Meta-axioms and Complex Preferences in Evolving Logical Agents

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

[1] Dip. di Informatica, Università di L'Aquila, Coppito 67010, L'Aquila, Italy
stefcost@di.univaq.it
[2] Dept. of Science and Technology - ITN, Linköping University, Norrköping, Sweden
pierangelo.dellacqua@liu.se
[3] Dept. de Informática, Centro de Inteligência Artificial (CENTRIA), Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt
[4] Dept. of Computing, Imperial College London, South-Kensington Campus, London, UK
ft@doc.ic.ac.uk

Abstract. In this paper, we introduce some improvements to a framework that we have previously proposed, based upon a multi-layered general agent model, where at each layer temporal-logic-like axioms and meta-axioms can be defined and dynamically checked. Their violation determines suitable repair actions to be undertaken by means of appropriate agent's self-modifications. This in the perspective of a society of agents, where individuals should be able to learn and enlarge their patterns of behavior by observing and generalizing their observations, and also by "imitating" other agents, or by "being told" by them. Here, we extend the overall setting on the one hand by augmenting the temporal-logic-like rules with an ordered conjunction for checking/enforcing an order on events and actions, and on the other hand by introducing the possibility of defining and exploiting complex preferences based upon a (simplified) form of modal reasoning.

1 Introduction

In previous work (cf. [21,13,15,14]), we have introduced several elements that in our view languages and architectures for logical agents might usefully encompass. We have started from our experience in developing and implementing logic agent-oriented languages and architectures (namely KGP [5,27,32], DALI [18,19,33,10] and EVOLP [1]) from which we have tried to go ahead towards a comprehensive setting.

Among the many possible applications of "our" agents, we particularly envisage a framework where agents interact with users and assist them in their tasks, especially in situations where the user is "disabled" either physically or psychologically, in that the environment is either unknown or for some reason difficult to cope with. In our view, a system which is a realization of the envisaged framework will bring to a user the following potential advantages: the user is relieved of some of the responsibilities related to her behavior, as directions about the "right thing" to do are constantly and punctually provided. She is assisted in situations where she perceives herself as inadequate, in

some respect, to perform her activities or tasks. She is possibly told how to cope with unknown, unwanted or challenging circumstances. She interacts with a "Personal Assistant" that improves in time, both in its "comprehension" of the user needs, cultural level, preferred kinds of explanations, etc. and in its ability to cope with the environment. This is in accordance with the vision of Ambient Intelligence as that of a digitally augmented environment centered around the needs of humans, where appliances and services proactively and un-intrusively provide support and assistance.

A flexible interaction with the user and with the other agents will be made easier by adopting a multi-layered underlying agent model where there is a base level, that we call PA for "Personal Assistant", and one or more meta-layers, that we call MPA. While the PA is responsible for the direct interaction with the user, the MPA is responsible for correct and timely PA behavior. Thus, while the PA monitors the user, the MPA monitors the PA. We have introduced the agent model and its semantics, that we summarize below, in [21,13,11]. The actions the PA will be able to undertake will include, for instance, behavioral suggestions, appliance manipulation, enabling or disabling user interventions. The actions the MPA will be able to undertake will include the modification of the PA in terms of adding/removing knowledge (modules) in the attempt at correcting inadequacies and generating a more appropriate agent behavior. In our framework, both the PA and the MPA will widely base their activity upon verification of temporal-logic-like rules that describe expected and un-expected/unwanted situations.

To this aim, in [11] we have defined A-ILTL (for "Agent-Interval LTL"), an extension to the well-known LTL logic which is tailored to the agent's world in view of run-time verification. The envisaged agents will try to either modify or reinforce the rules/plans/patterns they hold, based on appropriate evaluation performed by the internal MPA meta-control component. The evaluation might also convince the agent to modify its own behavior by means of advanced evolution capabilities. In the general view that we are pursuing, we consider to be necessary for an agent to acquire knowledge from other agents, i.e. learning "by being told" instead of learning only by experience. We treat at some length the topic of learning "by being told" in [12].

