Meta-level Constraints for Complex Event Processing in Logical Agents

Stefania Costantini and Giovanni De Gasperis
Dip. di Ingegneria e Scienze dell'Informazione (DISIM), Università di L'Aquila,
Coppito 67100, L'Aquila, Italy
stefania.costantini,giovanni.degasperis@univaq.it

Abstract

"Brittleness" can be intended as the propensity of an agent to perform poorly or fail in the face of circumstances not explicitly considered by the agent's designer. In opposition to brittleness, "Versatility" and "Perturbation-tolerance" express the ability of being able to cope with previously unanticipated situations. These properties are particularly important in many practical applications which have the need to actively monitor vast quantities of event data to make automated decisions and take time-critical actions, i.e., perform "Complex Event-Processing". In this paper, we propose temporal meta-level constraints to be dynamically checked, that allow an agent to timely self-check and adapt its behavior, and in particular its reactive modalities.

Introduction

A well-known and particularly difficult problem in AI is the so-called "brittleness" problem: automated systems tend to "break" when confronted with even slight deviations from the situations specifically anticipated in full detail by their designers. Brittleness and inflexibility are often attributed to rule-based systems (see, e.g., (SOAR-Research-Group 2010)), and consequently to logical agents, due to their supposed over-commitment to particular courses of action.

In opposition to brittleness, (Brachman 2006) mentions *Versatility* as the ability of being able to perform previously unanticipated tasks. (Anderson and Perlis 2005) introduce the concept of *Perturbation Tolerance*, where a perturbation is any unanticipated change, either in the world or in the system itself, that impacts an agent's performance. To achieve Perturbation Tolerance, (Anderson and Perlis 2005) define a time-based *active logic* and a *Metacognitive Loop* (MCL), that involves a system monitoring, reasoning and meta-reasoning about and if necessary altering its own behavior.

In past work, we have introduced the possibility of maintaining knowledge bases by means of temporal-logic-based special meta-constraints to be dynamically checked with a certain (customizable) frequency. This approach has been applied to agent run-time self-checking(cf. (Costantini 2013b) and (Costantini 2012) for a preliminary longer version), and to memory management (Costantini and Gasperis 2013).

These constraints are based upon a simple interval temporal logic particularly tailored to the agent realm, A-ILTL ('Agent-Oriented Interval LTL', LTL standing as customary for 'Linear Temporal Logic'). Thus, properties can be defined that should hold according to what has happened and to what is supposed to happen or not to happen in the future, also considering partially specified event sequences. A-ILTL constraints be seen as composing a kind of MCL, where A-ILTL can be seen as an active logic.

As A-ILTL constraints are defined over formulas of an underlying logic language $\mathcal{L}$, a first contribution of this paper is to make A-ILTL constraints independent of $\mathcal{L}$. By linking A-ILTL constraints to the Evolutionary semantics of agent programs (originally introduced (Costantini and Tocchio 2006) and summarized below), such constraints can in fact be defined for any model-based (or initial-algebra-based) logical framework. We thus obtain a fairly general setting, that could be adopted in several logic agent-oriented languages and formalisms. In particular, one such language is DALI (Costantini and Tocchio 2002; 2004), where we have prototypically implemented the approach [1].

A second contribution of this paper is the proposal to exploit A-ILTL constraints, that were originally proposed for agent run-time self-repair or self-improvement, for the definition of complex reactivity patterns, dynamically adaptable to changing circumstances. Event processing (also called CEP, for "Complex Event Processing") has emerged as a relevant new field of software engineering and computer science (Chandy, Etzion, and von Ammon 2011). In fact, a lot of practical applications have the need to actively monitor vast quantities of event data to make automated decisions and take time-critical actions (Paschke and Kozlenkov 2009; Etzion 2010; Paschke, Vincent, and Springer 2011; Vincent 2011) (cf. also the Proceedings of the RuleML Workshop Series). Software products for event processing have appeared on the market, provided by major software vendors and by many start-up companies around the world (see e.g. (Paschke, Vincent, and Springer 2011; Vincent 2011) and the references therein). Many of the current approaches are declarative and based on rules, and often on logic-

[1]DALI is an agent-oriented extension to prolog that we have defined and developed in previous work (cf. (Costantini 2013a) for a comprehensive list of references about DALI, while the DALI interpreter is publicly available at (Costantini et al. 2012)).

programming-like languages and semantics: for instance, (Paschke and Kozlenkov 2009) is based upon a specifically defined interval-based Event Calculus. In (Costantini, Dell'Acqua, and Tocchio 2007) and later in (Costantini 2011; Costantini and De Gasperis 2012) we tackled the issue of complex reactivity in logical agents, by considering the possibility of choosing among different possible reactive patterns by means of complex preferences. In this paper, we show how A-ILTL temporal constraints may help to enforce suitable reaction patterns upon need. In this sense, A-ILTL constraints contribute to alleviate brittleness and improve versatility, since they make the agent able to react flexibly in the face of changing and unexpected circumstances: in fact, in our proposal reaction patterns can be defined on partially specified event sequences.

