.e., by events perceived by the agent and in which order. In most practical cases however, the actual

Proceedings of the 17th International RCRA workshop (RCRA 2010):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Bologna, Italy, June 10–11, 2010

arrival order of events is unforeseeable, 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 actually unpractical. Moreover, 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. Therefore, we augment our previous approaches by allowing an agent to explicitly observe and record its past behavior so as to be able to decide the best actions to do, and to avoid errors performed in previous similar situations. This motivates the importance of recording the most relevant facts which happened in the past and of recovering error and behavioral anomalies by means of appropriate strategies.

The definition of frameworks such as the one that we propose here, for checking agent behavior during its life based on experience, has not been really treated up to now. In fact, there has been an increasing quest for agent platforms whose component entities would be capable of exhibiting a correct and rigorous behavior with respect to expectations. However, developers have mostly applied model-checking techniques that are based upon abstract models of an agent system, thus neglecting the run-time verification of behavior during the agent life according to what happens in actual evolution of the system. On the one hand, due to the combinatorial explosion, properties that can be statically verified are necessarily quite simple. On the other hand, there is no way to reinstate a correct behavior at run-time, in case unwanted situations should occur.

In this paper, we propose a method for checking the agent behavior correctness during the agent activity, based on maintaining information on its past behavior. This information is useful in that it records what has happened in the past to the agent (events perceived, conclusions reached and actions performed) and thus encodes relevant aspects of an agent's *experience*. If augmented by time-stamps, these records (that we call *past events*) constitute in a way the *history* of the agent activity. The set of past events evolves in time, and can be managed for instance by distinguishing the most recent versions of each past event, that contribute to the agent present perception of the state of the world.

Past events can be exploited for the purpose of self-checking agent activities: we propose in fact the introduction of specific constraints, defined as temporal-logic-like formulae expressed upon past events and events that are supposed to occur in the future. Alberti et al. in [1] have adopted a similar approach based on social constraints in order to model the interactions among (possibly heterogeneous) agents that form an open society.

2 Defining agent experience

A rule-based agent consists of a knowledge base and of rules aimed at providing the entity with rational, reactive, pro-active and communication capabilities. The knowledge base constitutes a part of the agent's "memory", where rules define the agent's behavior. Through "memory", the agent is potentially able to learn from experiences and ground what it knows on these experiences [10]. The interaction between the agent and the environment can play an important role in constructing its memory and may affect its future behavior. Most methods to design agent memorization mechanisms have been inspired by models of human memory as for instance [11], [12]. Memory, experience and knowledge are in general strongly related. Correlation between these elements can be obtained, e.g., via neural networks, via mathematical models or via logical deduction.

Some of the authors of this paper have proposed in [5],[6] a method of correlating agent experience and knowledge by using a particular construct, the internal events, that has been introduced in the DALI language (though it can be in principle adopted in any computational logic setting). We have defined the "static" agent memory in a very simple way as composed of the original knowledge base augmented with *past events* that record the external stimuli perceived, the internal conclusions reached and the actions performed.

Past events can play a role in reaching internal conclusions. These conclusions, which are proactively pursued, take the role of "dynamic" memory that supports decision-making and actions: in fact, the agent can inspect its own state and its view of the present state of the world, so as to identify the better behavioral strategy in that moment. More specifically, *past events*, in our approach, record external events that have happened, internal events that have been raised and actions that have been performed by the agent. Each past event is time-stamped to also record when the event has happened. Past events have at least two relevant roles: describe the agent experience; keep track of the state of the world and of its changes, possibly due to the agent intervention. With time, on the one hand past events can be overridden by more recent ones of the same kind (take for example temperature measurement: the last one is the "current" one) and on the other hand can also be overridden also by more recent ones of different kinds, which are somehow related.