In this paper, we aim at improving and enlarging the language features that can be profitably used both in the PA and MPA components. One the one hand we extend A-ILTL formulas by introducing an ordered conjunction which allows one to state the required order for events/action to occur, and which should be the relationship between the respective time-stamps. On the other hand, we introduce in our framework the possibility of defining and exploiting complex preferences in agent program's rules. The novel aspects of these preferences (for which we draw inspiration from, adapt and extend some aspects discussed in [29]) w.r.t. existing approaches is their "modal" fashion: preferring one alternative over others is based upon reasoning about what can be hopefully achieved by adopting that alternative. The motivation of the proposed improvements lies in the experiments that we have performed in the realm of user monitoring and training [13,14]. In fact, ordered conjunction allows a monitoring agent to check/enforce that user's actions are performed at the due time in the due order, also with respect to events which may happen. Handling user preferences is a key feature of all context-aware systems, to achieve an effective personalization. However, what a user

(or an agent) may prefer may vary according to the present context and objectives. To achieve this kind of flexible preferences, we adapt and extend the approach to reasoning about possibility and necessity in inner modules introduced in [9].

The paper is structured as follows. In Section 2 we outline the basic elements of our previous work, namely: the underlying general agent model and its semantics; A-ILTL, an interval temporal logic aimed at defining axioms and meta-axioms to be dynamically checked. In Section 3 we introduce the new contributions. Finally, in Section 4 we conclude. We assume that the reader has basic notions of logic programming (cf., [30,2]) and Answer Set Programming (ASP, cf., [24,25,31]).

2 Envisaged Agent: Background

In this section, we recall the parts of our previous work that define some basic elements which in our view each logic agent-oriented programming language might profitably encompass. We outline a very general agent model, then we introduce temporal-logic-like expressions to be checked at run-time, that allow for the definition of useful statements and meta-statements for an agent self-checking and self-regulation.

2.1 Agent model: basic structure

In order to meet the vision outlined in Section 1, we consider an agent as composed of two layers. A *base layer* PA (for "Personal Assistant") in charge of giving immediate answers to a user. We will assume that PA is a logic program, but will not commit to a particular semantics for it (for a survey of semantics for logic programs, see e.g. [2]). We will assume however a semantics possibly ascribing multiple models to PA, in order to deal with "uncertainty" (as we will see later). One such a semantics might be the answer set semantics [24]. A *meta-layer* MPA (for "Meta-Personal Assistant") in charge of: (i) supervising and checking the activity of PA, in order to ensure that on the one hand PA does not violate basic conditions on its behavior and on the other hand that PA behaves in a *sufficiently good* way. Later on we will discuss these aspects at some length; (ii) updating PA when either no model exists according to the chosen semantics for PA or a violation has been detected during the supervising activity. This meta-layer relies on meta-knowledge, e.g. reporting long-term objectives about the user (like for instance safety and good health) and some domain-dependent meta-knowledge related to the PA. This domain-dependent knowledge may be updated by learning (by being told) from other agents.

2.2 Agent model: operational behavior

Below we sketch the operational behavior of this agent model, which is further described in [21]. Each agent, once created, will in general pass through a sequence of stages, since it will be affected by the interaction with the environment, that will lead it to respond, to set and pursue goals, to either record or prune items of information, etc. This process, that we can call the agent *life*, can be understood in at least two ways: (i) the same agent proceeding into a sequence of states, each state encoding the present

version of beliefs, desires, intentions and each state transition encoding a control step; (ii) successive transformations of the initial agent into new agents, which are its descendants where the program has changed by modifying the beliefs, desires, intentions, and by learning and belief revision steps; each transformation is determined by the step that has been done. Formally, an agent starts from a program that defines it, according to the given agent model. In this paper we adopt the latter view, that of an evolution of the initial agent into subsequent (related) ones. In this paper, for the sake of generality and in order to propose a flexible framework adaptable to many real-world contexts, we do not go any deeper into the feature of the agent model that we refer to simply as $\mathcal{M}$.

Definition 1. *An agent program $\mathcal{P}_{\mathcal{M}}$ is a tuple $\langle PA, MPA, \mathcal{C}, \mathcal{CI}\rangle$ of software components where PA and MPA are logic programs, $\mathcal{C}$ is the control component and $\mathcal{CI}$ is some related (optional) control information.*

The control information $\mathcal{CI}$ is given as input to $\mathcal{C}$ together with PA and MPA. While however PA and MPA are programs written in a logic agent-oriented language, $\mathcal{CI}$ contains a set of *directives* that can affect in a relevant way the run-time behavior of an agent. Typically, directives will state at least which are the priorities among different events/actions the agent has to cope with and at which frequency they will be taken into consideration. Therefore, by modifying the control information while not touching the code, one can obtain a "lazy" agent rather that an "eager" one or affect the "interest" that the agent will show with respect to different matters. We can take the agent program as the *initial state* of the agent, where nothing has happened yet.