We assume that the reader is to some extent familiar with logic programming, and in particular with prolog-like logic programming, and to a small extent with Answer Set Programming (called 'ASP' in the rest of the paper for the sake of conciseness). The reader may refer to (Apt and Bol 1994) for the former and to (Gelfond 2007) for the latter, and to the references therein. As it is well-known, Answer Set Programming (Gelfond and Lifschitz 1988; Lifschitz 1999; Marek and Truszczyński 1999; Baral 2003) is a relatively recent and up to now well-established paradigm of logic programming which is not aimed at query-answering. Rather, for given answer set program, a *ASP solver* provides one, many (or no) answer sets. Each answer set represents a solution of the problem expressed in the program: if for instance the program aims at finding Hamiltonian paths in a graph, each answer set (if any) represents one such path.

Evolutionary Semantics

The Evolutionary semantics (Costantini and Tocchio 2006) is meant at providing a high-level general account of evolving agents, trying to abstract away from the details of specific agent-oriented frameworks. To this aim we define, in very general terms, an agent as the tuple $Ag = < P_{\mathcal{A}}, E >$ where $\mathcal{A}$ is the agent name and $P_{\mathcal{A}}$ (that we call "agent program") describes the agent according to some agent-oriented language $\mathcal{L}$. E is the set of the events that the agent is able to recognize or determine (so, E includes actions that the agent is able to perform), according to the specific agent-oriented framework.

Program $P_{\mathcal{A}}$ written by the programmer is in general transformed into an initial agent program P_0 by means of an *initialization step*. Later on, P_0 will go into execution, and will evolve according to events that happen, actions which are performed, etc., Evolution in our setting is recorded via corresponding program-transformation steps, each one transforming 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 semantics of P_i according to the semantics of $\mathcal{L}$. Notice in fact that the approach is parametric w.r.t $\mathcal{L}$. Let H be the *history* of an agent as recorded by the agent itself (in a form that will depend upon the specific agent-oriented framework), i.e., H includes agent's *memories*. For instance, in DALI the history consists of the set P of most recent "past events" and set PNV of previous instances (e.g., P may contain the last measure of temperature while PNV may contain older ones), plus *past constraints* that manage recording of events in the two sets.

Definition 1 (Evolutionary semantics) *Let Ag be an agent. The evolutionary semantics ε^{Ag} of Ag is a tuple $\langle H, PE, ME \rangle$, where H is the history of Ag, and PE and ME are respectively its program and semantic evolution sequences.*

The next definition introduces the notion of instant view of ε^{Ag}, at a certain stage of the evolution (which is in principle of unlimited length).

Definition 2 (Evolutionary semantics snapshot) *Let Ag be an agent, with evolutionary semantics $\varepsilon^{Ag} = \langle H, PE, ME \rangle$. The* snaphot at stage i *of ε_i^{Ag} is the tuple $\langle H_i, P_i, M_i \rangle$, where H_i is the history up to the events that have determined the transition from P_{i-1} to P_i.*

In (Costantini and Tocchio 2006) we have coped in detail with evolutionary semantics of DALI language, specifying which program transformation steps are associated with DALI language constructs. The approach however is in principle applicable to many other logical agent-oriented frameworks.

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, cf., e.g., (Emerson 1990)). These logics are called 'linear' because (in contrast to 'branching time' logics) they 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.

