Preferring and Updating in Multi-Agents

Pierangelo Dell'Acqua
Department of Science and Technology
Campus Norrköping, Linköping University
Norrköping, Sweden
pier@itn.liu.se

Luís Moniz Pereira
Centro de Inteligência Artificial - CENTRIA
Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract

We present a logical framework and the declarative semantics of a multi-agent system in which each agent can communicate with and update other agents, can update its knowledge state, of its own initiative or when it receives new incoming information, can react to the environment, and it is able to prefer beliefs and reactions, when several alternatives are possible. The knowledge state of an agent is represented by an updatable prioritized logic program, in which priorities among rules can be expressed to allow the agent to prefer, and where the preference relation itself can be updated as well. We exhibit examples to show how our approach functions, including how preferring can enhance choice of reactivity in agents, and refer to the work of others.

1 Introduction

In a previous paper [7] we presented a combination of the dynamic logic programming paradigm proposed by J. J. Alferes et al. [1, 12] and (a version of) KS-agents proposed by Kowalski and Sadri [9]. In the resulting framework, rational, reactive agents can dynamically change their own knowledge bases as well as their own goals. In particular, at every iteration of an observe-think-act cycle, the agent can make observations, learn new facts and new rules from the environment, and then it can update its knowledge accordingly. The agent can also receive a piece of information that contrasts with its knowledge. To solve eventual cases of contradiction within the theory of an agent, techniques of contradiction removal and preferences among several sources can be adopted [10]. The actions of an agent are modeled by means of updates, inspired by the approach in [3].

A semantic characterization of updates is given in [1] as a generalization of the stable model semantics of normal logic programs [8]. Such a semantics is generalized to the three-valued case in [3], which enable us to update programs under the well-founded semantics.

J. J. Alferes and L. M. Pereira [2] propose a logic programming framework that combines two distinct forms of reasoning: preferring and updating. In particular, they define a language capable of considering sequences of logic programs that result from the consecutive updates of an initial program, where it is possible to define a priority relation among the rules of all successive programs. Updating and preferring complement each other in the sense that updates create new models, while preferences allow us to select among pre-existing models. Moreover, within the framework, the priority relation can itself be updated. In [2] the authors also define a declarative semantics for such a language.

In this paper we set forth an extension of the language introduced in [7] for updating agents to include preferences among the rules of an agent. In the resulting framework agents can, not only update their own knowledge, for example on advice from some other agent, but can also express preferences about their own rules. Preferences will allow an agent to select among alternative models of his knowledge base. In our framework, preferences can also be updated, possibly on advice from others. We provide the declarative semantics of the multi-agent system, and we characterize the beliefs of an agent in terms of preferred stable models. The following example (taken from [2]), where rules as well as preferences change over time, shows the need to combine preferences and updates, including the updating of preferences themselves.

Example [A sad story] (1) In the initial situation Maria is living and working everyday in the city. (2) Next, as Maria have received some money, Maria conjures up other, alternative but more costly, living scenarios, namely traveling, settling up on a mountain, or living by the beach. And, to go with these, also the attending preferences, but still in keeping with the work context, namely that the city is better for that purpose than any of the new scenarios, which are otherwise incomparable amongst themselves. (3) Consequently, Maria decides to quit working and go on vacation, supported by her increased wealth, and hence to define her vacation priorities. To wit, the mountain and the beach are each preferable to travel, which in turn gainsays the city. (4) Next, Maria realizes her preferences keep her all the while undecided between the mountain and the beach, and advised by a friend opts for the former. (5) Forthwith, Maria ventures up the mountain, only to become ill on account of the height, and a physician advises her against too much sun exposure, be it at the mountain or the beach level. (6) So, she updates her knowledge regarding health, and her concomitant priorities, and thus travel becomes *the* choice par excellence. (7) She finally runs out of money for travel and returns, still ill, to the city, cannot work, and continues her sad vacation there.

The main contributions of the paper are:

- We reformulate some previous results of ours and combine them into a multi-agent context by modifying, generalizing some of the definitions and by providing new ones.