Definition 2. *The* initial agent A_0 *is an agent program $\mathcal{P}_{0\mathcal{M}}$ (or simply $\mathcal{P}_0$ when $\mathcal{M}$ is clear from the context), i.e., $\langle PA_0, MPA_0, \mathcal{C}_0, \mathcal{CI}_0\rangle$.*

The operational behavior of the agent will result from the control component and the control information, which rely on an underlying control mechanism that implements the operational counterpart of the agent model.

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

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

This general agent model and operational semantics admits for example KGP [5,27,32] and DALI [18,19,33] as instances.

Operationally, two different solutions are possible. In the first one $\mathcal{U}$ provides different parallel threads for the PA and MPA: therefore, MPA continuously monitors PA and can possibly make interventions in case of problems. In the second one (where no parallel threads are possible) the control is interleaved between PA and MPA where in turn a series of steps is performed at the PA level and a sequence of steps is performed at the MPA level. Control will *shift up* from the PA to the MPA by an act that is sometimes called *upward reflection* either periodically, in order to perform constraint verification

at the meta-level, or upon some conditions that may occur. Control will *shift down* by *downward reflection* from the MPA to the PA on completion of the MPA activities (a general theory of reflection in logic programming languages has been developed in [3]). How frequently and upon which conditions there will be a shift is control information that can be encoded in $\mathcal{CI}_i$. For a working examples of this kind of behavior, one can consider the *internal events* mechanism of DALI [20]. The MPA interventions on the PA may encompass a modification of the PA by replacing some of its rules/components by others.

Notice that the A_js do not deterministically follow from A_0, as there is the unforeseen interaction with the external environment. A full "evolutionary" declarative semantics that is not specific to a language/approach but is rather designed to encompass a variety of computational-logic based approaches, thus accounting for the agent model proposed here, is described in [20]. However, we also intend to explore how to express the semantics in terms of the Domain Theory proposed in [7,8].

2.3 Temporal logic rules and meta-rules

We assume that both PA and MPA include rules inspired by temporal logic, but adapted to the agent context. As discussed above, MPA will check whether some constraints on agent activity are respected. The checks that we consider here are not supposed to be performed in advance in model-checking style. Instead, here we consider run-time verification of properties: violations, if present, are treated by means of some kind of *repair*, and by possibly modifying PA. As it is not possible to foresee in advance all possible states that our agents can reach by interacting with both the user and the environment, we do not adopt a temporal logic based on a branching time, i.e., based on separately considering all paths that the agent may undertake. Rather, we intend to check that some properties are verified anyway, no matter the chosen path. So we adopt LTL [4,23,28], a "Linear Time Logic", that implicitly quantifies universally upon all possible paths. We have proposed in [11] an extension called A-ILTL, for "Agent-Interval LTL", which is tailored to the agent's world in view of run-time verification. Based on this new logic, we are able to enrich agent programs by means of A-ILTL rules. These rules are defined upon a logic-programming-like set of formulas where all variables are implicitly universally quantified. In this setting, the negation operator not is interpreted (as usual) as negation-as-failure.

As for LTL, the semantics of A-ILTL operators is given in terms of an infinite sequence $\pi : s_0, s_1, s_2, \ldots$ of states of a system, stating if and where each operator holds in states of the sequence. States are often understood as subsequent time instants. We introduce the possibility of accessing the current state (or "time"): the proposition $\tau(s_i)$ is true if s_i is the current state. Let φ be a proposition. The A-ILTL operators are the LTL operators plus the following ones. X_m, i.e. $X_m\varphi$ means that formula φ should be true at state s_m. F_m stands for bounded eventually, i.e., $F_m\varphi$ means that φ eventually has to hold somewhere on the path from the current state to s_m. $G_{m,n}$ stands for always in a given interval, i.e., $G_{m,n}\varphi$ means that φ should become true at most at state s_m and then hold at least until state s_n. $G_{\langle m,n\rangle}$ means that φ should become true just in s_m and then hold until state s_n, and not in s_{n+1}, where nothing is said for the remaining states. N stands for "never", i.e. $N\varphi$ means that φ should not become true in any future

state. $N_{m,n}$ stands for "bounded never", i.e. $N_{m,n}\varphi$ means that φ should not be true in any state between s_m and s_n, included.