In this paper, we extend and refine the concepts that we had introduced in the above-mentioned previous work. In particular, we introduce a set P of current "valid" past events that describe the state of the world as perceived by the agent. We also introduce a set PNV where we store all previous ones. Thus, the history H referred to in the definition of the evolutionary semantics is the tuple $\langle P, PNV \rangle$. Given history H, we introduce the notation $H.P$ and $H.PNV$ to refer to the two components. In practice, H is dynamically augmented with new events that happen. Let $\mathcal{E} = (E \cup I \cup A)$ be the set of the events that may happen, in which as already observed we include the sets of external (set E) and internal (set I) events and the actions (set A) that the agent itself performs. Each event in $X \in \mathcal{E}$ may occur none or several times in the agent's life. Each occurrence is therefore indicated as $X : T_i$ where T_i is a time-stamp indicating when this specific occurrence has happened (where the time-stamp can be omitted if irrelevant. Each $X \in \mathcal{E}$ is a ground term, with the customary prolog-like syntax. If one is interested in identifying which kind of event is X, a postfix (that can be omitted if irrelevant) can provide this indication. I.e., X_E is an external event, X_A is an action and X_I an internal event. As soon as X is perceived by the agent, it is recorded in P in the form ${X_P}^Y : T_i$ where P is a postfix that syntactically indicates past events and Y is a label indicating what is X, i.e., if it belongs to E, I or A. By abuse of notation for the sake of conciseness we will often omit label Y if the specific kind of event is irrelevant, and we will sometimes indicate ${X_P}^Y : T_i$ as X_i or simply X.

Clearly, as new "versions" of an event arrive, they should somehow "override" the old versions that have to be transferred into PNV: for instance, P will contain the most recent measure of the outside temperature, while previous measurements will be recorded in PNV. Past events in PNV may still have a relevant role for the entity decision process. In fact, an agent could be interested for instance in knowing how often an action has been performed or a particular stimuli has been received by the environment, or the first and last occurrences, etc. In the previous example, measurements recorded in PNV might for instance be used for computing the average temperature in a certain period of time. Clearly, PNV will have a limited size and thus older or less relevant events will have to be canceled. We do not cope with this issue in this paper, where instead we will cope with the issue of how to keep P up-to-date. Consider for example to have an agent that opens or respectively closes some kind of access. The action of opening the access can be performed only if the access is closed, and vice versa for closing. Assume that this very simple agent believes that no external interference may occur, and thus the access is considered (by the agent) to be closed if the agent remembers to have closed it, and vice versa it is considered to be open if the agent remembers to have opened it. These "memories", in our approaches, are past events in P. Therefore, the agent will have previously closed the door (and thus it considers itself enabled to open it) if a past event such as $close_P^A : t_1$ is in P. After performing the action $open_A : t_2$, not only the past event $open_P^A : t_2$ must be inserted into P, but for avoiding possible mistakes the previous past event $close_P^A : t_1$ should be removed from P and transferred into PNV. *Past Constraints* define which past events must be eliminated and under which conditions. They should be automatically applied in order to keep the agent memory consistent with the external world. More formally, we define a *Past Constraint* as follows (where we overlook the label Y indicating the kind of past event).

Definition 2.1 (Past Constraint)**.** *A Past Constraint has syntax:*

$$X_{kP} : T_k, ..., X_{mP} : T_m \trianglelefteq X_{sP} : T_s, ..., X_{zP} : T_z, \{C_1, ..., C_n\}$$

where $X_{kP} : T_k, ..., X_{mP} : T_m$ are the past events which are no longer valid whenever past events $X_{sP} : T_s, ..., X_{zP} : T_z$ become known and conditions $C_1, ..., C_n$ are true, i.e., as we will say, whenever the constraint holds.

For the previous example, we would have the following past constraint.

$$close_P^A : t_1 \trianglelefteq open_P^A : t_2, t_1 < t_2$$

We define $H \star X$ as the operation of adding the past-event version of event $X \in \mathcal{E}$ to the history, that also implies transferring past events from P to PNV according to the past constraints.

Definition 2.2. *Let PC be the set of past constraints and S a set of past events. By $F = PC(S)$ we indicate the result of the application of the past constraints in PC, i.e., F included the left-hand sides of all the constraints in PC which hold given as known the past events in S.*

Definition 2.3. *Given history $H = \langle P, PNV \rangle$, set of past constraints PC and event X, the result of $H \star X$ is an updated history $H' = \langle P', PNV' \rangle$ where: (i) $P' = S \setminus F$ with $S = H.P \cup \{X_P\}$ and $F = PC(S)$; (i) $PNV' = H.PNV \cup F$,* .

3 Checking the behavior of Evolving Agents

According to the evolutionary semantics defined in [4], time instants $s_0 s_1 \ldots$ concerning an agent's "life" can be understood in terms of the events that happen. In fact, at the i-th evolution step we have an history H_i, an agent program P_i and its intended semantics M_i, determined by events $E_1, \ldots, E_i$ occurred so far. The next evolution step will take place in accordance to the perception of next event E_{i+1}. Then, any property φ which holds w.r.t. ε_i^{Ag}, i.e. w.r.t. the agent evolutionary semantics up to step i, will keep holding until next event will determine a transition to the next snapshot. In other words, the agent understands the world only in terms of the event that it perceives. Therefore we can state the following.