Based upon our prior work, in (Costantini 2012) we formally introduced an extension to the well-known linear temporal logic LTL based on *intervals*, called A-ILTL for 'Agent-Oriented Interval LTL'. Though, as discussed in (Costantini 2012), several "metric" and interval temporal logic exist, the introduction of A-ILTL is useful in the agent realm because the underlying discrete linear model of time and the complexity of the logic remains unchanged with respect to LTL. This simple formulation can thus be efficiently implemented, and is nevertheless 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 (Costantini 2012). A-ILTL expressions are (like plain LTL ones) interpreted in a discrete, linear model of time. Formally, this structure is represented by $\mathcal{M} = \langle \mathbb{N}, \mathcal{I} \rangle$ where, given countable set Σ of atomic propositions, interpretation function $\mathcal{I} : \mathbb{N} \mapsto 2^{\Sigma}$ maps each natural number i (representing state s_i) to a subset of Σ. Given set $\mathcal{F}$ of formulas built out of classical connectives and of LTL and A-ILTL operators (where however nesting of A-ILTL operators is not allowed), the semantics of a temporal formula is provided by a satisfaction relation: for $\varphi \in \mathcal{F}$ and $i \in \mathbb{N}$ we write $\mathcal{M}, i \models \varphi$ if, in the satisfaction relation, φ is true w.r.t. $\mathcal{M}, i$. We can also say (leaving $\mathcal{M}$ implicit) that φ *holds* at i, or equivalently in state s_i, or that state s_i satisfies φ. A structure $\mathcal{M} = \langle \mathbb{N}, \mathcal{I} \rangle$ is a model of φ if $\mathcal{M}, i \models \varphi$ for some $i \in \mathbb{N}$.

Some among the A-ILTL operators are the following.

Definition 3 *Let $\varphi \in \mathcal{F}$ and let m, n be positive integer numbers.*
$F_{m,n}$ *(*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. I.e., $\mathcal{M}, i \models F_{m,n}\varphi$ if there exists j such that $j \geq m$ and $j \leq n$ and $\mathcal{M}, j \models \varphi$. Can be customized into F_m,* bounded eventually (or "finally")*, where φ should become true somewhere on the path from the current state to the (m)-th state after the current one.*
$G_{m,n}$ *(*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. I.e., $\mathcal{M}, i \models G_{m,n}\varphi$ if for all j such that $j \geq m$ and $j \leq n$ $\mathcal{M}, j \models \varphi$. 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, i.e., $\mathcal{M}, i \models N_{m,n}\varphi$ if there not exists j such that $j \geq m$ and $j \leq n$ and $\mathcal{M}, j \models \varphi$.*

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 defined before. In fact, 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 (Henzinger, Manna, and Pnueli 1992) the following definition of *timed state sequence*, that we tailor to our setting.

Definition 4 *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 $e_{i+1} \neq e_i$ then $t_{i+1} > t_i$. In our setting, it will always be the case that $e_{i+1} \neq e_i$ as 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 $s_i = \rho_i$. This requires however a refinement: in fact, in a writing Op_m or $Op_{m,n}$ occurring in an agent program parameters m and n will not necessarily coincide with time instants of the above-defined timed state sequence. To fill this gap, in (Costantini 2012) a suitable approximation is introduced.

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 5 *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 say that τ holds/does not hold w.r.t. ρ_{Ag}.*

A-ILTL properties will be verified at run-time, and thus they act as *constraints* over the agent behavior[2]. 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 even be different for different properties. Then, 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 given agent can be defined within the agent program, though they constitute an additional but separate layer, composed of formulas $\{\tau_1, \ldots, \tau_l\}$. Agent evolution can be considered to be "satisfactory" if it obeys all these properties.

Definition 6 *Given agent Ag and given a set of A-ILTL expressions $\mathcal{A} = \{\tau_1, \ldots, \tau_l\}$, timed state sequence ρ_{Ag} is* coherent *w.r.t. $\mathcal{A}$ if A-ILTL formula $G\zeta$ with $\zeta = \tau_1 \wedge \ldots \wedge \tau_n$ holds.*

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

Notice that the expression $G\zeta$ is an *invariance property* in the sense of (Manna and Pnueli 1984). In fact, coherence requires this property to hold for the whole agent's "life". In the formulation $G_{m,n}\zeta$ that A-ILTL allows for, one can express *temporally limited coherence*, concerning for instance "critical" parts of an agent's operation. Or also, one might express forms of *partial* coherence concerning only some properties.

An "ideal" agent will have a coherent evolution, whatever its interactions with the environment can be, i.e., whatever sequence of events arrives to the agent from the external "world". However, in practical situations such a favorable case will seldom be the case, unless static verification has been able to ensure total correctness of agent's behavior. Instead, violations will occasionally occur, and actions should be undertaken so as to attempt to regain coherence for the future.

A-ILTL rules may imply asserting and retracting rules or sets of object rules ("modules"). In this setting, we are able to consider assert and retract as special A-ILTL operators, to which we provide a formal semantics (cf. (Costantini 2012)).

