Expressing preferences declaratively in logic-based agent languages

Stefania Costantini
Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito,
I-67010 L'Aquila - Italy
Email: stefcost@di.univaq.it

Pierangelo Dell'Acqua
Department of Science and Technology - ITN
Linköping University
601 74 Norrköping, Sweden
Email: pier@itn.liu.se

Arianna Tocchio
Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito,
I-67010 L'Aquila - Italy
Email: tocchio@di.univaq.it

***Abstract*— In this paper we present an approach to introducing preferences among actions in logic-based agent-oriented languages. These preferences are expressed in the body of rules (i.e., they are local to the rule where they are defined). To the best of our knowledge, no similar approach has been proposed before, and cannot be easily simulated by means of preferences expressed in the head of rules, which are global. The approach is applied to choosing which action an agent should perform in reaction to an event, among the feasible ones.**

I. Introduction

Intelligent agents perform advanced activities such as negotiation, bargaining, etc. where they have to choose among alternatives. The choice will be based on some kind of preference or priorities related for instance to:

- the agent's objectives;
- the context (cooperative vs. competitive);
- available resources;
- the strategies that the agent intends to follow.

Agents will in general include specialized modules and/or meta-level axioms for applying priorities and preferences, like for instance those proposed in [9] for prioritized defeasible reasoning. However, it can be useful in logical agents to be able to express preferences at a more basic linguistic level. These basic preferences can then be employed in building more advanced high-level strategies. At the language level, preferences have already been expressed in various way in Answer Set Programming [8] [12]. In that context, the basic mechanism is that of computing the Answer Sets and then chose "preferred" ones. We will shortly review below the work of Brewka on LPODS (Logic Programs with Ordered Disjunction) [2] and the work of Sakama and Inue on PLP (Prioritized Logic Programming) [13]. The reader may refer to the latter paper and to [6] for a discussion of relevant existing approaches. Some of them are based on establishing priorities/preferences among atoms (facts), and typically introduce some form of disjunction in the head of rules. Other approaches express instead priorities among rules.

Our proposal is aimed at allowing an agent to express preferences concerning either which action they would perform in a given situation, or, in perspective, which goal they would pursue in a certain stage. Since actions are often performed in reaction to events and goals are set in accordance to some internal conclusion that has been reached, we propose to introduce disjunction in the body of rules. If the body of a rule contains a disjunction among two or more actions, the *preferred* one will be chosen according to *preference rules*, that may have a body in that (following [13]) priorities may be conditional. If agents evolve with time and enlarge their knowledge and experience, priorities may dynamically change according to the agent evolution.

In agent languages that are not based on Answer Set Programming, one cannot select the preferred model(s) by means of a filter on possible models. Then, other techniques are needed in order to provide a semantic account to this proposal. Recently, an approach to declarative semantics of logical agent-oriented languages that considers evolution of agents has appeared in the literature [5]: changes that occur either externally (i.e., reception of exogenous events) or internally (i.e., courses of actions undertaken based on internal conditions) are considered as making a change in the agent program, which is a logical theory, and in its semantics (however defined). For such a change to be represented, it is understood as the application of a program-transformation function. Thus, agent evolution is seen as program evolution, with a corresponding semantic evolution.

This semantic approach can be applied to the present setting by adapting the proposal of the *split programs* introduced in [2]. A split program is a version of the given program obtained by replacing each disjunction by one of its options. Then, at each step we would have a set of *possible evolutions*, each corresponding to a split program. Among them, the preferred one (according to the present conditions) is taken, while all the others are pruned. As mentioned before, given similar situations in different stages of the agent life, different options can be taken, according to the present assessment of the agent knowledge.

Though simple, this mechanism is to the best of our knowledge new, as no similar approach has been proposed before, and it cannot be easily simulated by existing ones.

In Section II we review some features that intelligent logical agents should in our opinion possess, and the related usefulness of introducing preferences. In Section III we briefly review previous related work on preferences. In Section V we introduce the approach, and in Section VI its semantics. Finally, we conclude in Section VII.

