Complex Reactivity with Preferences in Rule-Based Agents

Stefania Costantini and Giovanni De Gasperis[1]

Dip. di Ingegneria e Scienze dell'Informazione e Matematica (DISIM), Università di L'Aquila, Coppito 67100, L'Aquila, Italy
{stefania.costantini, giovanni.degasperis}@univaq.it

Abstract. In this paper, we extend our previous work on complex reaction in rule-based logical agents. In particular, we introduce the possibility of defining and exploiting complex preferences for choosing the course of action to undertake in response to external events, also based upon a (simplified) form of modal reasoning and on sequences of past events.

1 Introduction

In past work we developed DALI, a logic agent-oriented language that extends prolog with reactive and proactive features [1,2,3,4] (cf. [5] for a comprehensive list of references). DALI in fact is equipped with "event-condition-action" rules (ECA rules) for defining the behavior of an agent in consequence of perception of external events. A distinguished feature that makes DALI proactive and strongly event-oriented is the *internal events*, i.e., the programmer can indicate internal conditions to be interpreted as events, to which a reaction can be defined. These conditions are checked automatically at a certain frequency, and then treated exactly like external events. The DALI interpreter provides directives that allow a programmer to influence reactivity by indicating at which frequency (if different from default one) events occurrence should be checked, and also since when and until when and/or upon which conditions. We defined management of events which occurred together (with the possibility to customize the time interval defining 'together') and priorities between events. In [6] and later in [7] we tackled the issue of complex reactivity in logical agents, by considering the possibility of choosing among different possible reactive patterns by means of simple preferences.

In the meanwhile, event processing (also called CEP, for "Complex Event Processing") has emerged as a relevant new field of software engineering and computer science [8]. 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 [9,10,11,12] (cf. also the Proceedings of the RuleML Workshop Series). Many 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. [11,12] and the references therein). With cloud computing, the effectiveness and usefulness of event processing is even more visible, and a connection of "event pattern languages" with ontologies and with the semantic web is envisaged. Many of the current approaches are declarative and based on rules, and often on logic-programming-like languages and semantics: for instance, [9] is based upon a specifically defined interval-based Event Calculus.

In this paper, we continue and extend the work of [6] by introducing more complex forms of preferences among applicable reactive behaviors. Such preferences can be also defined in terms of "possible worlds" elicited from a declarative description of a current or hypothetical situation, and can depend upon past events, and the specific sequence in which they occurred. To the best of our knowledge, these advancements are orthogonal to the various proposals appeared in the literature, into which they could be possibly merged. The declarative formalism that we use for describing a situation is Answer Set Programming (ASP, cf., [13,14,15,16]), that has proved apt at a variety of complex reasoning tasks (cf., among others, [17,18,19,16] and the references therein), and provides various options for connection to ontologies and semantic web frameworks (cf., e.g., [20] and the references therein).

As we have prototypically implemented the proposed approach using the DALI language [1,2] (cf. [5] for a complete list of references about DALI), in the rest of the paper we use a sample syntax which is reminiscent of DALI. However, we invite the reader not to consider this syntax as mandatory. In fact, we have tried to design the approach so as to make it applicable to many languages and frameworks. The approach is particularly well-suited for rule-based (even better prolog- or datalog-based) languages. It would not be difficult to transform our syntax into a more standard XML-like form, so as to fit into the area of Semantic Web and Rule Markup Languages (e.g., RuleML [21]). Some of the features of our approach could be in fact usefully integrated into existing approaches to CEP. In this paper we do not propose examples taken from real industrial applications. We instead propose intuitive simple examples that should make it easier to understand the new proposed features. We assume the reader to have basic notions of logic programming and Answer Set Programming.

2 Background

In this section, we recall parts of our previous work that define some basic foundation elements of the proposed approach.

2.1 Declarative Semantics of Logic Agent-Oriented languages

We base our proposal upon the declarative semantic framework introduced in [22], aimed at encompassing approaches to evolving logical agents, by understanding changes determined by external events and by the agent's own activities as the result of the application of program-transformation functions.

We abstractly formalize an agent as the tuple $Ag = \langle P_{Ag}, E, I, A \rangle$ where Ag is the agent name and P_{Ag} is the "agent program" according to the specific language adopted. E is the set of the external events, i.e, events that the agent is capable to perceive and recognize: let $E = \{E_1, \ldots, E_n\}$ for some n. I is the set of internal events (distinguished internal conclusions, that may include desires and intentions): let $I = \{I_1, \ldots, I_m\}$ for some m. A is the set of actions that the agent can possibly perform: let $A = \{A_1, \ldots, A_k\}$ for some k. Let $\mathcal{Y} = (E \cup I \cup A)$.

In DALI syntax, used below for the examples, atoms indicated with a postfix correspond to events of various kinds. In particular, if p is an atom, pE is an external event, pA is an action and pI an internal event.