Complex Reaction: Past Work

In order to determine which actions can possibly be performed in reaction to an event in a given situation, in (Costantini 2011) we have introduced *reactive ASP modules* (ASP standing for "Answer Set Programming") that describe a situation and, triggered by events that have happened, will have answer sets which encompass the possible reactions to these events. A reactive ASP module has an input/output interface. The input interface specifies the event(s) that trigger the module. The output interface specifies the actions that the module answer sets (if any) can possibly encompass. Each answer set can give as output one or more actions, that can be selected either indifferently or according to preferences/priorities. So, there will be at least as many actions to consider (according to preconditions and preferences) as the number of answer set of M.

For preferences, some of the authors of this paper have proposed approaches to preferences in agents (Costantini, Dell'Acqua, and Tocchio 2007) or more generally in logic languages (Costantini and Formisano 2009; 2011). In particular, the approach defined in (Costantini and Formisano 2009) allows for the specification of various kinds of non-trivial preferences.

It can also be interesting to define reaction in terms of meta-statements involving possibility and necessity w.r.t. module M (cf. (Costantini 2011)), possibility of atom A with (pseudo-modal) meaning that M belongs to some answer set of M, and necessity that A belongs to every answer set of M. Precisely, given answer set program Π (also called 'module') with answer sets as $M_1, \ldots, M_k$, and an atom A, the *possibility* expression $P(w_i, A)$ is deemed to hold (w.r.t. Π) whenever $A \in M_{w_i}$, $w_i \in \{1, \ldots, k\}$. The possibility operator $P(A)$ is deemed to hold whenever $\exists M \in \{M_1, \ldots, M_k\}$ such that $A \in M$. Given answer set program Π with answer sets $M_1, \ldots, M_k$, and an atom A, the *necessity* expression $N(A)$ is deemed to hold (w.r.t. Π) whenever $A \in (M_1 \cap \ldots \cap M_k)$. Module Π can be implicit (if unique) or explicit, where expressions take the form $P(\Pi, w_i, A)$, $P(\Pi, A)$ and $N(\Pi, A)$ respectively.

Below are sample reactive patterns discussed in (Costantini and De Gasperis 2012), in a syntax which is reminiscent of DALI. In the first example, reaction to event evE (events are indicated with postfix E, reaction is indicated with $:>$) can be either any action produced by M as a possible reaction, or a *necessary* action, i.e., an action that belongs to all the answer sets of M. The latter is preferred in a critical situation, connective $>$ expressing simple conditional preference: the former option is preferred over the latter if the condition after the :- holds. Otherwise, any of the two options can be indifferently taken.

$evE :> necessary(M) | action(M).$
$necessary(M) \; > \; action(M)\text{:-}critical\_situation.$

The second example states that if a baby is hungry one should feed the baby with the available food (feeding is an action, indicated with postfix A), paying attention to choose the healthier. The conjunction $food(F)$, $available(F)$ provides a number of values for F, among which one is chosen that corresponds to a maximum in the partial order imposed by the binary predicate $healthier$. This construct for complex preference, the p-set, was originally introduced in (Costantini and Formisano 2009).

$baby\_is\_hungryE :>$
$\{feed\_babyA(F) : food(F), available(F) :$
$healthier\}.$

A-ILTL for Complex Reaction

In this section we illustrate the usefulness of A-ILTL constraints for defining and enforcing complex reaction patterns. To this aim, we use the *pragmatic* form that we have adopted in DALI, where an A-ILTL expression is represented as $OP(m, n; k)\, \varphi$: 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. For instance, $EVENTUALLY(m, n; k)\, \varphi$ 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. However, we allow variables to occur in an A-ILTL formula, to be instantiated via a *context* χ (we then talk about *contextual A-ILTL formulas*). Notice that, 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 (Costantini 2012) we have specified how to *operationally* check whether such a formula holds. This by observing that for most A-ILTL operators there is a *crucial state* where it is definitely possible to assess whether a related formula holds or not in given state sequence, by observing the sequence up to that point and ignoring the rest.

The following formulation deals with complex reaction according to a temporal condition. The way reaction is performed will depend upon the underlying language $\mathcal{L}$, and will be defined by an expression that we call *reactive pattern*. In DALI for instance, it can be any of the patterns defined in (Costantini and De Gasperis 2012).