II. Enhancing capabilities of Logical Agents by introducing Preferences

A great deal can be said about features that agents in general and logical agents in particular should possess (for a review the reader may refer for instance to [14], for a discussion to [11]). It is widely recognized however that agents, whatever the language and the approach on which they are based, should be able to cope with a changing and partially known environment. In this environment, agents should be able to interact, when they deem it appropriate, with other agents or with the user in order to complete their own problem solving and to help others with their activities.

Interacting agents may act according to suitable strategies, which include expressing preferences and establishing priorities, possibly with the aid of past experiences. In our view, complex strategies can take profit of basic linguistic constructs reminiscent of those introduced in Answer Set Programming.

Our proposal is aimed at allowing an agent to express preferences/priorities (in the following, we will often interchange the two terms) concerning either which action they would perform in a given situation, or also, in perspective, which goal they would pursue in a certain stage. Since actions are often performed in reaction to events and goals are set in accordance to some internal conclusion that has been reached, we propose to introduce disjunction in the body of rules. If the body of a rule contains a disjunction among two or more actions, the *preferred* one is chosen according to *preference rules*, that may have a body. I.e., following [13], priorities may be conditional. Also, preference rules may contain references to the agent past experience, and then the preferred choice may change over time. More precisely, whenever the body of a rule contains a disjunction among two or more actions, the intended meaning is the following:

- preference rules establish which action is preferred;
- precondition of the action state whether it can be actually performed, i.e., if it is feasible;
- the agent should perform the best preferred feasible action.

III. Previous Related Work

The reader may refer to [6] for a discussion of many existing approaches to preferences. The main distinction is among those that define priorities/preferences among atoms (facts), and typically introduce some form of disjunction in the head of rules, and those that express instead priorities among rules. Among the latter ones, we mention [10] that applies preferences among rules in negotiating agents based on argumentation, so as to tune argumentations according to changing contexts.

The approach of [13] considers *general extended disjunctive programs* where a rule has the syntax:

$$L_1 | \ldots | L_k | not\, L_{k+1} | \ldots | not\, L_{k+h} \leftarrow Body$$

where "|" represents disjunction and not is negation as failure under the Answer Set semantics. A preference, or priority, between two ground literals e_1, e_2 is expressed in the form $e_1 \prec e_2$. An answer set S_2 of a given program is preferable onto another answer set S_1 iff $S_2 \setminus S_1$ contains an element e_2 whose priority is higher than some element e_1 in $S_1 \setminus S_2$, and the latter does not contain another element e_3 whose priority is strictly higher that e_2. Then, preferred answer sets (or p-answer sets) are a subset of the traditional ones, that can be seen as a special case corresponding to empty priorities.

Basic PLP is exploited in [13] so as to express priorities not only between plain facts, but also between more general forms of knowledge. The approach allows many forms of commonsense reasoning to be modeled.

An interesting application is that of *priority with preconditions*. For instance, borrowing the example from [13], the situation where a person drinks tea or coffee but she prefers coffee to tea when sleepy can be represented as follows (in a prolog-like syntax):

$$tea \mid coffee.$$
$$tea \prec coffee \text{ :- } sleepy.$$

This program can be translated in a standard way in plain PLP and, assuming that *sleepy* holds, has the p-answer set $\{sleepy, coffee\}$.

In LPODS [2], one can write expressions such as $A \times B$ in the head of rules, where the new connective $\times$ stands for ordered disjunction. The expression intuitively stands for: if possible A, but if A is impossible then (at least) B. If there are several disjuncts, the first one represents the best preferred option, the second one represents the second best option, etc. The following is an example where a person who wishes to spend the evening out and has money prefers to go to theatre, or else (if impossible) to go to the cinema, or else (if both previous options cannot be taken) to go to dine at a restaurant.

$$theatre \times cinema \times restaurant \text{ :-}$$
$$want\_to\_go\_out, have\_money.$$

For selecting the preferred answer set(s) of a program P, one obtains the possible split programs of P, where a split program P' is obtained from P by replacing each disjunctive rule by one of its options. Then, the answer sets of P are taken to be the answer sets of the split programs. To choose preferred ones given that there may be several disjunctions, a notion of *degree of satisfaction* of disjunctive rules must be defined, that

induces a partial ordering on answer sets. Preferred answer sets are those that satisfy all rules of P to the better degree.