In practice, run-time verification of A-ILTL properties may not occur at every state (of the given interval). Rather, properties will 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 infinite sequence $\pi : s_0, s_1, s_2, \ldots$ of states of the system 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.

The representation of A-ILTL operators within a logic agent-oriented programming language can be, e.g., the one illustrated in Table 1, that we have adopted in DALI, where m and n denote the time interval in which the formula must hold and k is the frequency. When not needed from the context, we omit the arguments of the operator

A-ILTL Op^k	**OP(m,n;k)**
$\tau(t)$	$NOW(t)$
X^k	$NEXT(1;k)$
${X_j}^k$	$NEXT(j;k)$
F^k	$FINALLY(1;k)$
F_m^k	$FINALLY(m;k)$
G^k	$ALWAYS(1;k)$
${G_{m,n}}^k$	$ALWAYS(m,n;k)$
${G_{\langle m,n \rangle}}^k$	$ALWAYS\_2(m,n;k)$
N^k	$NEVER(1;k)$
${N_{m,n}}^k$	$NEVER(m,n;k)$

Table 1. A-ILTL operators

and simply write OP (instead of $OP(m,n;k)$).

Definition 5. *Given a set S of literals (i.e., atoms and negated atoms), we write $conj(S)$ to indicate the set of all the conjunctions that can be formed by using literals in S. Let m, n and k be natural numbers. Define the set Q as follows: (i) $S \subset Q$ (ii) if $\varphi \in conj(Q)$, then $OP(m,n;k)\,\varphi \in Q$. An A-ILTL rule is any rule of the form $OP(m,n;k)\,\varphi$ in Q.*

In many monitoring situations, one has to check that what *normally* should happen actually occur. The occurrence of an event is said to be "normal" when it occurs sufficiently often, if not always. We define a new operator called $USUALLY$ in terms of $ALWAYS$ that is checked at a certain frequency, say f, that reinforces "normality".

Definition 6. *Given a sentence φ and a natural number f, we let $USUALLY\varphi$ = $ALWAYS(1;f)\varphi$.*

Notice that frequencies can possibly be specified separately, e.g., as control information included in $\mathcal{CI}$. For simplicity, both the interval and the frequency, indicated in the

definition as $(m, n; k)$, can be omitted if not relevant to understand the context. That is, one can write $ALWAYS\varphi$ if φ has to be checked on all the future states. Some examples of use of A-ILTL rules are presented in the following sections. To show the potential of A-ILTL rules we define below a check for an agent that, once decided to achieve the goal g, is blindly committed to actually obtain g within a given deadline d. After the deadline, a resource-bounded agent can possibly drop the commitment (or keep it, but only if possible). The fact $goal(g)$ means that g is a goal that has been selected to be executed. $achieved(g)$ means that the plan for reaching the goal g has been successfully completed. In contrast, $dropped(g)$ means that the agent has given up any attempt to achieve g. The following A-ILTL rule checks that an agent respects the blind commitment to its goals.

$$NEVER$$
$$(goal(G), deadline(G, T), NOW(T1), T1 \leq T, not\ achieved(G), dropped(G))$$

In order to fulfill their semantic specification, A-ILTL rules must be ground when they are evaluated, i.e. no variables must occur in them. For instance, in the above example the evaluation will involve ground instances obtained by suitably instantiating the variables G, T and $T1$.

A-ILTL rules with time-related variables The syntax of A-ILTL rules defines time instants as constants. We introduce a further extension where time instants can possibly be variables which are instantiated by what we call an *evaluation context*.

Definition 7. *Let $OP(m, n; k)\varphi$ be an A-ILTL rule. The corresponding* contextual A-ILTL rule *has the form $OP(M, N; K)\varphi :: \chi$ where:*

- *M, N and K can be either variables or constants;*
- *χ is called the* evaluation context *of the rule, and consists of a quantifier-free conjunctions of literals;*
- *each of the M, N and K which is a variable* must *occur in an atom (non-negated literal) in χ.*

A contextual A-ILTL rule will be evaluated whenever ground. The context χ can possibly instantiate not only the time instants, but also other variables occurring in φ. More precisely, all its ground instances are subject to evaluation. In the example below, the contextual A-ILTL rule states that the time-out for completion of a goal is established according to its priority.

$$FINALLY(T; F)\, G :: goal(G), priority(G, P), timeout(P, T), frequency(P, F)$$

A-ILTL rules with repairs During the monitoring process, each A-ILTL rule is attempted at a certain frequency and with certain priorities (possibly customizable by means of directives specified in $\mathcal{CI}$). If the current state of affairs satisfies every A-ILTL rule, then no action is required. Otherwise, some kind of repair action has to be undertaken with respect to the violated A-ILTL rule. To this aim, we extend the definition of contextual A-ILTL rules to specify a corresponding repair action.