Definition 7 *A reactive A-ILTL rule is of the form (where M, N, K can be either variables or constants)*

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

where:(i) $OP(M, N; K)\varphi :: \chi$ is a contextual A-ILTL formula, called the monitoring condition*, that should involve the observation of either external or internal events; (ii) ρ is called the* reactive part *of the rule, and it consists of a reactive pattern.*

Whenever the monitoring condition (automatically checked at frequency K) is violated (i.e., it does not hold) within given interval, then the reactive part ρ is executed.

Take for instance the example of a controller that has to keep the temperature in office hours (say between 8 a.m. and 5 p.m.) in the range 19–21 (celsius degrees). In this case, $temperature$ is an external event which is observed at a certain frequency by the system. If the condition is violated, a reaction will try to restore the wished-for situation. However, we assume to be in a smart building (where in fact the temperature is monitored by intelligent agents) where one is able to select, in order to modify the temperature, the best suitable energy source, for instance the less expensive. Notice that at different times of the day different fonts of energy can be less expensive. Remember also that the A-ILTL constraints is dynamically checked at a certain frequency (the default one if not specified explicitly, here say every 10 minutes). So, in given interval the monitoring condition will sometimes succeed (then nothing is done) and sometimes fail, where in the latter case the font of energy S which is cheaper *in that moment* is used in order to suitably affect the temperature and try to keep it in the specified range. In the proposed approach, this can be formalized as follows (where, as there are no variables, context is omitted):

$$\begin{array}{l} ALWAYS(8:00\,a.m.,\ 5:00\,p.m.; 10m) \\ \quad 19 \leq temperature \leq 21 \div \\ \quad modify\_temperatureA(S), S\ IN \\ \qquad \{external\_electricity, \\ \qquad\ gas, \\ \qquad\ solar\_panel\_electricity\ : \\ \qquad\qquad\qquad less\_expensive\} \end{array}$$

The next example is a meta-statement expressing open-minded commitment in agents, i.e., that a goal should be pursued until no longer believed possible. We suppose that a goal G being possible is evaluated w.r.t. a module M that represents the context for G (notation $P(G, M)$). In case the goal is still deemed to be possible and is not timed-out but has not been achieved yet, then the reaction consists in re-trying the goal, that might imply either resuming a suspended plan, or a re-planning.

$$\begin{array}{l} NEVER \\ \quad goal(G), \\ \quad eval\_context(G, M), P(G, M), \\ \quad not\ timed\_out(G) \\ \quad not\ achieved(G) \div \\ \qquad retry(G) \end{array}$$

Another possibility is not simply retrying the goal, but also reconsidering the evaluation context, that might for some reason have become obsolete. Thus, the reactive part might be

$$retry(G), reconsider\_context(G, M, M')$$

As an effect, if the goal should still remain unachieved, the evaluation module might be updated. Then, a subsequent check of the A-ILTL constraint might lead to stop retrying the goal.

Each element of the conjunction composing the reactive part can have preconditions. If preconditions do not hold for some element, then that element is skipped. In case for instance one would add the precondition that one can retry a goal if sufficient resources are available, i.e.,

$$retry(G) :< have\_resources(G)$$

then the goal would not be retried in the negative case.

Evolutionary Expressions

It can be useful to define properties to be checked upon arrival of event sequences, of which however only relevant events (and their order) should be considered. To this aim we introduce a new kind of A-ILTL constraints, that we call *Evolutionary A-ILTL Expressions*. To define partially known sequences of any length, we admit for event sequences a syntax inspired to that of regular expressions so as to specify irrelevant/unknown events, and repetitions (cf. (Costantini 2012)).

Definition 8 (Evolutionary LTL Expressions) *Let $\mathcal{S}^{\mathcal{E}vp}$ be a sequence of past events, and $\mathcal{S}^{\mathcal{F}}$ and $\mathcal{J}^{\mathcal{J}}$ be sequences of events. Let τ be a contextual A-ILTL formula $Op\,\varphi\ ::\ \chi$. An* Evolutionary LTL Expression *ϖ is of the form $\mathcal{S}^{\mathcal{E}vp}\ :\ \tau\ :::\ \mathcal{S}^{\mathcal{F}}\ ::::\ \mathcal{J}^{\mathcal{J}}$ where: (i) $\mathcal{S}^{\mathcal{E}vp}$ denotes the sequence of relevant events which are supposed to have happened, and in which order, for the rule to be checked; i.e., these events act as preconditions: whenever one or more of them happen in given order, τ will be checked; (ii) $\mathcal{S}^{\mathcal{F}}$ denotes the events that are expected to happen in the future without affecting τ; (iii) $\mathcal{J}^{\mathcal{J}}$ denotes the events that are expected* not *to happen in the future; i.e., whenever any of them should happen, φ is not required to hold any longer, as these are "breaking events".*

An Evolutionary LTL Expression can be evaluated w.r.t. a state s_i which includes among its components the *history* of the agent, i.e., the list of past events perceived by the agent. A history H *satisfies* an event sequence S whenever all events in S occur in H, in the order specified by S itself.