According to this semantic account, one will have an initial program $Pe_0 = P_{Ag}$ which, according to events that happen, agent's activities and actions which are performed, will pass through corresponding program-transformation steps (each one transforming Pe_i into Pe_{i+1}, cf. [22]), and thus gives rise to a Program Evolution Sequence $PE = [Pe_0, ..., Pe_n, ...]$. The program evolution sequence will imply a corresponding Semantic Evolution Sequence $ME = [M_0, ..., M_n, ...]$ where M_i is the semantic account of Pe_i.

Different languages and different formalisms in which an agent can possibly be expressed will influence the following key points: (i) when a transition from Pe_i to Pe_{i+1} takes place, i.e., which are the external and internal factors that determine a change in the agent; (ii) which kind of transformations are performed; (iii) which semantic approach is adopted, i.e., how M_i is obtained from Pe_i.

The semantic account includes an *Initialization step*, where the program P_{Ag} written by the programmer is transformed into a corresponding program Pe_0 by means of some sort of knowledge compilation. In DALI for instance, the initialization step extracts the list of internal and external events, and the control directives that are associated to the program (e.g., for defining priorities among events and frequencies for checking the occurrence of events). In general in fact, Pe_0 can be simply a program (logical theory) or can have additional control information associated to it.

Agents usually record events that happened and actions that they performed. Notice that an agent can describe the state of the world only in terms of its perceptions, where more recent remembrances define the agent's approximation of the current state of affairs. We thus define set Pof current (i.e., most recent) past events, and a set PNV where we store all previous ones. We define the 'history' H of an agent as the tuple $\langle \mathcal{P}, PNV \rangle$, dynamically augmented with new events that happen. In DALI, a past event in $\mathcal{P}$ is in the form $pP : T_i$, where p is an atom corresponding to an event, postfix P stands for 'past' and T_i is a time-stamp indicating when the event has been perceived. In [23] we have defined *Past Constraints*, which allow one to define when and upon which conditions (apart from arrival of more recent versions) past events should be moved into PNV.

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 its program and semantic evolution sequence.*

In [22] the detailed semantic treatment of some basic agent-oriented constructs is provided, in particular that of "condition-action-rules"

$IF \ \langle Conditions \rangle \ DO \ \langle Actions \rangle$

that define simple reactivity: whenever the $Conditions$ hold, the corresponding $Actions$ are performed. Whenever the conditions include external events that have happened, such rules are called "event-condition-action-rules" (ECAs). These rules have been introduced in logic agent-oriented languages since the seminal work of [24].

In our sample syntax, a reactive rule will be indicated with $pE :> Body$ meaning that whenever the external event p is perceived (as said, for the sake of immediate readability external events are indicated with postfix E), the agent will execute $Body$. In what follows, for the sake of clarity we will assume the body to consist just of a single action. This implies no loss of generality as the extension to a more general form is easily done. In [22], these rules are treated in the initialization step so as to be transformed into an intermediate form then managed by a suitably defined program-transformation step.

2.2 Answer Set Modules

In [7], we have proposed kinds of ASP (Answer Set Programming) modules to be invoked by a logical agents. In particular, one kind is defined so as to allow forms of reasoning to be expressed on possibility and necessity analogous to those of modal logic. In this approach, the "possible worlds" that we consider refer to an ASP program Π and are its answer sets. Therefore, given atom A, we say that A is possible if it belongs to some answer set, and that A is necessary if it belongs to the intersection of all the answer sets.

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.

Possibility and necessity can possibly be evaluated within a context, i.e., if $E(Args)$ ($E = P$ or $E = N$) is either a possibility or a necessity expression, the corresponding *contextual* expression has the form $E(Args) : Context$ where $Context$ is a set of ground facts and rules. $E(Args) : Context$ is deemed to hold whenever $E(Args)$ holds w.r.t. $\Pi \cup Context$, where, with some abuse of notation, we mean that each rule in $Context$ is added to Π. The answer set module T where to evaluate an operator can possibly be explicitly specified, in the form: $E(T, Args) : Context$.

In this approach, one is able for instance to define meta-axioms, like, e.g., the following, which states that a proposition is plausible w.r.t. theory T if, say, it is possible in at least two different worlds, given context C:

$$plausible(T, Q, C) \leftarrow P(T, I, Q) : C, P(T, J, Q) : C, I \neq J.$$

3 Complex Reactivity with Preferences in Logical Agents: Past Work, Integrations and Extensions

In [6] we have proposed to employ preferences for defining forms of complex reactivity in logical agents.

The studies of the processes that support the construction or the elicitation of preferences have historically deep roots. In logic, [25] initiated a line of research that was subsequently systematized in [26] and continues nowadays, e.g, in [27] and [28]. Preferences handling in computational logic has been extensively studied too. The reader may refer to [29,30] for recent overviews and discussion of many existing approaches to preferences that for lack of space we cannot directly mention here. The approach of [31] adopts in particular a form of defeasible logic.