- We provide the declarative semantics of the multi-agent system.

- We exhibit examples to show how our approach functions, including how preferring can enhance choice of reactivity in agents.

2 Logic Programming Framework

This section introduces the language of an agent. Typically, an agent can hold positive and negative information. An agent, for example, looking at the sky, can observe that it is snowing. Then, after a while, when weather conditions change, the agent observes that it is not snowing anymore. Therefore, he has to update his own knowledge with respect to the new incoming information. This implies that the language of an agent should be expressive enough to represent both positive and negative information. Knowledge updating refers not only to facts, as above, but also to rules, say new legislation, which can also be overridden by newer rules which negate previous conclusions.

In order to represent negative information in logic programs, we need more general logic programs that allow default negation $not\,A$ not only in premises of their clauses but also in their heads[1]. We call such programs generalized logic programs. It is convenient to syntactically represent generalized logic programs as propositional Horn theories. In particular, we represent default negation $not\,A$ as a standard propositional variable.

Propositional variables whose names do not begin with "*not*" and do not contain the symbols ":" and "÷" are called *objective atoms*. Propositional variables of the form *not* A are called *default atoms*. Objective atoms and default atoms are generically called *atoms*.

[1]For further motivation and intuitive reading of logic programs with default negations in the heads see [1].

Propositional variables of the form $i{:}C$ (where C is defined below) are called *projects*. $i{:}C$ denotes the intention (of some agent j) of proposing the updating the theory of agent i with C. We assume that projects cannot be negated.

Propositional variables of the form $j \div C$ are called *updates*. $j \div C$ denotes an update that has been proposed by j of the current theory (of some agent i) with C. We assume that updates can be negated. A *negated update not* $j{\div}C$ in the theory of an agent i indicates that agent j does not have the intention to update the theory of agent i with C.

Definition 1 Let $\mathcal{K}$ be a set of propositional variables consisting of objective atoms and projects such that $false \notin \mathcal{K}$. The propositional language $\mathcal{L}_{\mathcal{K}}$ generated by $\mathcal{K}$ is the language which consists of the following set of propositional variables:

$$\begin{aligned}\mathcal{L}_{\mathcal{K}} = \;& \mathcal{K} \cup \{false\} \cup \{not\,A \mid \text{for every atom } A \in \mathcal{K}\} \\ & \cup \{i \div C, not\, i \div C \mid \text{for every project } i{:}C \in \mathcal{K}\}.\end{aligned}$$

Definition 2 [Generalized rule] A generalized rule in the language $\mathcal{L}_{\mathcal{K}}$ is a rule of the form: $L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$ $(n \geq 0)$, where L_0 $(L_0 \neq false)$ is an atom and every L_i $(1 \leq i \leq n)$ is an atom, an update or a negated update from $\mathcal{L}_{\mathcal{K}}$.

Note that, according to the above definition, only objective atoms and default atoms can occur in the head of generalized rules.

We use the following convention. Given a generalized rule r of the form $L_0 \leftarrow L_1 \wedge \ldots \wedge L_n$, we use $head(r)$ to indicate L_0, $body(r)$ to indicate the conjunction $L_1 \wedge \ldots \wedge L_n$, $body_{pos}(r)$ to indicate the conjunction of all objective atoms in $body(r)$, and $body_{neg}(r)$ to indicate the conjunction of all default atoms in $body(r)$. Whenever L is of the form *not* A, *not* L stands for the atom A.

Definition 3 [Integrity constraint] An integrity constraint in the language $\mathcal{L}_{\mathcal{K}}$ is a rule of the form: $false \leftarrow L_1 \wedge \ldots \wedge L_n \wedge Z_1 \wedge \ldots \wedge Z_m$ $(n \geq 0, m \geq 0)$, where every L_i $(1 \leq i \leq n)$ is an atom, an update or a negated update, and every Z_j $(1 \leq j \leq m)$ is a project from $\mathcal{L}_{\mathcal{K}}$.