Definition 9 *An Evolutionary A-ILTL Expression ϖ, of the form specified in Definition 8: (1)* holds *in state s_i whenever (i) history H_i satisfies $\mathcal{S}^{\mathcal{E}vp}$ and $\mathcal{S}^{\mathcal{F}}$ and does not include any event in $\mathcal{J}^{\mathcal{J}}$, and τ holds or (ii) H_i includes any event occurring in $\mathcal{J}^{\mathcal{J}}$ (the expression is* broken*); (2) is* violated *in state s_i whenever H_i satisfies $\mathcal{S}^{\mathcal{E}vp}$ and $\mathcal{S}^{\mathcal{F}}$ and does not include any event in $\mathcal{J}^{\mathcal{J}}$, and τ does not hold.*

Operationally, an Evolutionary A-ILTL Expression can be finally deemed to hold if either the critical state has been reached and τ holds, or an unwanted event has occurred. Instead, an expression can be deemed *not* to hold (or, as we say, to be *violated* as far as it expresses a wished-for property) whenever τ is false at some point without the occurrence of breaking events.

The following is an example of Evolutionary A-ILTL Expression stating that, after withdrawing a sum at the cash machine (P standing for 'past' indicates a past event, occurring at time T, in this case a past action), one is supposed to have money in her wallet for at least one week, despite everyday expenses, unless a robbery occurs.

$withdrawalP : T$:
$ALWAYS(T, T + 1_{week})\ have\_cash$
::: $minor\_expenses*$
:::: $robbery$

Whenever an Evolutionary A-ILTL expression is either violated or broken, a reaction can be attempted aiming at recovering a desirable or at least acceptable agent's state.

Definition 10 *An evolutionary LTL expression with repair* ϖ^r *is of the form* $\varpi|\eta_1||\eta_2$ *where* ϖ *is an Evolutionary LTL Expression adopted in language* $\mathcal{L}$*, and* η_1, η_2 *are atoms of* $\mathcal{L}$. η_1 *will be executed (according to* $\mathcal{L}$*'s procedural semantics) whenever* ϖ *is violated, and* η_2 *will be executed whenever* ϖ *is broken.* η_1 *and* η_2 *are called* countermeasures.

In previous example, countermeasure η_1 might imply either withdrawing more money, or asking a friend for some money, or cutting down expenses. η_2 might imply reporting to the police. Also in this case preferences may come into play. For instance, one may prefer to withdraw again if she is rich, or to ask a friend is she has urgent needs. In fact, several preferences can be expressed at the same time. If several actions are equally preferred, a feasible one will be nondeterministically selected. The overall expression will take the form:

$withdrawalP : T$:
$ALWAYS(T, T + 1_{week})\ have\_cash$
::: $minor\_expenses*$
:::: $robbery$
| $withdrawalA > cut\_expensesA$:- $rich$
$ask\_friendA > cut\_expensesA$:- $urgent\_needs$
|| $report\_to\_policeA$

Evolutionary LTL expressions can be further enhanced by introducing a third kind of counter-action, aimed at preventing a potentially breaking event from disrupting the wished-for property. In the following example, having taken an appointment for a certain day and time D, one does not want to be late. Normally, being punctual would be made impossible by a bus strike, which is a potential breaking event for the property. However, a third kind of counter-action may be introduced aimed at trying to disable the potential breaking property. In the example, it consists in either renting a car or taking a taxi, according to which option is considered to be more effective in terms of time.

$appointmentP(D) : T$:
$NEVER)\ late(D)$
:::: bus_strike
||| $alternative\_transportation\ IN$
$\{rent\_car, taxi : quicker\}$

In previous example, another potential breaking property might be an unexpected major expense, to which one might for instance react by asking a relative for money. The formulation becomes the following.