Some of the authors of this paper have proposed approaches to preferences in agents [6] or more generally in logic languages [32,33]. In particular, the approach defined in [32] allows for the specification of various kinds of non-trivial preferences. These preferences follow the quite intuitive principles first formalized in [26], and illustrated at length, e.g., in [27]. The first two principles state that any preference relation is asymmetric and transitive. For simplicity we stick to strict preferences, i.e., if in a certain context one prefers ϕ to ψ, then in the same context one cannot also prefer ψ to ϕ. In our approach, each preference holds in the context of the rule where it is defined. Different (even contrasting) preferences can be expressed (and simultaneously hold) in different rules. The third principle states that preferring ϕ to ψ means that a state of affairs where $\phi \wedge \neg\psi$ holds is preferred to a state of affairs where $\psi \wedge \neg\phi$ holds. The fourth principle states that if one prefers ψ to $(\phi \vee \zeta)$ then (s)he will prefer ψ to ϕ and ψ to ζ. Finally, the last principle states that a change in the world might influence the preference order between two states of affairs, but if all conditions stay constant in the world ("ceteris paribus"), then so does the preference order.

We propose an example (reported from [34]) in order to illustrate the approach to preferences in logical agents and languages that we have developed in previous work [6,32,33]. The logic program below defines a recipe for a dessert. The construct $icecream > zabaglione$ is called a *p-list* (preference list) and states that with the given ingredients one might obtain either ice-cream or zabaglione, but the former is preferred. This is, in the terminology of [26], an "intrinsic preference", i.e., a preference without a specific reason. In preparing the dessert, one might employ either skim-milk or whole milk. The p-list $skimmilk > wholemilk \leftarrow diet$ states that, if on a diet, the former is preferred. Finally, to spice the dessert, one would choose, by the *p-set* $\{chocolate, nuts, coconut\ :\ less\_caloric\}$, the less caloric among chocolate, nuts, and coconut. These are instead instances of "extrinsic preferences", i.e., preferences which come with some kind of "reason", or "justification". Notice that, in an agent, extrinsic preferences may change even non-monotonically as the agent's knowledge base evolves in time, as the justification can be any conjunction of literals.

$$\begin{aligned} &icecream > zabaglione \leftarrow egg,\ sugar, (skimmilk > wholemilk \leftarrow diet),\\ &\qquad\qquad\qquad\qquad\qquad \{chocolate, nuts, coconut\ :\ less\_caloric\}. \end{aligned}$$

$$\begin{aligned} &less\_caloric(X,Y)\ \leftarrow\ calory(X,A), calory(Y,B), A < B.\\ &calory(nuts,2). \qquad calory(coconut,3). \end{aligned}$$

In general terms, a rule such as the first one in the above program fragment 'fires', i.e. can be applied, when the body is entailed by the present knowledge base. In particular, using skim milk or whole milk is conditioned from 'diet' (if both are available), and the preferred outcome is 'ice-cream'. Consider however that preferences expressed in this rule must be combined to preferences possibly expressed in other rules. There are several politics for doing so, discussed in the above references. In the approach of [35] these politics can be explicitly specified via suitable operators by associating a set of *preference rules* to given program.

The above features can be smoothly incorporated into agent-oriented rule-based languages. In fact, the evolutionary semantics presented in [22] can easily accommodate this kind of preference reasoning.

As a first step towards complex reactivity, in the approach of [6], a disjunction among two or more actions may occur in the body of a reactive rule that specifies the response to an event. Associated to the reactive rule are preferences, which are local to the rule where the disjunction occurs, that establish which action is preferred under which conditions; actions as customary have preconditions, thus in any situation the agent should perform the best preferred feasible action.

As an agent evolves in time and its knowledge changes, preferred choices will change as well. Then, according to the same preference structure an agent will in general prefer differently at different stages of its life.

In our sample syntax, we assume action a to be represented as aA. Actions may have preconditions: the connective $:<$ indicates that a rule defines the precondition of an action. I.e., a precondition rule will be indicated as $qA :< Body$, meaning that the action qA can be performed only if $Body$ is true. We do not cope here with the effective execution of actions, that is left to the language run-time support.

A disjunction (indicated with "|") of actions may occur in the body of a reactive rule. Then, a rule $pE :> q1A \mid q2A, Body.$ means that in reaction to pE the agent may perform indifferently either action $q1A$ or action $q2A$ and then it executes $Body$ [1].

Preferences among actions are defined in *preference condition expressions* associated to a reactive rule. Then, a rule $pE :> q1A \mid q2A, Body \,::\, q2A \,>\, q1A \,\text{:-}\, Conds.$ means that in reaction to pE the agent may perform either action $q1A$ or action $q2A$, but action $q2A$ is preferred over action $q1A$ provided that $Conds$ holds. I.e., if $Conds$ is not verified then the preference is not applicable, and thus any of the actions can be indifferently executed. In general, a disjunction may contain several actions, and several preference condition expressions can be expressed, by means of preference lists (seen before).

These expressions define a *partial order* among actions, where preferences are transitively applied and actions that are unordered can be indifferently executed. In our approach preferences are applied on *feasible* actions. I.e., the partial order among actions must be re-evaluated at each step of the agent life where a choice is possible, according to the preconditions of the actions. The preferred actions at each stage are those that can actually be performed and that are selected by the preference partial order.