Definition 3.1. *Given agent Ag with evolutionary semantics ε^{Ag}, we let s_i = ε_i^{Ag} = $\langle H_i, P_i, M_i \rangle$.*

I.e., a state is taken to be the snapshot at stage i of the evolutionary semantics of the agent.

In model-checking, the aim is to establish if some LTL formula (where LTL stands for Linear-Time Temporal Logic: for a survey, the reader can refer to [8]) $Op\,\varphi$ or $\varphi\, Op\, \psi$ (where Op can be, for instance, N for "never", E for "eventually", etc.) can be established to be true, given a description of the system at hand from which the system evolution can be somehow elicited. In order to cope with the many cases where this evolution cannot be fully foreseen, we propose a reformulation of temporal logic operators so as to take into account the events that have happened already and those that are expected to happen in the future and to be relevant to the property that we intend to check. We do so because indeed checking a property w.r.t. any possible sequence of events would determine a combinatorial explosion of the checks that should be made. Moreover, many of the checks would be useless, as they would concern combination of events that are irrelevant to the property at hand.

Definition 3.2 (Evolutionary LTL Expressions)**.** *Let τ be a temporal logic expression of the form $Op\,\varphi$ if operator Op is unary or $\varphi\, Op\, \psi$ if operator Op is binary. The evolutionary version of τ, that we will call* Evolutionary LTL Expression*, is of the form*

$$\{E_{P_1}, \ldots, E_{P_{n-1}}\}\, \tau\, :\, \{F_1, \ldots, F_m\}$$

where: $n, m \geq 0$*;* $\{E_{P_1}, \ldots, E_{P_{n-1}}\} \subseteq H_n.P$ *denote the relevant events which are supposed to have happened;* $s_n = \varepsilon_n^{Ag}$ *is the state from which the property is required to be checked;* $\{F_1, \ldots, F_m\}$ *denote the events that are expected to happen in the future; if* $k-1$ *is the state in which* F_m *will happen,* $s_{n+k} = \varepsilon_{n+k}^{Ag}$ *is the state until which* τ *is required to be checked.*

We may notice that we might adapt for this case the enhanced temporal logic operators that we have discussed above, i.e., in τ, we might adopt $Op_{n,n+k}$ instead of Op, except that in general we do not know k, i.e., we cannot foresee at which state the last expected relevant event F_m will happen. We may also notice that in many practical cases we are unable to provide a full sequence of the expected events, and sometimes we will be interested only in some of them. Thus, in the above definition, to be able to indicate the sets of past and future events in a more flexible way we admit the syntax of regular expressions (see, e.g., $http://en.wikipedia.org/wiki/Regular\_expression$ and the references therein). We also extend this syntax as follows.

Definition 3.3. *Let* X *be a wild-card standing for any event. The expression* $X^+(Y_1^{v1}, \ldots, Y_m^{v_m})$*, where* $m > 0$ *and for each of the* v_i*'s, either* $v_i > 0$ *or* v_i *= '*$+$*', stands for a non-empty sequence of* X*'s in which each event* Y_i *occurs* v_i *times, and in particular any number of times if* v_i *= '*$+$*'.*

Moreover, in Definition 3.2 we do not require the E_{P_i}'s and the F_i's to be ground terms. Instead, we admit each of them to contain variables if we are not interested in precisely specifying some of their parameters. For instance, the expression $X^+(consume_A{}^+(r, Q))$ indicates a sequence of events where the action of consuming (some resource r) occurs at least once. Each action will refer to a quantity Q which is not specified. An evolutionary LTL expression could be for instance (where N stands for the LTL operator "never"):

$$X^+(supply_A(r, s))\, N(quantity(r, V), V < th)\, X^+(consume_A{}^+(r, Q))$$

stating that, after having provided a supply of resource r for a total quantity s, the agent is expected to consume unknown quantities of the resource itself. Nevertheless, the expression states a constraint requiring that the available quantity of resource r remains over a certain threshold th. Evolutionary LTL expressions are in fact supposed to act as constraints to be verified at run-time whenever new events are perceived. At any state between s_i and s_{n+k} a violation may occur if the inner LTL formula τ does not hold of that state. The proposition below formally allows for dynamic run-time checking of evolutionary LTL expressions. In fact, it says that if a given expression holds in a certain state and is supposed to keep holding after the first expected event has happened, then checking this expression amounts to checking the modified expression where the occurred event has become a past event, and subsequent events are still expected.