$withdrawalP : T$:
$ALWAYS(T, T + 1_{week})\ have\_cash$
::: $minor\_expenses*$
:::: $unexpected\_major\_expense$
| $withdrawalA > cut\_expensesA$:- $rich$
$ask\_friendA > cut\_expensesA$:- $urgent\_needs$
||| $ask\_relative\_for\_moneyA$

The modified formal definitions are easy to devise, though we are forced to omit them here for lack of space.

Related Work and Concluding Remarks

In this paper, we have generalized our past work on A-ILTL run-time constraints in logical agents. On the one hand we have made them parametric with respect to the underlying logic. On the other hand, we have devised a version of A-ILTL constraints aimed at complex reactivity in Complex Event Processing environments. The overall aim is that of contributing to the construction of non-brittle adaptable logical agents.

We may easily notice similarities between A-ILTL constraints and event-calculus formulations (Kowalski and Sergot 1986). Also, approaches based on abductive logic programming such as, SCIFF (cf. (Montali et al. 2011) and the references therein) allow one to model dynamically upcoming events, and specify positive and negative expectations, and the concepts of fulfilment and violation of expectations. Reactive Event Calculus (REC) stems from SCIFF (Bragaglia et al. 2012) and adds more flexibility by reacting to new events by extending and revising previously computed results. However, these approaches have been devised for static checking or dynamic checking when performed by a third party. Event sequences, the concepts of violated and broken expressions, complex reaction patterns, and independence of the underlying logic are however distinguished features of the proposed approach. We drew inspiration from Cohen and Levesque work on rational agency (cf., e.g., (Cohen and Levesque 1990)), thus our focus is not on observable behavior to be confronted with expectations: rather, A-ILTL rules are aimed at expressing inherent agent properties, and at defining what can be done to enforce these properties. As mentioned, we see a close relationship to the approach of (Anderson and Perlis 2005).

We have been experimenting the approach in the context of energy management in smart buildings (Caianiello et al. 2013). Such intelligent control must be dynamic by nature, including real-time requirement as the building has its own dynamical thermo-physical behavior and is immersed in a dynamical environment where weather events change its energy footprint in function of time. The outcome of the experiments is encouraging, in the sense that adopting agents equipped with the proposed features allows for not only general but also local (room-by-room or area-by-area) control of energy saving according to user comfort requirements and preferences.

References

Anderson, M. L., and Perlis, D. 2005. Logic, self-awareness and self-improvement: the metacognitive loop and the problem of brittleness. *J. Log. Comput.* 15(1):21–40.

Apt, K. R., and Bol, R. 1994. Logic programming and negation: A survey. *The Journal of Logic Programming* 19-20:9–71.

Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press.

Brachman, R. J. 2006. (AA)AI more than the sum of its parts. *AI Magazine* 27(4):19–34.

Bragaglia, S.; Chesani, F.; Mello, P.; Montali, M.; and Torroni, P. 2012. Reactive event calculus for monitoring global computing applications. In Artikis, A.; Craven, R.; Cicekli, N. K.; Sadighi, B.; and Stathis, K., eds., *Logic Programs, Norms and Action - Essays in Honor of Marek J. Sergot on the Occasion of His 60th Birthday*, volume 7360 of *Lecture Notes in Computer Science*, 123–146. Springer.

Caianiello, P.; Costantini, S.; Gasperis, G. D.; Florio, N.; and Gobbo, F. 2013. Application of hybrid agents to smart energy management of a prosumer node. In *Proc. of DCAI 2013, 10th International Symposium on Distributed Computing and Artificial Intelligence*, Advances in Intelligent and Soft Computing. Springer. to appear.

Chandy, M. K.; Etzion, O.; and von Ammon, R. 2011. 10201 Executive Summary and Manifesto – Event Processing. In Chandy, K. M.; Etzion, O.; and von Ammon, R., eds., *Event Processing*, number 10201 in Dagstuhl Seminar Proceedings. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Cohen, P. R., and Levesque, H. J. 1990. Intention is choice with commitment. *Artif. Intell.* 42(2-3):213–261.

Costantini, S., and De Gasperis, G. 2012. Complex reactivity with preferences in rule-based agents. In Bikakis, A., and Giurca, A., eds., *Rules on the Web: Research and Applications, RuleML 2012 - Europe, Montpellier, France, August 27-29, 2012. Proceedings*, volume 6826 of *Lecture Notes in Computer Science*, 167–181. Springer.

Costantini, S., and Formisano, A. 2009. Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of of Algorithms in Cognition, Informatics and Logic* 64(1).

Costantini, S., and Formisano, A. 2011. Weight constraints with preferences in ASP. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR2011)*, Lecture Notes in Computer Science. Springer.