Example 1. Consider a person who receives an invitation to go out for dinner. She would prefer accepting the invitation rather than refusing, provided that the invitation comes from nice people. She is able to accept if she has time. The invitation is an *external event* that reaches the agent from her external environment. Accepting or refusing constitutes the *reaction* to the event, and both are actions. One of the actions (namely, accepting) has preconditions, i.e., to have time, and the money to pay for the restourant. In our sample syntax, an agent program fragment formalizing this situation may look as follows.

[1] Notice that, as reactive rules perform forward reasoning, the part after the $:>$ is a 'consequence' of the head and not vice versa. Procedurally, any of $q1A, q2A, Body$ might fail.

$invitation\_dinnerE :> acceptA \mid refuseA ::$
$\quad acceptA\ >\ refuseA$:- $nice\_people\_inviting.$
$acceptA :< have\_time, have\_money.$

Notice that what the agent will do is not known in advance, as the agent evolves in time: the invitation may arrive at a stage of the agent operation when time and money are available, and then the preferred action is chosen. If instead the invitation arrives when there are no resources for accepting, then the agent will have to refuse.

Consider instead a person who receives an invitation to a boring work meeting. She will prefer (unconditionally) to refuse. However, she cannot do that if she does not have an excuse to present. As we can see, preference among the same actions varies according to the context. Also, if the preconditions of a preferred action are not verified, a less preferred one will have to be performed.

$invitation\_meetingE :> acceptA \mid refuseA ::$
$\quad refuseA\ >\ acceptA.$
$refuseA :< acceptable\_excuse.$

In [6], semantics of reaction with preference is provided as a suitable customization of the evolutionary semantics seen before. In particular, the program evolution step related to reaction with preferences does the following: (i) evaluates actions preconditions to identify feasible actions; (ii) solves the optimization problem defined by preference expressions (where any action among equally preferred ones can be indifferently selected); (iii) replaces the reactive rules with preferences with a plain reactive rule where a best preferred action occurs; (iv) performs as usual for reactive rules.

An immediate extension that can be done to [6] is to allow all forms of p-lists and p-sets as seen above to occur in preference expressions.

A limitation of the approach is that possible actions have to be listed explicitly, while in different situations different action sets should be considered by the agent. For instance, if one is at home in bed with a flu, (s)he cannot possibly accept an invitation, whatever the preference. A distinction between preconditions of actions and general feasibility/unfeasibility is in order. In fact, for the sake of elaboration-tolerance [36] it is impractical and sometimes impossible to specify in advance all circumstances that may prevent an action from being feasible. In our opinion it is better to evaluate general feasibility w.r.t. the present situation, and then evaluate specific preconditions.

In order to determine which actions can possibly be performed in reaction to an event in a given situation, in [7] we have introduced *reactive ASP modules* that describe a situation and, triggered by events that have happened, will have answer sets which encompass the possible reactions to these events (e.g., in the above examples, there will possibly be an answer set containing $acceptA$ and another one containing $refuseA$. 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. A possible new form of the above rule can be for instance:

$invitation\_meetingE :> action(M) ::$
$\quad refuseA > acceptA.$
$refuseA :< acceptable\_excuse.$

where after the $:>$ it is stated that any of the actions that are outputs of the ASP module M can be possibly performed, given the local preconditions and the preferences. Module M must be updated according to the present *context*. Say, e.g., that in at present the meeting is important but one has flu: then, the outcome instead of $acceptA$ or $refuseA$ could be for instance $postponeA$.

It can be intersting to define reaction in terms of meta-statements involving possibility and necessity w.r.t. module M. For instance, in the following example the reaction to event evE 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.

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

4 Complex Reactivity with Preferences in Logical Agents: Modal Preferences

Often, the kind of reaction that an agent might prefer to pursue in consequence of a certain event is related to the objectives that the agent intends to reach, or to conditions that the agent would like to fulfill. In order to introduce new more involved forms of reactions possibly based on commonsense and non-monotonic reasoning, or on "local" planning about the consequences that selected reaction strategies may have, we improve the forms of preferential reasoning introduced so far.

To this aim, by drawing inspiration from the work of [28], we define further extensions to expressing preferences in logical agents[2]. In particular, referring to agents, [28] introduces a concept of complex preference where an agent prefers ϕ over ψ if, for any "plausible" (i.e., presumably reachable) world where ψ holds, there exists a world which is *at least as good* as this world and *at least as plausible* where ϕ is true. [28] writes $B(\psi \rightarrow \langle H \rangle \phi)$ where H is a new modality, and the reading is "Hopefully ϕ". Semantically: if $\mathcal{M}$ is a preference model encompassing a set of worlds W and $s, t \in W$, $\leq$ is a reachability relation meaning "at least as plausible" and $\preccurlyeq$ a preference relation, we have that:

$$\mathcal{M}, s \models H\,\phi \textit{ iff for all } t \textit{ with both } s \leq t \textit{ and } s \preccurlyeq t \;:\; \mathcal{M}, t \models \phi$$