Integrity constraints are rules that enforce some condition over the state, and therefore always take the form of denials, without loss of generality, in a 2-valued semantics. Note that generalized rules are distinct from integrity constraints and should not be reduced to them. In fact, in generalized rules it is of crucial importance which atom occurs in the head.

Definition 4 [Generalized logic program] A generalized logic program P in the language $\mathcal{L}_{\mathcal{K}}$ is a set of generalized rules and integrity constraints in the language $\mathcal{L}_{\mathcal{K}}$.

Definition 5 [Query] A query Q in the language $\mathcal{L}_{\mathcal{K}}$ takes the form: $?-\ L_1 \wedge \ldots \wedge L_n$ $(n \geq 1)$, where every L_i $(1 \leq i \leq n)$ is an atom, an update or a negated update from $\mathcal{L}_{\mathcal{K}}$.

The following definition introduces rules that are evaluated bottom-up. To emphasize this aspect, we employ a different notation for them.

Definition 6 [Active rule] An active rule in the language $\mathcal{L}_{\mathcal{K}}$ is a rule of the form: $L_1 \wedge \ldots \wedge L_n \Rightarrow Z$ $(n \geq 0)$, where every L_i $(1 \leq i \leq n)$ is an atom, an update or a negated update, and Z is a project from $\mathcal{L}_{\mathcal{K}}$.

We use the following convention: given an active rule r of the form $L_1 \wedge \ldots \wedge L_n \Rightarrow Z$, we use $head(r)$ to indicate Z, and $body(r)$ to indicate $L_1 \wedge \ldots \wedge L_n$.

Active rules are rules that can modify the current state, to produce a new state, when triggered. If the body $L_1 \wedge \ldots \wedge L_n$ of the active rule is satisfied, then the project (fluent) Z can be selected and executed. The head of an active rule must be a project that is either internal or external. An *internal project* operates on the state of the agent itself, e.g., if an agent gets an observation, then it updates its knowledge, or if some conditions are met, then it executes some goal. *External projects* instead are performed on the environment, e.g., when an agent wants to update the theory of another agent.

We assume that for every project $i{:}C$ in $\mathcal{K}$, C is either a generalized rule, an integrity constraint, an active rule or a query. Thus, a project can only take one of the following forms:

$$i{:}(L_0 \leftarrow L_1 \wedge \ldots \wedge L_n)$$
$$i{:}(false \leftarrow L_1 \wedge \ldots \wedge L_n \wedge Z_1 \wedge \ldots \wedge Z_m)$$
$$i{:}(L_1 \wedge \ldots \wedge L_n \Rightarrow Z)$$
$$i{:}(?{-}L_1 \wedge \ldots \wedge L_n)$$

Note that projects can only occur in the head of active rules and in the body of integrity constraints. Updates and negated updates can occur in the body of generalized rules and in the body of integrity constraints. For example, the integrity constraint $false \leftarrow A \wedge j{:}B$ in the theory of an agent i enforces the condition that i cannot perform a project $j{:}B$ when A holds. The active rule $A \wedge not\, j \div B \Rightarrow j{:}C$ in the theory of i states to perform project $j{:}C$ if A holds and agent j has not wanted to update the theory of i with B.

Let $<$ be a binary predicate symbol whose set of constants includes all the generalized rules in the language $\mathcal{L}_{\mathcal{K}}$.

Definition 7 [Priority rule] A priority rule in the language $\mathcal{L}_{\mathcal{K}}$ is a generalized rule defining the predicate symbol $<$.

It is assumed that the set of constants of $<$ does not include $<$ itself. $r_1 < r_2$ means that rule r_1 is preferred to rule r_2.

Definition 8 [Prioritized logic program] A prioritized logic program P is a generalized logic program possibly containing priority rules.

The definition of prioritized logic program generalizes the one given in [7] to allow for priority rules. The intuition is that the program P formalizes the theory of an agent, and the priority rules express preferences among rules in P, in a sense to be made concrete later.

3 Agents and Multi-Agent Systems

This section presents the conception of agent and of multi-agent system. The initial knowledge of an agent is modeled by the notion of initial theory.