Definition 8. *An A-ILTL rule with a repair is a rule the form:* $OP(M, N; K)\varphi :: \chi \div \psi$*, where:*

- $OP(M, N; K)\varphi :: \chi$ *is a contextual A-ILTL rule;*
- ψ *is called the* repair action *of the rule, and it consists of an atom* ψ*.*

Whenever the monitoring condition $OP(M, N; K)$ of an A-ILTL rule is violated, the repair action ψ is attempted. The repair action is specified via an atom that is executed as an ordinary goal. Consider again the previous example which monitors the achievement of goals, but extended to specify that, in case of violation, the present level of commitment of the agent to its objectives has to be increased. This is specified as:

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

$$incr\_comt(\,T) \leftarrow level(commitment, L), increase\_level(L, L1), \\ assert(inc\_comt\_at(\,T)), assert(neg(commitment\_mod(L)), \\ assert(commitment\_mod(L1))$$

Suppose that at a certain time-stamp t the monitoring condition that needs to be tested is $NEVER\,(not\,achieved(g), dropped(g)\,)$ is violated for some goal g. Upon detection of the violation, the system will attempt the repair action consisting in executing the goal $?{-}inc\_comt(t)$. In turn, its execution will allow the system to perform the specified run-time re-arrangement of the program that attempts to cope with the unwanted situation. Semantically, the execution of the repair action will determine the update of the current agent program $\mathcal{P}_i$, returning a new agent program $\mathcal{P}_{i+1}$.

3 Proposed Extensions

In this section we enrich the setting introduced in previous section. First, we further improve A-ILTL rules. Second, and this is in our view the main contribution of this paper, we introduce in A-ILTL rules the possibility of expressing and exploiting complex preferences. We "import" some of our previous results, but we also propose a significant extension in the direction of "modal" preferences.

3.1 A-ILTL rules with ordered conjunction

In an agent system, each event that occurs in the system can be time-stamped with the time-instant in which the event has occurred. We assume that this time-stamp can be made explicit in A-ILTL rules of the form specified above: each atom A_i occurring in φ can take the form $A_i : t_i$, where t_i is its time-stamp. The conjunction of atoms in φ can be usefully considered to be an ordered conjunction, where an event can be required to occur before another one, and the relationship between the respective time-stamp can be explicitly stated.

Definition 9. *An* ordered conjunction *is an expression of the form*

$$A_1 : t_1 << ... << A_n : t_n, L_1, ..., L_k$$

where the A_i's are atoms, the t_i's are constants and the L_i's are literals in which the t_i's possibly occur as arguments.

Definition 10. *An ordered A-ILTL rule is an A-ILTL rule where φ is an ordered conjunction.*

Below are some examples of the usefulness of the enhanced A-ILTL rules in the realm of Ambient Intelligence, and in particular of agents that perform user monitoring and training. The following rule expresses that a student has to enroll to a course before attending it and finally (s)he can give the exam. Here the time-stamps are not made explicit.

$$ALWAYS\ enroll(S, C) << attend(S, C) << exam(S, C, Grade),\\ student(S), course(C)$$

As a second example, we have a rule where there are two actions, namely $drink$ and $drive$. The rule states, by means of a simple constraint involving the time-stamps of the actions, that one cannot drive within one hour since when a drink has been taken.

$$NEVER\,(drink : T1) << (drive : T2), T1 - T2 < 60$$

Similarly, the rule below states that one should take a certain medicine before dinner, precisely between half-an-hour and an hour before.

$$ALWAYS\,(take\_medicine : T1) << (have\_dinner : T2),\\ T2 - T1 \geq 30, T2 - T1 \leq 60$$

3.2 Preferences

Preference is deeply related to an agent's personal view of the world, and it drives the actions that (s)he takes in it. In fact, preference has been studied in many disciplines, especially in philosophy and social sciences, but also in psychology, economics (when the need arises of formalizing some form of rational mental process that human beings activate during decision making, possibly in presence of uncertain knowledge), and, last but not least, in logic. The studies of the processes that support the construction or the elicitation of preferences have historically deep roots.