ASP modules are a good tool for redefining, extending and implementing the H operator in practical languages. In particular, to stay within a logic programming computationally affordable setting, we do not fully represent reachability and preferability among worlds. Rather, "possible worlds" will be interpreted as the answer sets of a suitable ASP module Π representing the situation at hand. So, all the answer sets of Π are equally reachable. Preferability among worlds is explicitly stated as a property that the

[2] A preliminary version of part of the material presented in this section was presented in [37]

agent desires to hold. Thus, we define the H operator in terms of the aspect that makes a world in which ϕ holds preferable. In fact, we want to express preferences such as the following, where one may choose to prefer a certain food rather than another one, in the hope that the preferred food is good for health:

$$eat(pasta) \ > \ eat(meat) \ : \ H(healthy)$$

Here, ψ is $eat(meat)$ while ϕ is $eat(pasta)$, and I prefer a world where ϕ holds because I hope that $healthy$ (speaking of myself) will hold as well. Below we indicate this third novel element that we introduce, i.e., the hoped-for reason for preference, with ξ.

As a basic step for defining such a preference we define $\phi \ : \ H(\xi)$ (or $\phi \ : \ H(\Pi, \xi)$ if ASP module is explicitly indicated) meaning that, assuming ϕ, we expect that ξ will hold in some reachable world, that in our setting is an answer set of an ASP module that can be either implicit or explicitly indicated. Thus, ξ is the "reason why" reachable worlds in which ϕ holds are preferred. We can thus define the operator H by a simple adaptation of the contextual possibility operator as introduced in Section 2.2, where ϕ is taken as the context.

Definition 2. *Given an answer set module Π with answer sets M_1,. . .,M_k, an atom ϕ and an atom ξ, the expression $\phi \ : \ H(\Pi, \xi)$ is deemed to hold (w.r.t. Π) whenever the contextual possibility expression $P(\Pi, \xi) \ : \ \phi$ holds. The answer set module Π can be left implicit if it is unique, thus leading to the simplified form $\phi \ : \ H(\xi)$.*

We can now formally define the above preference expression as follows.

Definition 3. *Given atoms A, B, C and answer set module T, the construct $A > B \ : \ H(T, C)$ is called an* mp-list *(modal preference list) meaning that A is preferred to B (i.e., we have the p-list $A > B$) if $A \ : \ H(T, C)$ holds. Otherwise, any of A or B can be indifferently chosen.*

A similar but stronger formulation of previous example can be the following, where one chooses to prefer a food that in most situations can "reasonably" expected to procure better health (where as before these situations are interpreted as the answer sets of an underlying ASP module). The operator $maxH$ ("maximal hope") intuitively means that the former food is preferred if it is most likely to procure good health.

$$eat(pasta) \ > \ eat(meat) \ : \ maxH(healthy)$$

To define these more involved preferences, the basic operator H must be extended to a form $\phi \ : \ H[N](\xi)$ meaning that, given ϕ, the hoped-for property ξ holds in exactly N different possible worlds.

Definition 4. *Given an answer set module Π with answer sets M_1,. . .,M_k, an atom ϕ and an atom ξ, the expression $\phi \ : \ H[n](\xi)$ is deemed to hold (w.r.t. Π) whenever there exist $\{v_1, \ldots, v_n\}$, $v_i \in \{1, \ldots, k\}$ such that $P(v_i, \xi) \ : \ \phi$ holds, $i \leq n$, and for every $P(w_i, \xi) \ : \ \phi$ which holds, $w_i \in \{v_1, \ldots, v_n\}$. By convention, we assume $\phi \ : \ H[0](\xi)$ to signify that $\phi \ : \ H[n](\xi)$ holds for no n.*

We can now extend Definition 3 so as to compare A and B w.r.t. how often a satisfactory state of affairs can be reached. That is, we compare A and B on the basis of hoped-for condition C. We prefer A over B if we assess that by assuming A it is more plausible to reach C, i.e., C holds in more worlds than it is by assuming B.

Definition 5. *Given atoms A, B, C and answer set module T, the construct $A > B\ :\ maxH(T, C)$ is called an* mmp-list *(modal max-preference list) meaning that A is preferred to B (i.e., we have the p-list $A > B$) iff we have $A\ :\ H[N_A](T, C)$ and $B\ :\ H[N_B](T, C)$, and $N_A \geq N_B$.*

The extension to preference lists with more than two elements is straightforward. The "preference condition" C can be generalized to conjunctions. Operators H and $maxH$ can be contextual, in the form:

$$A\ :\ Context\ :\ H(T, C) \text{ and } A\ :\ Context\ :\ maxH(T, C)$$

where $Context$ is a conjunction of rules, each of which to be added to the ASP module T before evaluating H. This allows for expressions such as:

$$A > B\ :\ Context\ :\ H(T, C)\ or, respectively,\ A > B\ :\ Context\ :\ maxH(T, C)$$

The aim of introducing contextual operators is expressivity in the practical sense: this formulation in fact allows one to state in when and why one has a certain preference. We can thus propose for instance the following variation of the above example:

$$eat(fruit\_salad)\ >\ eat(cake)\ :\ diabetes\ :\ maxH(healthy)$$

It can be also useful to introduce a generalized form of p-set, so as to allow for instance for the representation below, which takes the varieties of food generated by $food(F)$ and the context $diabetes$, and generates a p-list where the various kinds f of foods are ordered according to the degree of healthiness, interpreted as the value of N in expression $f\ :\ diabetes\ :\ H[N]healthy$.

$$\{food(F),\ eatA(F)\ :\ diabetes\ :\ H(healthy)\}$$

To define these new expressions, that we call *modal p-sets*, we have to start from their ground version, where atoms $eat(f)$ are explicitly listed, instead of being generated by $food(F)$.