Proposition 1. *Given expression* $\mathcal{F} = \{E_{P_1}, \ldots, E_{P_n}\}\,\tau\,:\,\{F_1, \ldots, F_m\}$*, assume that* $\mathcal{F}$ *holds at state* s_n *and that* τ *still holds after the occurrence of event* F_1*. Given* $\mathcal{F}_{F_1} = \{E_{P_1}, \ldots, E_{P_n}, F_{P1}\}\,\tau\,:\,\{F_2, \ldots, F_m\}$ *we have* $\mathcal{F} \equiv \mathcal{F}_{F_1}$*.*

In prior work (see e.g., [2, 3]), we introduced temporal logic rules with *improvement*, where actions could be specified in order to cope with unwanted situations. We extend this approach to the present work. As discussed above, we consider evolutionary LTL expressions as constraints that can hold or not at any state. We enrich these constraints by means of the specification of which actions to perform in order to try regain a suitable state of affairs. For lack of space, we illustrate our proposal by means of the following example. The evolutionary LTL expression with improvement listed below states that no more consumption can take place if the available quantity of resource r is scarce (thus, in this case, the improvement it is rather a repair).

$$X^{+}(\mathit{supply}_A(r,s))\,N(\mathit{quantity}(r,V), V < th)\,X^{+}(\mathit{consume}_A{}^{+}(r,Q)):$$

$$\mathit{prevent}(\mathit{consume}_A(r,Q))$$

We assume the distinguished predicate $\mathit{prevent}$ to be implicitly added to the preconditions of every action, that can take place only if not prevented. We might as well add another constraint, that forces the agent to limit consumption to small quantities (say $th1$) if it is approaching the threshold (say that the remaining quantity is $th + s$, for some s). Again, the distinguished predicate allow should be a precondition of every action, that should be performed only if not prevented and allowed.

$$X^{+}(\mathit{supply}_A(r,s))\,N(\mathit{quantity}(r,V), V < th + s)\,X^{+}(\mathit{consume}_A{}^{+}(r,Q)):$$

$$\mathit{allow}(\mathit{consume}_A(r,Q), Q < th1)$$

4 Concluding Remarks

In this paper, we have presented an approach to update agent memory and to detect and correct behavioral anomalies by using dynamic constraints, based on introducing particular events, past events, that record what has happened. The runtime observation of actual anomalous behavior with dynamic possible correction of detected problems, as opposed to full prior classical program verification and validation on all inputs, can be a key to bringing down the well-known computational complexity of the agent behavior assurance problem.

References

[1] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, and G. Sartor. An abductive interpretation for open agent societies. In *Proceedings of the 8th National Congress on Artificial Intelligence, AI*IA 2003*, number 2829 in LNAI, pages 287–299. Springer-Verlag, 2003.

[2] S. Costantini, P. Dell'Acqua, and L. M. Pereira. A multi-layer framework for evolving and learning agents. In A. R. M. T. Cox, editor, *Proceedings of Metareasoning: Thinking about thinking workshop at AAAI 2008, Chicago, USA*, 2008.

[3] S. Costantini, P. Dell'Acqua, L. M. Pereira, and P. Tsintza. Runtime verification of agent properties. In *Proc. of the Int. Conf. on Applications of Declarative Programming and Knowledge Management (INAP09)*, 2009.

[4] S. Costantini and A. Tocchio. About declarative semantics of logic-based agent languages. In M. Baldoni and P. Torroni, editors, *Declarative Agent Languages and Technologies*, LNAI 3904, pages 106–123.

[5] S. Costantini and A. Tocchio. 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.

[6] S. Costantini and A. Tocchio. 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.

[7] S. Costantini, A. Tocchio, F. Toni, and P. Tsintza. 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.

[8] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B.* MIT Press, 1990.

[9] M. Fisher, R. H. Bordini, B. Hirsch, and P. Torroni. Computational logics and agents: a road map of current technologies and future trends. *Computational Intelligence Journal*, 23(1):61–91, 2007.

[10] J. S. Gero and W. Peng. Understanding behaviors of a constructive memory agent: A markov chain analysis. *Know.-Based Syst.*, 22(8):610–621, 2009.

[11] R. H. Logie. *Visuo-Spatial Working Memory*. Psychology Press, Essays in Cognitive Psychology, 1994.

[12] D. Pearson and R. H. Logie. Effect of stimulus modality and working memory load on menthal synthesis performance. *Imagination, Cognition and Personality*, 23(2-3):183–191, 2004.