IV. Comparison

To the best of our knowledge, the approach of introducing preferences in the body of logical rules is novel, and has never appeared in the literature. It cannot be easily simulated by using preferences in the head: in fact, preferences expressed in the body are *local* to the rule where they occur, while preferences defined in the head are *global*. The application to agents performing actions is also new. 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 in different stages of its life.

V. The approach in more detail

We will now introduce a simple though in our opinion effective construct that can be employed in agent-oriented logic languages based on logic (horn-clause) programming. Similarly to [13], we assume the following:

- preferences are expressed between two ground facts;
- preferences are expressed explicitly by means of special rules, that may have conditions;
- preference is transitive, irreflexive and anti-symmetric.

In our approach, preferences can be defined between *actions* that agents may perform. We make some preliminary assumption about the agent languages we are considering. We do not commit to any particular syntax, though we will propose a sample one in order to introduce and illustrate examples. We will discuss the semantics of the class of languages that we consider in Section VI. By saying "an agent" we mean a program written in the language at hand, that behaves as an agent when it is put at work. We assume in particular the following syntactic and operational features.

- The agent is able to perceive external events coming from the environment where the agent is situated. In our sample syntax an external event is an atom which is distinguished by postfix E. E.g., $rainE$ indicates an external event.
- The agent is able to react to external events, i.e., the language provides some kind of condition-action construct. In our sample syntax ee indicate reaction by means of the connective $:>$. Then, a reactive rule will be indicated with $pE :> Body$ meaning that whenever the external event pE is perceived, the agent will execute $Body$. There are languages (like, e.g., the one presented in [3]) where an agent can react to its own internal conclusions, that are interpreted as events (thus modeling proactivity). We assume that the syntax for reaction is the same in both cases. However, an internally generated event is indicated with postifix I, i.e., in the form pI.
- The agent is able to perform actions. Actions will occur in the agent program as special atoms. In our sample syntax we assume them to be in the form qA, i.e., they are distinguished by suffix A. E.g., $open\_umbrellaA$ indicates an action. Actions may have preconditions: In our sample syntax we assume them to be expressed by rules. The connective $:<$ indicates that the 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.

In the proposed approach, a disjunction (indicated with "|") of actions may occur in the body of a reactive rule. Preferences among actions are defined in *preference rules*, that are indicated by the new connective $<<$. Then, a rule $pE :> q1A \mid q2A$ means that in reaction to pE the agent may perform either action $q1A$ or action $q2A$. A rule $q1A << q2A$:- $Body$ means that action $q2A$ is preferred over action $q1A$ provided that $Body$ is true. I.e., if $Body$ is not true the preference is not applicable, and then any of the actions can be indifferently executed. A set of preference rules define in general 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 5.1: Consider a person who receives an invitation to go out. 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 money and 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. In our sample syntax, an agent program fragment formalizing this situation may look as follows.

$$invitationE :> acceptA \mid rejectA.$$
$$acceptA :< have\_money, have\_time.$$
$$refuseA << acceptA \text{ :- } nice\_people\_inviting.$$

When the external event $invitationE$ is perceived by the agent, it can react by alternatively performing one of two actions. The action $acceptA$ will be performed if its preconditions are verified. As preferences are among feasible actions, $acceptA$ is preferred provided that $nice\_people\_inviting$ holds. Notice that this 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 (or, another future invitation) arrives when there are no resources for accepting, the agent will refuse the invitation.

Another example will introduce further aspects.