Costantini, S., and Gasperis, G. D. 2013. Memory, experience and adaptation in logical agents. In *Proc. of ISMIS 2013, International Symposium on Management Intelligent Systems*, Advances in Intelligent and Soft Computing. Springer. to appear.

Costantini, S., and Tocchio, A. 2002. 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.

Costantini, S., and Tocchio, A. 2004. 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.

Costantini, S., and Tocchio, A. 2006. About declarative semantics of logic-based agent languages. In Baldoni, M.; Endriss, U.; Omicini, A.; and Torroni, P., eds., *Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Selected and Revised Papers*, volume 3904 of *LNAI*. Springer. 106–123.

Costantini, S.; D'Alessandro, S.; Lanti, D.; Tocchio, A.; and al. 2012. DALI web site, download of the interpreter. Released: basic DALI features. For beta versions please ask the authors.

Costantini, S.; Dell'Acqua, P.; and Tocchio, A. 2007. Expressing preferences declaratively in logic-based agent languages. In *Proc. of Commonsense'07, the 8th International Symposium on Logical Formalizations of Commonsense Reasoning*. AAAI Press. Event in honor of the 80th birthday of John McCarthy.

Costantini, S. 2011. Answer set modules for logical agents. In de Moor, O.; Gottlob, G.; Furche, T.; and Sellers, A., eds., *Datalog Reloaded: First International Workshop, Datalog 2010*, volume 6702 of *LNCS*. Springer. Revised selected papers.

Costantini, S. 2012. Self-checking logical agents. In *Proc. of LA-NMR 2012*, volume 911. CEUR Workshop Proceedings (CEUR-WS.org). Invited paper.

Costantini, S. 2013a. The DALI agent-oriented logic programming language: References. at URL http://www.di.univaq.it/stefcost/info.htm.

Costantini, S. 2013b. Self-checking logical agents. In *Proc. of AAMAS 2013, twelfth Intern. Conf. on Autonomous Agents and Multi-Agent Systems*. Sheridan Communications. Extended Abstract, to appear.

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

Etzion, O. 2010. Event processing - past, present and future. *Proceedings of the VLDB Endowment, PVLDB Journal* 3(2):1651–1652.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proc. of the Fifth Joint Int. Conf. and Symposium*, 1070–1080. MIT Press.

Gelfond, M. 2007. Answer sets. In *Handbook of Knowledge Representation, Chapter 7*. Elsevier.

Henzinger, T. A.; Manna, Z.; and Pnueli, A. 1992. Timed transition systems. In de Bakker, J. W.; Huizing, C.; de Roever, W. P.; and Rozenberg, G., eds., *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, volume 600 of *Lecture Notes in Computer Science*, 226–251. Springer.

Kowalski, R., and Sergot, M. 1986. A logic-based calculus of events. *New Generation Computing* 4:67–95.

Lifschitz, V. 1999. Answer set planning. In *Proc. of the 16th Intl. Conference on Logic Programming*, 23–37.

Manna, Z., and Pnueli, A. 1984. Adequate proof principles for invariance and liveness properties of concurrent programs. *Sci. Comput. Program.* 4(3):257–289.

Marek, V. W., and Truszczyński, M. 1999. *Stable logic programming - an alternative logic programming paradigm*. Springer. 375–398.

Montali, M.; Chesani, F.; Mello, P.; and Torroni, P. 2011. Modeling and verifying business processes and choreographies through the abductive proof procedure sciff and its extensions. *Intelligenza Artificiale, Intl. J. of the Italian Association AI*IA* 5(1).

Paschke, A., and Kozlenkov, A. 2009. Rule-based event processing and reaction rules. In *RuleML*, volume 5858 of *Lecture Notes in Computer Science*, 53–66. Springer.

Paschke, A.; Vincent, P.; and Springer, F. 2011. Standards for complex event processing and reaction rules. In Olken, F.; Palmirani, M.; and Sottara, D., eds., *RuleML America*, volume 7018 of *Lecture Notes in Computer Science*, 128–139. Springer.

SOAR-Research-Group. 2010. SOAR: A comparison with rule-based systems. URL: http://sitemaker.umich.edu/soar/home.

Vincent, P. 2011. Event-driven rules: Experiences in cep. In Olken, F.; Palmirani, M.; and Sottara, D., eds., *RuleML America*, volume 7018 of *Lecture Notes in Computer Science*, 11. Springer.