Definition 6. *A* ground modal p-set *(gmp-set) is an expression of the form:*

$$\{A_1, \ldots, A_s :\ B\ :\ H(T, E)\}$$

where T is an ASP module, the A_is are atoms, and B, E are conjunctions of atoms (B possibly empty). This expression stands for the p-list $A_1\ >\ \ldots\ >\ A_s$ where for each A_j, A_k in this p-list, A_j precedes (is preferred to) A_k iff the following expression holds:

$$A_j > A_k\ :\ B\ :\ maxH(T, E))$$

Then, to introduce an implicit specification of the A_i's such as the one adopted in the above example (thus avoiding to list all them explicitly), we resort to a solution similar to the one adopted in Answer Set Programming for weight constraints. In [38], the authors introduce a notion of a conditional literal of the form $l : d$ where l is a literal and the conditional part d is a domain predicate, where the subset of given program defining *domain predicates* consists of *domain rules*, syntactically restricted so that their extension should be relatively efficiently computable.

Definition 7. *A* modal p-set *(mp-set) is an expression of the form:*

$$\{p(X_1, \ldots, X_n),\ q(X_1, \ldots, X_n) : B : H(T, E)\}$$

where T is an ASP module, q is a predicate (possibly defining an action), p is a domain predicate, $X_1, \ldots, X_n$ are variables, B is a conjunction of atoms not involving the X_is and involving terms $Y_1, \ldots, Y_v$, $v \geq 0$ and E is a conjunction of atoms possibly involving the X_is and the Y_js. This expression stands for the gmp-set

$$\{A_1, \ldots, A_s : B : H(T, E)\}$$

where the A_is are all the possible ground atoms of the form $q(t_1, \ldots, t_n)$ such that $p(t_1, \ldots, t_n)$ holds.

Modal p-sets allow for the definition of a variety of useful statements and meta-statements for complex reaction. In the following example for instance, in facing a danger the agent has to choose between screaming, running or phoning to the police. The choice will be done that, in the present situation, provides the best expectation of a happy ending of the adventure. This is obtained via a ground modal p-set.

$dangerE :>$
$\quad\quad \{call\_policeA, runA, screamA : H(safe)\}.$
$call\_policeA :< have\_phone.$

The following example states that if a baby is hungry one should feed the baby with attention to healthy food.

$baby\_is\_hungryE :>$
$\quad\quad \{food(F),\ give\_food\_to\_babyA(F) : : H(healthy)\}.$

It is important to emphasize that the quality of reactive reasoning obtainable via the H and $maxH$ operators is high, as ASP allows one to declaratively represent many forms of reasoning and commonsense reasoning, and is able, e.g., to model complex forms of configuration and planning and of reasoning about resources (cf., among others, [17,18,19,16,39] and the references therein). ASP also provides various options for connection to ontologies and semantic web frameworks (cf., e.g., [20,40,41] and the references therein).

The evolutionary semantics can be customized to manage the above reactive rules by defining a program evolution step that: (i) computes the answer sets of involved ASP modules; (ii) computes preconditions of actions; (iii) computes preferences among feasible actions; (iv) re-writes reactive rule into standard form and treats them accordingly.

5 Complex Reactivity with Preferences in Logical Agents: Conditional Preferences on Event Sequences

There can be situations where the course of actions to be undertaken depends upon what happened in the past, i.e., upon past events. Past events may occur in the preconditions of actions, so this need is partially covered already in standard DALI language. However, sometimes the order and the number of times in which events have happened count, thus another extension is in order. In the following example, one prefers (for the sake of civility) to accept an invitation from nice people if it has been reiterated several times in the past without being accepted. One instead prefers to refuse an invitation from relatives already accepted several times. We apologize because for lack of space we use a simplified syntax that doesn't really allow specific invitations to be coupled to the related acceptance/rejection, so the example just provides and intuitive idea of the envisaged construct.

For event sequences we adapt the syntax of regular expressions (for lack of space we can't formally describe the adapted syntax here, some more detail is provided in [42]).

$invitation\_dinnerE :> acceptA \mid refuseA ::$
$\quad invitation\_dinner_P^{E^+}\ not\ accept_P^A :$
$\quad\quad acceptA\ >\ refuseA$:- $nice\_people\_inviting.$
$\quad invitation\_dinner_P^{E^+}\ accept_P^{A^+} :$
$\quad\quad refuseA\ >\ acceptA$:- $relatives\_inviting.$
$acceptA :< have\_time, have\_money.$

To manage this case, the evolutionary semantics must provide a program evolution step that evaluates a preference expression only if the specified event sequence matches the agent's knowledge base. If several alternatives are feasible, some politics will be applied so as to select one and put it into play.