Example 5.2: Let us now rephrase the example of the person preferring coffee over tea if sleepy. Let us put it in

a proactive perspective, where the person wonders whether it is time to take a break from working, e.g., at mid-afternoon. If so, she will consider whether to drink tea or coffee. The corresponding program fragment might look as follows, where $take\_break$ is an *internal conclusion* that triggers a proactive behavior: the first rule reaches the conclusion that taking a break is in order; the second rule states what to do then, i.e., specifies a reaction to the internal conclusion itself (indicated in the second rule with postfix I for "internal"). For the mechanism to be effective, $take\_break$ must be attempted from time to time, so as to trigger the consequent behavior as soon as it becomes true.

$$take\_break \text{ :- } five\_oclock.$$
$$take\_breakI :> drink\_teaA \mid drink\_coffeeA.$$
$$drink\_coffeeA :< espresso.$$
$$drink\_teaA << drink\_coffeeA \text{ :- } sleepy.$$

Again, what the agent will do depends upon the present conditions, i.e., upon whether the agent feels sleepy or not. Moreover, in this variation the agent drinks coffee only if she can have an espresso.

Assume now that there is also the option of drinking juice, though the agent will only drink orange juice, and that the agent prefers juice to tea. Then the program becomes:

$$take\_break \text{ :- } five\_oclock.$$
$$take\_breakI :> drink\_teaA \mid drink\_coffeeA \mid drink\_juiceA.$$
$$drink\_coffeeA :< espresso.$$
$$drink\_juiceA :< orange.$$
$$drink\_teaA << drink\_coffeeA \text{ :- } sleepy.$$
$$drink\_teaA << drink\_juiceA.$$

The expected behavior is the following:

- If $sleepy$ holds and $espresso$ holds as well, the agent can drink coffee (the action $drink\_coffeeA$ is allowed) and will not drink tea, which is less preferred. If $orange$ does not hold, the agent will definitely drink coffee.
- If $sleepy$ holds and $espresso$ holds as well, the agent can drink coffee (the action $drink\_coffeeA$ is allowed) and will not drink tea, which is less preferred. If $orange$ holds, also the action $drink\_juiceA$ is allowed, and preferred over $drink\_teaA$. The agent can indifferently drink either coffee or juice, as they are unrelated.
- If $espresso$ does not hold, the agent cannot drink coffee (the action $drink\_coffeeA$ is not allowed). Then, if $orange$ holds then the agent will drink juice (the action $drink\_juiceA$ will be performed), otherwise it will drink tea (as the action $drink\_teaA$ is always allowed, not having preconditions).
- If $sleepy$ does not hold, there is no preference between tea and coffee. If $orange$ does not hold and $espresso$ holds, one of the two actions $drink\_teaA$ or $drink\_coffeeA$ can be indifferently executed. If $orange$ holds and $espresso$ holds as well, $drink\_juiceA$ is preferred over $drink\_teaA$, but as no other priority is specified, one of the actions $drink\_coffeeA$ or $drink\_juiceA$ can be indifferently executed.

VI. Declarative Semantics of Evolving Agents with Preferences

The evolutionary semantics that has been proposed in [5] has the objective of providing a unifying framework for various languages and semantics for reactive and proactive logical agents.

This semantic approach is based upon declaratively modeling the changes inside an agent which are determined both by changes in the environment and by the agent's own self-modifications. The key idea is to understand these changes as the result of the application of program-transformation functions. In this view, a program-transformation function is applied for instance upon reception of either an external or an internal event, the latter having a possibly different meaning in different formalisms. That is, perception of an event can be understood as having an effect on the program which defines the agent: for instance, the event can be stored as a new fact in the program. Similarly, actions which are performed can be recorded as new facts. All the "past" events and actions will constitute the "experience" of the agent.

Recording each event or action or any other change that occurs inside an agent can be semantically interpreted as transforming the agent program into a new program, that may procedurally behave differently than before: e.g., by possibly reacting to the event, or drawing conclusions from past experience. Or also, the internal event corresponding to the decision of the agent to undertake an activity triggers a more complex program transformation, resulting in version of the program where the corresponding *intention* is somewhat "loaded" so as to become executable.