Definition 9 [Initial theory] The initial theory $\mathcal{T}$ of an agent α is a pair (P, R), where P is a prioritized logic program and R is a set of active rules.

P formalizes the initial knowledge state of the agent, and R characterizes its reactive behaviour. The knowledge of an agent can dynamically evolve when the agent receives new knowledge, albeit by self-updating rules. The new knowledge is represented in the form of an updating program.

Definition 10 [Updating program] Let $\mathcal{M}$ be a multi-agent system (defined below). An updating program U is a finite set of updates such that if an update $v \div C \in U$ then Ψ_v is an agent of $\mathcal{M}$.

An updating program contains the updates that will be performed on the current knowledge state of the agent. To characterize the evolution of the knowledge of an agent we need to introduce the notion of sequence of updating programs. Let $S = \{0, 1, \ldots, m\}$ be a set of natural numbers. We call the elements $s \in S$ *states*. A *sequence of updating programs* $\mathcal{U} = \{U^s \mid s \in S \text{ and } s > 0\}$ is a set of updating programs U^s superscripted by the states $s \in S$.

Definition 11 [Agent at state] Let $s \in S$ be a state. An agent α at state s, written as Ψ^s_α, is a pair $(\mathcal{T}, \mathcal{U})$, where $\mathcal{T}$ is the initial theory of α, and $\mathcal{U} = \{U^1, \ldots, U^s\}$ is a sequence of updating programs. If $s = 0$, then $\mathcal{U} = \{\}$.

An agent α at state 0 is defined by its initial theory and an empty sequence of updating programs, that is $\Psi^0_\alpha = (\mathcal{T}, \{\})$. At state 1, α is defined by $(\mathcal{T}, \{U^1\})$, where U^1 is the updating program containing all the updates that α has received at state 1, either from other agents or as self-updates. In general, an agent α at state s is defined by $\Psi^s_\alpha = (\mathcal{T}, \{U^1, \ldots, U^s\})$, where each U^i is the updating program containing the updates that α has received at state i.

Definition 12 [Multi-agent system] A multi-agent system $\mathcal{M} = \{\Psi^s_{\alpha_1}, \ldots, \Psi^s_{\alpha_n}\}$ at state s is a set of agents $\alpha_1, \ldots, \alpha_n$ each of which at state s.

Note that the definition of multi-agent system characterizes a static society of agents in the sense that it is not possible to add/remove agents from the system, and all the agents are at one common state.

To begin with, the system starts at state 0 where each agent is defined by $\Psi^0_\alpha = (\mathcal{T}_\alpha, \{\})$. Suppose that at state 0 an agent β wants to propose an update of the knowledge state of agent α with C by triggering the project $\alpha : C$. Then, at the next common state α will receive the update $\beta \div C$ indicating that an update has been proposed by β. Thus, at state 1, α will be $(\mathcal{T}_\alpha, \{U^1\})$, where $U^1 = \{\beta \div C\}$ if no other updates occur for α.

Within logic programs we refer to agents by using the corresponding subscript. For instance, if we want to express the update of the theory of an agent Ψ_α with C, we write the project α:C.

4 Semantics

This section introduces the declarative semantics of a multi-agent system. We need the following definitions. Let U be an updating program.

$$\Pi(U) = \{r \mid r \text{ is a generalized rule and } i \div r \in U\}$$

$$\Omega(U) = \{a \mid a \text{ is an active rule and } i \div a \in U\}$$

Definition 13 [Default assumptions] Let P be a prioritized logic program and M a model of P. The set of default assumptions is:

$$Default(P, M) = \{not\ A \mid \nexists r \in P \text{ such that } head(r) = A \text{ and } M \models body(r)\}.$$