6 Concluding Remarks

For lack of space we can't discuss here related work, and in particular the relationship and the possible integration of our approach with the interesting work of [9] that proposes a very comprehensive logical reaction rule language that explicitly considers time intervals. As mentioned however, we believe that the treatment of preferences and the use of ASP modules for reasoning about present situation in terms of what is possible and/or necessary and of objectives that can be reached is new.

Though DALI is fully implemented and has been widely experimented in practical (also industrial) applications (see [43] for a recent application that has won the third prize as best demo), up to now there is no full implementation available for the approach proposed here. Rather, some features have been implemented and some simulated in DALI, mainly by means of the "internal events" construct. We have also implemented in DALI the ASP modules and the related operators. A future aim is that of constructing a full implementation an instance of the proposed framework. In the agent context, we intend to consider KGP [44,45,46], DALI and EVOLP [47] (which are fully-defined and fully-implemented approaches to logical agents) that provide the main elements and can be exploited in combination in an implementation. However, it would also be very interesting to seek an integration with existing (even commercial) approaches and languages for CEP.

In the perspective of CEP, the present proposal fits into a wider framework where we provide (cf. [37,23,42,48] and the references therein) dynamic self-checking, mandatory whenever it is not known in advance which events will happen and in which order, and advanced memory management.

Acknowledgements

My mentor Gaetano Aurelio Lanzarone ('Elio' for his friends) died at the age of 66 in October 2011 after a long illness. He has been one of the pioneers of prolog and rule-based automated reasoning in Italy, and initiated me into this fascinating research area. My affection and gratitude to Elio will last forever. This paper is dedicated to him.

References

1. Costantini, S., Tocchio, A.: A logic programming language for multi-agent systems. In: Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf.,JELIA 2002. LNAI 2424, Springer-Verlag, Berlin (2002)
2. Costantini, S., Tocchio, A.: The DALI logic programming agent-oriented language. In: Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004. LNAI 3229, Springer-Verlag, Berlin (2004)
3. Tocchio, A.: Multi-Agent systems in computational logic. PhD thesis, Dipartimento di Informatica, Università degli Studi di L'Aquila (2005)
4. Costantini, S., D'Alessandro, S., Lanti, D., Tocchio, A.: Dali web site, download of the interpreter (2010) http://www.di.univaq.it/stefcost/Sito-Web-DALI/WEB-DALI/index.php.
5. Costantini, S.: The dali agent-oriented logic programming language: References (2012) at URL http://www.di.univaq.it/stefcost/info.htm.
6. Costantini, S., Dell'Acqua, P., Tocchio, A.: 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 (2007) Event in honor of the 80th birthday of John McCarthy.
7. Costantini, S.: Answer set modules for logical agents. In de Moor, O., Gottlob, G., Furche, T., Sellers, A., eds.: Datalog Reloaded: First International Workshop, Datalog 2010. Volume 6702 of LNCS. Springer (2011) Revised selected papers.
8. Chandy, M.K., Etzion, O., von Ammon, R.: 10201 Executive Summary and Manifesto – Event Processing. In Chandy, K.M., Etzion, O., von Ammon, R., eds.: Event Processing. Number 10201 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2011)
9. Paschke, A., Kozlenkov, A.: Rule-based event processing and reaction rules. In: RuleML. Volume 5858 of Lecture Notes in Computer Science., Springer (2009) 53–66
10. Etzion, O.: Event processing - past, present and future. Proceedings of the VLDB Endowment, PVLDB Journal **3**(2) (2010) 1651–1652
11. Paschke, A., Vincent, P., Springer, F.: Standards for complex event processing and reaction rules. In Olken, F., Palmirani, M., Sottara, D., eds.: RuleML America. Volume 7018 of Lecture Notes in Computer Science., Springer (2011) 128–139