Then, in general one will have an initial program P_0 which, according to these program-transformation steps (each one transforming P_i into P_{i+1}), gives rise to a Program Evolution Sequence $PE = [P_0, ..., P_n]$. The program evolution sequence will have a corresponding Semantic Evolution Sequence $ME = [M_0, ..., M_n]$ where M_i is the semantic account of P_i according to the specific language and the chosen semantics. The couple $\langle PE; ME \rangle$ is called the *Evolutionary Semantics* of the agent program P_{Ag}, corresponding to the particular sequence of changes that has happened, and to the order in which they have been considered. The evolutionary semantics of an agent represents the history of an agent without introducing a concept of a "state".

The different languages and different formalisms will influence the following key points:

1) When a transition from P_i to P_{i+1} takes place, i.e., which are the external and/or internal factors that determine a change in the agent.
2) Which kind of transformations are performed.
3) Which semantic approach is adopted, i.e., how M_i is obtained from P_i. M_i might be for instance a model,

or an initial algebra, or a set of Answer Sets if the given language is based on Answer Set Programming (that comes from the stable model semantics of [8]). In general, given a semantics $\mathcal{S}$ we will have $M_i = \mathcal{S}(P_i)$.

A transition from P_i to P_{i+1} can reasonably take place, for instance:

- When an event happens.
- When an action is performed.
- When a new goal is set.
- Upon reception of new knowledge from other agents.
- In consequence to the decision to accept/reject the new knowledge.
- In consequence to the agent decision to revise its own knowledge.

We say that at stage P_{i+1} of the evolution the agent *has perceived* event ev (whatever its class) meaning that the transition from P_i to P_{i+1} has taken place in consequence of reception of ev. It is reasonable to assume that in the stage P_{i+1} the agent will cope with ev, e.g., by reacting to it if it is an external event.

Example 6.1: It is useful to discuss how the program transformation step related to actions might be formalized. Intuitively, an action atom (like e.g. $drink\_coffeeA$ in a previous example) should become true given its preconditions, if any ($espresso$ in the example) whenever the action is actually performed in some rule. For the sake of simplicity assume that (like in the examples presented above) actions can occur only in the body of reactive rules.

Declaratively, this means that the action occurs in the body of an applicable reactive rule. Practically, whenever that rule will be processed by the interpreter because the corresponding (external or internal) event has happened, the action will be actually performed (by means of any kind of mechanism that connects the agent to its environment). To account for this behavior, in the initialization step each rule defining preconditions for actions, say of the form

$actA :< C_1, \ldots, C_s$

is transformed into a set of rules of the form:

$actA$:- $D_1, \ldots, D_h, C_1, \ldots, C_s$

where $D_1, \ldots, D_h$, $h \geq 0$ are the conditions (except for other actions) of each reactive rule where $actA$ occurs in the body. The C_i's are omitted if $actA$ has no preconditions.

Whenever at some stage P_i of the program evolution $actA$ will be attempted and feasible as its preconditions are true, we will have $actA \in M_i$.

It can be useful in general to perform an *Initialization step*, where the program P_{Ag}, written by the programmer, is transformed into a corresponding initial program P_0 by means of some sort of knowledge compilation. This initialization step can be understood as a rewriting of the program in an intermediate language and/or as the loading of a "virtual machine" that supports language features. This stage can on one extreme do nothing, on the other extreme it can perform complex transformations by producing "code" that implements language features in the underlying logical formalism. P_0 can be simply a program (logical theory) or can have additional information associated to it.

This semantic approach can be extended so as to encompass the present proposal. As a first point, in the initialization step preferences must be collected and preference rules removed. Then, P_0 will not contain preference rules, but will be associated to a structure $Pref$ where preferences between couples of (ground) actions are made explicit, by performing the transitive closure of preference rules. The conditions of a preference rule (if any) are added as preconditions of the preferred action.