In logic, [26] initiated a line of research that was subsequently systematized in [35] which is usually taken to be the seminal work in preference logic. This line of research continues nowadays: the works of [34] and [29], for instance, develop new modal preference logics that improve over [26] in several directions. Preferences handling in computational logic has been extensively studied too. The reader may refer, e.g., to [22,6] for recent overviews and discussion of many existing approaches to preferences.

Some of the authors of this paper have proposed approaches to preferences in agents [15] or more generally in logic languages [16,17]. In particular, the approach defined in [16] allows for the specification of various kinds of non-trivial preferences. These

preferences follow the quite intuitive principles first formalized in [35], and illustrated at length, e.g., in [34]. 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. An advancement of our approach over others is that preferences, as illustrated below, have a local flavor. I.e., a preference holds in the context of the rule where it is defined, where different (even contrasting) preferences can be expressed (and simultaneously hold) in different contexts. 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 I prefer ψ to $(\phi \vee \zeta)$ then I 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 in order to illustrate the approach to preferences in logical agents and languages that we have developed in previous work [15,16,17]. 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 [35], 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*
$\{chocolate, nuts, coconut \mid less\_caloric\}$, 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.

$$icecream > zabaglione \leftarrow egg,\ sugar, (skimmilk > wholemilk \leftarrow diet), \{chocolate, nuts, coconut \mid less\_caloric\}.$$

$$less\_caloric(X, Y) \leftarrow calory(X, A), calory(Y, B), A < B.$$
$$calory(nuts, 2). \qquad calory(coconut, 3).$$

The above features can be smoothly incorporated in the present approach. In fact, the evolutionary semantics presented in [20] can easily accommodate this kind of preference reasoning. Below we propose a further extension inspired by the work of [29]. In particular, referring to agents, [29] 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. [29] 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:

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

In our setting, for defining and implementing the H operator we resort to the approach that we have introduced in [9]. There, 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 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 $A \in (M_1 \cap \ldots \cap M_k)$. Possibility and necessity can possibly be evaluated within a context, i.e., if $E(Args)$ 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 atom in $Context$ is added to Π as a new fact. 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:

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

Based on the above, we are able to define the H operator, explicitly stating which is the aspect that makes a world in which ϕ holds preferable. In fact, by $\phi : H(G)$ we mean that, assuming ϕ, we expect that G 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, G 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 possibility operator, taking ϕ as the context.

Definition 11. *Given an answer set program Π with answer sets $M_1,\ldots,M_k$, an atom ϕ and an atom G, the expression $\phi : H(G)$ is deemed to hold (w.r.t. Π) whenever the contextual possibility expression $P(G) : \phi$ holds.*

We can also extend the operator to a form $\phi : H[N](G)$ meaning that, given ϕ, we expect the hoped-for property G to hold in exactly N different possible worlds.

Definition 12. *Given an answer set program Π with answer sets $M_1,\ldots,M_k$, an atom ϕ and an atom G, the expression $\phi : H[n](G)$ is deemed to hold (w.r.t. Π) whenever there exist $\{v_1, \ldots, v_n\}$, $v_i \in \{1, \ldots, k\}$ such that $P(v_i, G) : \phi$ holds, $i \leq n$, and for every $P(w_i, G) : \phi$ which holds, $w_i \in \{v_1, \ldots, v_n\}$. By convention, we assume $\phi : H[0](G)$ to signify that $\phi : H[n](G)$ holds for no n.*

The answer set module T where to evaluate the operator H can possibly be explicitly specified, in the form $\phi : H(T, G)$ or, respectively, $\phi : H[N](T, G)$. Whenever T is not specified, we assume a unique underlying ASP module. The atom G occurring in the above definitions can be easily generalized to a conjunction of atoms. We are now able to introduce new preference expressions involving the operator H.

Definition 13. *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.*

Intuitively, we prefer A to B whenever, by assuming A, a situation where C holds can be possibly reached. We can extend the definition 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 if assuming A it is more plausible to reach C, i.e., C holds in more worlds than it is if assuming B. We prefer B over A otherwise.

Definition 14. *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 $N_A \geq N_B$. Otherwise, we get the p-list $B > A$.*

The extension to preference lists with more than two elements is straightforward. The "preference condition" C can be generalized to conjunctions. In an agent program, atoms A and B can have several meanings, for instance they can encode plans, e.g., in the form $plan(p, a_1, \ldots, a_n)$ where p is the identifier of a plan consisting of a sequence of actions $a_1, \ldots, a_n$. In this case, the operator H can evaluate plans w.r.t. some wished-for outcome.