12. Vincent, P.: Event-driven rules: Experiences in cep. In Olken, F., Palmirani, M., Sottara, D., eds.: RuleML America. Volume 7018 of Lecture Notes in Computer Science., Springer (2011) 11
13. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88). The MIT Press (1988) 1070–1080
14. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
15. Marek, V.W., Truszczyński, M.: Stable logic programming - An alternative logic programming paradigm. In: 25 years of Logic Programming Paradigm. Springer (1999) 375–398
16. Gelfond, M.: Answer sets. In: Handbook of Knowledge Representation. Elsevier (2007)
17. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)
18. Leone, N.: Logic programming and nonmonotonic reasoning: From theory to systems and applications. In Baral, C., Brewka, G., Schlipf, J., eds.: Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007. (2007)
19. Truszczyński, M.: Logic programming for knowledge representation. In Dahl, V., Niemelä, I., eds.: Logic Programming, 23rd International Conference, ICLP 2007. (2007) 76–88
20. Eiter, T., Brewka, G., Dao-Tran, M., Fink, M., Ianni, G., Krennwallner, T.: Combining nonmonotonic knowledge bases with external sources. In Ghilardi, S., Sebastiani, R., eds.: Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, September 16-18, 2009. Proceedings. Volume 5749 of Lecture Notes in Computer Science., Springer (2009) 18–42
21. Boley, H.: The ruleml family of web rule languages. In: Principles and Practice of Semantic Web Reasoning, 4th International Workshop, PPSWR 2006. (2006) 1–17
22. Costantini, S., Tocchio, A.: About declarative semantics of logic-based agent languages. In Baldoni, M., Torroni, P., eds.: Declarative Agent Languages and Technologies. LNAI 3229. Springer-Verlag, Berlin (2006)
23. Costantini, S.: Defining and maintaining agent's experience in logical agents. In: Proc. of the Seventh Latin American Workshop on Non-Monotonic Reasoning LANMR 2011. Volume 804. (2011) 151–165 (also in the Informal Proc. of the LPMAS "Logic Programming for Multi-Agent Systems" Workshop at ICLP 2011).
24. Kowalski, R., Sadri, F.: Towards a unified agent architecture that combines rationality with reactivity. In: Logic in Databases. Number 1154 in Lecture Notes in Computer Science. Springer-Verlag (1996) 135–149
25. Hallden, S.: On the logic of better. Library of Theoria, No. 2, Lund: Library of Theoria. Cambridge University Press (1957)
26. von Wright, G.H.: The logic of preference. Edinburgh University Press (1963)
27. van Benthem, J., Girard, P., Roy, O.: Everything else being equal: A modal logic for ceteris paribus preferences. J. Philos. Logic **38** (2009) 83–125
28. Liu, F.: Von Wright "the logic of preference" revisited. Synthese **175**(1) (2009) 69–88
29. Delgrande, J., Schaub, T., Tompits, H., Wang, K.: A classification and survey of preference handling approaches in nonmonotonic reasoning. Computational Intelligence **20**(12) (2004) 308–334
30. Brewka, G., Niemelä, I., Truszczyński, M.: Preferences and nonmonotonic reasoning. AI Magazine **29**(4) (2008)
31. Dastani, M., Governatori, G., Rotolo, A., van der Torre, L.: Programming cognitive agents in defeasible logic. In: Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference LPAR 2005, Springer (2005) 621–636

32. Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. Journal of of Algorithms in Cognition, Informatics and Logic **64**(1) (2009)
33. Costantini, S., Formisano, A.: 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 (2011)
34. Costantini, S., Formisano, A.: Augmenting weight constraints with complex preferences. In: Logical Formalizations of Commonsense Reasoning, Papers from the 2011 AAAI Spring Symposium, USA, AAAI Press (2011) –
35. Brewka, G.: Complex preferences for answer set optimization. In Dubois, D., Welty, C.A., Williams, M.A., eds.: Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004. (2004) 213–223
36. McCarthy, J.: Elaboration tolerance. In: Proc. of Common Sense'98. (1998) Available at http://www-formal.stanford.edu/jmc/ elaboration.html.
37. Costantini, S., Dell'Acqua, P., Pereira, L., Toni, F.: Meta-axioms and complex preferences in evolving logical agents. In: Proc. of 15th Portuguese Conference on Artificial Intelligence. (2011) – also in the Informal Proc. of the LPMAS "Logic Programming for Multi-Agent Systems" Workshop at ICLP 2011.
38. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
39. Costantini, S., Formisano, A.: Answer set programming with resources. Journal of Logic and Computation **20**(2) (2010) 533–571
40. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. **172**(12-13) (2008) 1495–1539
41. Eiter, T., Ianni, G., Krennwallner, T., Polleres, A.: Rules and ontologies for the semantic web. In: Reasoning Web. Volume 5224 of Lecture Notes in Computer Science., Springer (2008) 1–53 Tutorial Lecture.
42. Costantini, S.: Temporal meta-axioms in logical agents. submitted
43. Bevar, V., Muccini, H., Costantini, S., Gasperis, G.D., Tocchio, A.: The DALI logic programming agent-oriented language. In: Proc. of the 10th Conference on Practical Applications of Agents and Multi-Agent Systems. Advances in Intelligent and Soft Computing, Springer, Berlin, in press (2012) Paper and demo.
44. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP model of agency: Computational model and prototype implementation. In: Global Computing: IST/FET International Workshop, Revised Selected Papers. LNAI 3267. Springer-Verlag, Berlin (2005) 340–367
45. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of agency. In: Proc. ECAI-2004. (2004)
46. Stathis, K., Toni, F.: Ambient Intelligence using KGP Agents. In Markopoulos, P., Eggen, B., Aarts, E.H.L., Crowley, J.L., eds.: Proceedings of the 2nd European Symposium for Ambient Intelligence (EUSAI 2004). LNCS 3295, Springer Verlag (2004) 351–362
47. Alferes, J.J., Brogi, A., Leite, J.A., Pereira, L.M.: Evolving logic programs. In: Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002. LNAI 2424, Springer-Verlag, Berlin (2002) 50–61
48. Costantini, S.: Memory, experience and adaptation in logical agents. submitted