To begin with, the knowledge of an agent α is characterized by its initial theory $\mathcal{T}$. Its knowledge can dynamically evolve when α receives new knowledge, via a sequence of updating programs $\mathcal{U} = \{U^1, \ldots, U^s\}$. Every updating program U^i in $\mathcal{U}$ contains the updates proposed at state i to α by the agents, including itself. Intuitively, the evolution of knowledge may be viewed as the result of, starting with $\mathcal{T}$, updating it with U^1, updating next with U^2, and so on. Similarly to the rationale of Dynamic Logic programming [1], the rules proposed via updates can be added to the knowledge state of α provided that α does not distrust the update and without bothering whether they conflict with previous knowledge of α itself. For example, if an agent α receives an update of the form $\beta \div C$ and α does not distrust it, then the rule C will be added to the knowledge state of α. Also similarly to DLP, the role of updating is to ensure that the rules contained in these newly added updates are in force, and that previous rules are still valid (by inertia) as far as possible, i.e., they are in force while they do not conflict with newly added rules, and they lose force if they so conflict and the new rules remain in force themselves. This rationale is at the basis of the notion of rejected rules.

Definition 14 [Rejected rule] Let $s \in S$ be a state and $\mathcal{U} = \{U^i \mid i \in S \text{ and } i > 0\}$ a sequence of updating programs. Let P be a prioritized logic program and M a model of P. The set of rejected rules at state s is:

$$Reject(P, \mathcal{U}, s, M) =$$
$$\{r \in (P \cup \Pi(U^a)) \mid \exists (i \div r') \in U^b \text{ such that: } r' \text{ is a generalized rule, } a < b \leq s \text{ (with } a = 0 \text{ if } r \in P), head(r) = not\ head(r'), M \models body(r'), \text{ and } M \models not\ distrust(i \div r')\}.$$

According to the definition above, a rule r either in P or proposed via an update in U^a is rejected at state s by a model M if there exists a newer rule r' proposed via a subsequent update in U^b by any agent i, such that the head of r' is the complement of the head of r, the body of r' is true in M and the update is not distrusted. Mark that: (1) whenever an agent distrust an update proposed by another agent, the rule proposed is not in force; (2) at state 0 no rule is rejected, i.e., $Reject(P, \mathcal{U}, 0, M) = \{\}$; (3) active rules cannot be rejected as the head of an active rule can only be a project, not a negated project; (4) all non rejected rules persist by inertia; (5) *distrust/1* is a reserved predicate which can itself be updated.

While updates allow us to deal with a dynamically evolving world, where rules change in time, preferences allow us to choose among various possible models of the world and among possible incompatible reactions. In [2], two criteria are established to remove unpreferred rules a program: removing unsupported rules, and removing less preferred rules defeated by the head of some more preferred one. Unsupported rules are rules whose head is true in the model and whose body is defeated by the model.

Definition 15 [Unsupported rule] Let P be a prioritized logic program and M a model of P. The set of unsupported rules is:

$$\mathit{Unsup}(P, M) = \{r \in P \mid M \models \mathit{head}(r), M \models \mathit{body}_{pos}(r) \text{ and } M \not\models \mathit{body}_{neg}(r)\}.$$

Notice that $M \not\models \mathit{body}_{neg}(r)$ implies that there exists a default atom $\mathit{not}\, A$ in $\mathit{body}_{neg}(r)$ such that $M \models A$.

Definition 16 [Unpreferred rule] Let P be a prioritized logic program and M a model of P. $\mathit{Unpref}(P, M)$ is a set of unpreferred rules of P and M iff:

$$\mathit{Unpref}(P, M) = \mathrm{least}(\mathit{Unsup}(P, M) \cup \mathcal{X})$$

where

$$\begin{aligned}\mathcal{X} = \{r \in P \mid\ & \exists r' \in (P - \mathit{Unpref}(P, M)) \text{ such that:}\\ & M \models r' < r,\ M \models \mathit{body}_{pos}(r') \text{ and}\\ & (\ \mathit{not}\ \mathit{head}(r') \in \mathit{body}_{neg}(r) \text{ or}\\ & (\mathit{not}\ \mathit{head}(r) \in \mathit{body}_{neg}(r'),\ M \models \mathit{body}(r))\ )\ \}.\end{aligned}$$

In other words, a rule is unpreferred if it is unsupported or is defeated by a more preferred rule (which is not itself unpreferred), or if it attacks (i.e., attempts to defeat) a more preferred rule.