We adapt the idea of *split program* from [2]. A split program is a version of the given program obtained by replacing a disjunction by one of its options. In our case, whenever an agent at stage P_i of its evolution has perceived an (either external or internal) event, say pE, it will react to it. However, if there is a disjunction of actions in the body of the corresponding reactive rule, then the agent may react in more that one way. The different ways of reacting are represented by different *split programs*, each one representing an alternative. Precisely,

Definition 1: Let P_{Ag} be an agent program that has been transformed into a program P_0 by the initialization step. Let P_i be the program obtained from the evolution of P_0 at the i-th step, corresponding to the perception of event pE. Let $pE :> Body$ be the corresponding reactive rule in P_i, where a disjunction of actions occurs in $Body$. A *split program* P_i' is obtained by replacing the disjunction with one of its options.

Referring to the program of Example 5.1, at the initialization step it is transformed into:

$$invitationE :> acceptA \mid rejectA.$$
$$acceptA :< have\_money, have\_time, nice\_people\_inviting.$$

where the preference $refuseA << acceptA$ is recorded in the structure $Pref$. Then, whenever the event $invitationE$ will be perceived will be two split programs: a first one, say ϕ_1, where the body of the reactive rule contains only $acceptA$, and a second one, say ϕ_2, where the body of the reactive rule contains only $refuseA$.

We will have a set $\{P_i^1, \ldots, P_i^k\}$ of split programs corresponding to the number k of actions occurring in the disjunction. Assuming that events are considered one at a time (i.e., an evolution step copes with a single event), at each stage split programs will be relative to a single reactive rule, and will correspond to a set $\{M_i^1, \ldots, M_i^k\}$ where M_i^j is the semantics of P_i^j. We say that we *a split occurs* at stage P_i of program evolution whenever at that stage the incoming event is related to a reactive rule with a disjunction of actions in its body.

The preferred split programs are those whose semantics contain the preferred actions. Precisely:

Definition 2: Let P_{Ag} be an agent program that has been transformed into a program P_0 by the initialization step, and let $Pref$ be the preference structure that has been associated

to the program. Let P_i correspond to a step of the evolution of P_0, where a split occurs. Given two split programs P_i^r and P_i^s obtained from P_i by splitting a disjunction $act^1 A \mid \ldots \mid act^k A$, then P_i^r is preferred over P_i^s if the following conditions hold:

- the semantics M_i^r of P_i^r contains $act^r A_i$;
- $act^r A_i$ is preferred over $act^s A_i$ according to $Pref$.

Notice that both M_i^r and M_i^s may not contain the corresponding action ($act^r A_i$ and $act^s A_i$ respectively), in case its preconditions are false. Then, a split program is preferred upon another one if (i) its semantics entails the related action and (ii) either the semantics of the other one does not entail the related action, or the former action is preferred.

Then, at each step where a split occurs we have a set of *possible evolutions*, each corresponding to a split program. Among them, the preferred one (according to the present conditions) is taken, while all the others are pruned. As mentioned before, given similar situations at different stages of the agent life, different options can be taken, according to the present assessment of the agent knowledge. In the example, ϕ_1 will be preferred to Φ_2 whenever it actually entails $acceptA$.

We can have a unique best preferred split program P_i^b if $Pref$ is a total order with respect to the actions over which we split, or we may have more than one equally preferred split programs. Any of them can be indifferently selected.

Definition 3: Let P_i be a stage of the program evolution sequence, where a split occurs. We let P_{i+1} be any of the best preferred split programs.

VII. Concluding Remarks

In this paper we have presented an approach to expressing preferences among actions and in logical agents. The approach builds on previous relevant work related to answer set programming, but is rephrased for reactive and proactive agents that evolve in time. In fact, the semantics of the approach is given in *evolutionary* terms, where an agent program is considered to be modified by events that happen and actions that are performed, while its semantic account evolve correspondingly.