We now provide some simple examples of use of the above-introduced preference expressions. For instance, one may choose to prefer a certain food rather than another one, in the hope that the preferred food is good for health:

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

A similar but stronger formulation can be the following, where one chooses to prefer a food that in most of the situations that can "reasonably" envisaged will procure better health, where in our setting these situations are interpreted as the answer sets of the underlying ASP module:

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

where one chooses the food that is presumably healthier (i.e., it can be concluded to be healthy in most cases).

It can be convenient to include more context in the H operator, by allowing for

$A > B : Context : H(T, C)$

or, respectively,

$A > B : Context : maxH(T, C)$

where $Context$ is a conjunction of atoms, each of which will be added to the ASP module as a new fact before evaluating H. Formally, it is simple to extend the definitions above so as to include $Context$ in the evaluation of the H and $maxH$ expressions. This extension allows for instance for the following variation of the above example:

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

Finally, it can 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 of foods are ordered according to the degree of healthiness, interpreted as the value of N in f, $diabetes \ : \ H[N]healthy$ for each food f.

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

Definition 15. *A* modal p-set *(mp-set) is an expression of the form:*

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

where p, q are predicates, $X_1, \ldots, X_n$ 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 p-list $A_1 > \ldots > A_s$ where: the A_is are all the possible ground atoms of the form $q(t_1, \ldots, t_n)$ where $p(t_1, \ldots, t_n)$ holds; for each A_j, A_k in this p-list, where $A_j = q(g_1, \ldots, g_n)$ and $A_k = q(h_1, \ldots, h_n)$,$A_j$ precedes (is preferred to) A_k iff the following expression holds:

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

The expressions shown above allows for the definition of a variety of useful meta-statements. For instance the following A-ILTL rule states that the agent, having a deadline, always prefers to pursue the goal (adopt the intention) for which there exists a plan such that the goal can with the highest possible confidence be reached within the deadline.

$$ALWAYS \ adopt\_intention(G, P), \\ \{plan(G, P), \ goal(G) \ : \ deadline(t) \ : \ H(reached(G, P, t))\}$$

4 Future Work

There are several future directions for the ideas that we have discussed. Up to now, most of the proposed features have been simulated in DALI by means of the "Internal Events" construct. We have also implemented in DALI the ASP modules and the related operators. However, we intend to fully implement an instance of the proposed framework, starting from EVOLP, DALI and KGP agents (which are fully-defined and fully-implemented approaches) that provide the main elements and can be exploited in combination in an implementation. Then, we intend to experiment our setting on practical cases. Next, we aim at exploring a generalization of our setting to the multi-agent

case, allowing A-ILTL and preference expressions to be written that involve several agents, or even the "society" (talking for instance of goals that the "society" should hopefully reach). In this direction, we intend to design a meta-meta level for controlling knowledge exchange. Particular attention should be dedicated to strategies involving reputation and trust for the evaluation of exchanged knowledge.