Active rules are rules that enjoy the property that they can be triggered when their body is true in the model. Triggering an active rule means to execute the project occurring in its head. For example, given an active rule $L_1 \wedge L_2 \Rightarrow i : C$ and a model M, if $M \models L_1 \wedge L_2$, then the project $i : C$ can be executed. Executing the project $i : C$ denotes the intention of the agent of updating the knowledge state of agent i with C. The set of executable projects is the set containing the projects of all the active rules that are triggered and preferred by a model M.

Definition 17 [Executable project] Let R be a set of active rules and M a model. The set of executable projects is:

$$\mathit{Project}(R, M) = \{i{:}C \mid \exists r \in R \text{ such that } \mathit{head}(r) = i{:}C \text{ and } M \models \mathit{body}(r)\}.$$

The following definition presents the notion of preferred stable model of an agent α. Given the initial theory $\mathcal{T}$ of α and the sequence of updating programs $\mathcal{U}$, a preferred stable model at state s is a stable model of the program $\mathcal{X}$ that extends P to contain all the updates in $\mathcal{U}$, all the priority rules, and all those rules whose updates are not distrusted but that are neither rejected nor unpreferred. The preferred stable model contains also the projects of all active preferred rules (both in P or proposed via an update) that are triggered (i.e., all the executable projects).

Definition 18 [Preferred stable model] Let $s \in S$ be a state. Let $\Psi_\alpha = (\mathcal{T}, \mathcal{U})$ be an agent α and M an interpretation such that $\mathit{false} \notin M$. Assume that $\mathcal{T} = (P, R)$ and $\mathcal{U} = \{U^i \mid i \in S \text{ and } i > 0\}$. M is a preferred stable model of agent α at state s iff:

1. $\forall r_1, r_2 :$ if $(r_1 < r_2) \in M$, then $(r_2 < r_1) \notin M$
2. $\forall r_1, r_2, r_3 :$ if $(r_1 < r_2) \in M$ and $(r_2 < r_3) \in M$, then $(r_1 < r_3) \in M$
3. $M = \mathrm{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{PR}, M) \cup \mathit{Project}(R \cup \Omega(\bigcup_{1 \le i \le s} U^i), M))$, where:

   $\mathcal{PR} = P \cup \bigcup_{1 \le i \le s} U^i \cup \{r \mid \exists j{\div}r \in \bigcup_{1 \le i \le s} U^i \text{ and } M \models \mathit{not}\ \mathit{distrust}(j{\div}r)\}$

   $\mathcal{X} = \mathcal{PR} - \mathit{Reject}(P, \mathcal{U}, s, M) - \mathit{Unpref}(\mathcal{PR} - \mathit{Reject}(P, \mathcal{U}, s, M), M)$.

.

In general, an agent α may have several preferred stable models. Note that if $s = 0$ or each updating program in $\mathcal{U}$ is empty, then the last condition of the definition above reduces to:

$$\mathcal{PR} = P \qquad \mathcal{X} = \mathcal{PR} - \mathit{Unpref}(\mathcal{PR}, M)$$
$$M = \mathrm{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{PR}, M) \cup \mathit{Project}(R, M)).$$

This condition reduces further, if (in addition) there are no preference rules in P, to:

$$\mathcal{PR} = P \qquad \mathcal{X} = \mathcal{PR} - \mathit{Unsup}(\mathcal{PR}, M)$$
$$\begin{aligned} M &= \mathrm{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{PR}, M) \cup \mathit{Project}(R, M))\\ &= \mathrm{least}(P \cup \mathit{Default}(\mathcal{PR}, M) \cup \mathit{Project}(R, M)).\end{aligned}$$

When there are neither projects nor updates, the semantics reduces to the Preferential semantics of Brewka and Eiter [6]. If updates are then introduced, the semantics generalizes to the Updates and Preferences semantics of Alferes and Pereira [2], which extends DLP with preferences. Our semantics takes the latter and complements it with mutual and self-updates by means of active rules and projects, plus queries, within a society of agents.