There are others approaches in computational logic that are related to the present one, and to which we are indebted, namely [1] and [7], where preferences and updating preferences are coped with in the context of a more general approach to updating logic programs. We may notice that the examples that we have presented basically refer to the syntax and procedural semantics of the DALI language [3], [4], [14]. Actually they correspond to working DALI programs, though the implementation is prototypical and is being experimented.

Our next research objective is to extend the possibility of expressing preferences to all kinds of subgoals occurring in the body of logical rules, instead of coping with actions only.

Another important objective is to extend the approach so as to be able to express preferences among agent goals (objectives to reach). In fact, the reasoning can be similar. Actually, setting an objective is related to building a plan for achieving it. A plan however can be seen as divided into:

1) a preliminary check stage, where feasibility of subsequent actions is checked (are the tickets available? Do I have the money? Do my friends accept to join me? May I rent a car?);
2) an operative stage, where actions that influence the environment (and in general cannot be retracted, or at least not so easily) are performed.

The first stage can be seen as a feasibility stage for setting an objective. Then, if there is a disjunction of objectives in the body of a rule, we mean that the agent should set the most preferred feasible one.

References

[1] J. J. Alferes, P. Dell'Acqua and L. M. Pereira, *A compilation of updates plus preferences*. Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424, Springer-Verlag, 2002, pp. 62-74.

[2] G. Brewka, *Logic programming with ordered disjunction*, In Proc. of AAAI-02, Edmonton, Canada, 2002.

[3] S. Costantini and A. Tocchio, *A logic programming language for multi-agent systems*. Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424, Springer-Verlag, 2002.

[4] S. Costantini, A. Tocchio, *The DALI logic programming agent-oriented language*. Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004, Lisbon, September 2004. LNAI 3229, Springer-Verlag, Germany, 2004.

[5] S. Costantini, A. Tocchio, *About declarative semantics of logic-based agent languages*. In M. Baldoni and P. Torroni (eds.), Declarative Agent Languages and Technologies, Post-Proc. of DALT 2005. LNAI 3229, Springer-Verlag, Germany, 2006.

[6] J. Delgrande, T. Schaub, H. Tompits and K. Wang, *A classification and survey of preference handling approaches in nonmonotonic reasoning*. Computational Intelligence, 2004.

[7] P. Dell'Acqua and L. M. Pereira, *Preferring and updating in logic-based agents*. In: Web-Knowledge Management and Decision Support. Selected Papers from the 14th Int. Conf. on Applications of Prolog (INAP), LNAI 2543, Springer-Verlag, Berlin, 2003, pp. 70-85.

[8] M. Gelfond and V. Lifschitz, *The stable model semantics for logic programming.* Proc. of 5th ILPS conference, 1988, pp. 1070-1080.

[9] M. Gelfond and T. C. Son, *Reasoning with prioritized defaults*. In Proc. of the 3rd Int. Works. on Logic Programming and Knowledge Representation, LNAI 1471, Springer-Verlag, Berlin, 1998, pp. 164-223.

[10] A. Kakas and P. Moraitis, *Adaptive agent negotiation via argumentation*. In Proc. of the 5th International Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'06), Hakodate, Japan, 2006.

[11] R. A. Kowalski, *The logical way to be artificially intelligent*, *Computational Logic in Multi-Agent Systems* (CLIMA VI Post-Proceedings), LNAI 3900, Springer-Verlag, Berlin, 2005, pp. 1-22.

[12] V. Lifschitz, *Answer set planning.* Invited talk. Proc. of ICLP '99 Conf., pp. 23–37. The MIT Press, 1999.

[13] C. Sakama and K. Inoue, *Prioritized logic programming and its application to commonsense reasoning*. Artif. Int. 123(1-2), Elsevier, 2000, pp. 185-222.

[14] A. Tocchio, *Multi-Agent systems in computational logic*, Ph.D. Thesis, Dipartimento di Informatica, Universitá degli Studi di L'Aquila, 2005.