References

1. J. J. Alferes, A. Brogi, J. A. Leite, and L. M. Pereira. Evolving logic programs. In *Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002*, LNAI 2424, pages 50–61. Springer-Verlag, Berlin, 2002.
2. K. R. Apt and R. Bol. Logic programming and negation: A survey. *The Journal of Logic Programming*, 19-20:9–71, 1994.
3. J. Barklund, P. Dell'Acqua, S. Costantini, and G. A. Lanzarone. Reflection principles in computational logic. *J. of Logic and Computation*, 10(6):743–786, 2000.
4. M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.
5. A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, G. Terreni, and F. Toni. The KGP model of agency: Computational model and prototype implementation. In *Global Computing: IST/FET International Workshop, Revised Selected Papers*, LNAI 3267, pages 340–367. Springer-Verlag, Berlin, 2005.
6. G. Brewka, I. Niemelä, and M. Truszczyński. Preferences and nonmonotonic reasoning. *AI Magazine*, 29(4), 2008.
7. A. Costa and G. Dimuro. Towards a domain-theoretic model of developmental machines. In *Computation and Logic in the Real World, Proceedings of the Third Conference on Computability in Europe, CiE 2007*.
8. A. Costa and G. Dimuro. On the notion of developmental computing machine. In *Anais do XXXIV Seminario Integrado de Software e Hardware, Rio de Janeiro*, 2007.
9. S. Costantini. Answer set modules for logical agents. In G. Gottlob, editor, *Datalog 2.0*, LNCS. Springer, 2011. Forthcoming.
10. S. Costantini, S. D'Alessandro, D. Lanti, and A. Tocchio. Dali web site, download of the interpreter, 2010. http://www.di.univaq.it/stefcost/Sito-Web-DALI/WEB-DALI/index.php.
11. S. Costantini, P. Dell'Acqua, and L. M. Pereira. A multi-layer framework for evolving and learning agents. In A. R. M. T. Cox, editor, *Proceedings of Metareasoning: Thinking about thinking workshop at AAAI 2008, Chicago, USA*, 2008.
12. S. Costantini, P. Dell'Acqua, and L. M. Pereira. Conditional learning of rules and plans by knowledge exchange in logical agents. In *RuleML 2011, 5th International Symposium on Rules, in conjunction with IJCAI*, 2011.
13. S. Costantini, P. Dell'Acqua, L. M. Pereira, and F. Toni. Towards a model of evolving agents for ambient intelligence. In *Proc. of the Symposium on "Artificial Societies for Ambient Intelligence (ASAmI'07)*, 2007.
14. S. Costantini, P. Dell'Acqua, L. M. Pereira, and F. Toni. Learning and evolving agents in user monitoring and training. In *Proc. of the AICA 2010 Italian Conference*, 2010. held in L'Aquila.
15. S. Costantini, P. Dell'Acqua, and A. Tocchio. Expressing preferences declaratively in logic-based agent languages. In *Proc. of Commonsense'07, the 8th International Symposium on Logical Formalizations of Commonsense Reasoning*, AAAI Spring Symposium Series, 2007.
16. S. Costantini and A. Formisano. Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of of Algorithms in Cognition, Informatics and Logic*, 64(1), 2009.

17. S. Costantini and A. Formisano. 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.
18. S. Costantini and A. Tocchio. A logic programming language for multi-agent systems. In *Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf.,JELIA 2002*, LNAI 2424. Springer-Verlag, Berlin, 2002.
19. S. Costantini and A. Tocchio. The DALI logic programming agent-oriented language. In *Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004*, LNAI 3229. Springer-Verlag, Berlin, 2004.
20. S. Costantini and A. Tocchio. About declarative semantics of logic-based agent languages. In M. Baldoni and P. Torroni, editors, *Declarative Agent Languages and Technologies*, LNAI 3229. Springer-Verlag, Berlin, 2006.
21. S. Costantini, A. Tocchio, F. Toni, and P. Tsintza. A multi-layered general agent model. In *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, 10th Congress of the Italian Association for Artificial Intelligence*, LNCS 4733. Springer-Verlag, Berlin, 2007.
22. J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 20(12):308–334, 2004.
23. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*. MIT Press, 1990.
24. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88)*, pages 1070–1080. The MIT Press, 1988.
25. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
26. S. Hallden. *On the logic of better*. Library of Theoria, No. 2, Lund: Library of Theoria. Cambridge University Press, 1957.
27. A. C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *Proc. ECAI-2004*, 2004.
28. O. Lichtenstein, A. Pnueli, and L. Zuch. The glory of the past. In *Proc. Conf. on Logics of Programs*, LNCS 193. Springer Verlag, 1985.
29. F. Liu. Von wrights the logic of preference revisited. *Synthese*, 175(1):69–88, 2009.
30. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
31. V. W. Marek and M. Truszczyński. Stable logic programming - An alternative logic programming paradigm. In *25 years of Logic Programming Paradigm*, pages 375–398. Springer, 1999.
32. K. Stathis and F. Toni. Ambient Intelligence using KGP Agents. In P. Markopoulos, B. Eggen, E. H. L. Aarts, and J. L. Crowley, editors, *Proceedings of the 2nd European Symposium for Ambient Intelligence (EUSAI 2004)*, LNCS 3295, pages 351–362. Springer Verlag, 2004.
33. A. Tocchio. *Multi-Agent systems in computational logic*. PhD thesis, Dipartimento di Informatica, Università degli Studi di L'Aquila, 2005.
34. J. van Benthem, P. Girard, and O. Roy. Everything else being equal: A modal logic for ceteris paribus preferences. *J. Philos. Logic*, 38:83–125, 2009.
35. G. H. von Wright. *The logic of preference*. Edinburgh University Press, 1963.