We can now present the semantics $\mathcal{S}$ of a multi-agent system $\mathcal{M}$. The role of $\mathcal{S}$ is to characterize the relationship among the agents of $\mathcal{M}$. This is achieved by formalizing the relation between projects and updates. $\mathcal{S}$ is defined as the set of preferred stable models of each agent of $\mathcal{M}$. It uses the following function. Let $\{M^{s-1}_{\alpha_1}, \ldots, M^{s-1}_{\alpha_n}\}$ be a set of the set of models of the agents $\alpha_1, \ldots, \alpha_n$ at state $s - 1$.

$$\gamma(\{M^{s-1}_{\alpha_1}, \ldots, M^{s-1}_{\alpha_n}\}) = \{U^s_{\alpha_1}, \ldots, U^s_{\alpha_n}\}$$

where an update $(\alpha_j \div C) \in U^s_{\alpha_i}$ iff the project $\alpha_i{:}C$ belongs to all the models of α at state $s-1$, that is: $(\alpha_i{:}C) \in \bigcap M^{s-1}_{\alpha_j}$, for $1 \leq i, j \leq n$. The intuition is that α at state s can have several preferred stable models, each of which may trigger distinct active rules (and therefore each model will contain distinct executable projects). According to γ the projects that will be executed are the selectable projects that occur in every model of α at state s. Thus, the definition of γ characterizes a cautious behaviour for agents.[2] Note that when $j = i$ we have an internal update: the agent chooses to update its own theory.

Definition 19 [Semantics of $\mathcal{M}$] Let $\mathcal{M} = \{\Psi^0_{\alpha_1}, \ldots, \Psi^0_{\alpha_n}\}$ be a multi-agent system at state 0. Let $s \in S$ be a state. The semantics $\mathcal{S}^s$ of $\mathcal{M}$ at state s is defined inductively as follows.

Base Case ($s = 0$) That means that $\Psi^0_{\alpha_i} = (\mathcal{T}_{\alpha_i}, \{\})$, for every agent α_i at state 0, with $1 \leq i \leq n$. The semantics $\mathcal{S}^o$ of $\mathcal{M}$ at state 0 is $\mathcal{S}^0 = \{M^0_{\alpha_1}, \ldots, M^0_{\alpha_n}\}$, where every $M^0_{\alpha_i}$ is the set of preferred stable models of the agent α_i at state 0.

Inductive step ($s > 0$) Let $\{\Psi^{s-1}_{\alpha_1}, \ldots, \Psi^{s-1}_{\alpha_n}\}$ be the multi-agent system $\mathcal{M}$ at state $s-1$ and $\mathcal{S}^{s-1} = \{M^{s-1}_{\alpha_1}, \ldots, M^{s-1}_{\alpha_n}\}$ its semantics. Suppose that $\Psi^{s-1}_{\alpha_i} = (\mathcal{T}_{\alpha_i}, \{U^1_{\alpha_i}, \ldots, U^{s-1}_{\alpha_i}\})$, for $1 \leq i \leq n$. If $\gamma(\mathcal{S}^{s-1}) = \{U^s_{\alpha_1}, \ldots, U^s_{\alpha_n}\}$, then the multi-agent system $\mathcal{M}$ at state s is $\{\Psi^s_{\alpha_1}, \ldots, \Psi^s_{\alpha_n}\}$, with $\Psi^s_{\alpha_i} = (\mathcal{T}_{\alpha_i}, \{U^1_{\alpha_i}, \ldots, U^{s-1}_{\alpha_i}, U^s_{\alpha_i}\})$. Let $M^s_{\alpha_i}$ be the set of preferred stable models of agent α_i at state s, for $1 \leq i \leq n$. Then, $\mathcal{S}^s = \{M^s_{\alpha_1}, \ldots, M^s_{\alpha_n}\}$.

5 Distributed Databases and Cooperative Agents

The aim of this example is to illustrate the semantics of a multi-agent system. For simplicity, we do not yet make use of preference rules. Assume that we want to integrate several databases in order to minimize the amount of work required for administration in public offices. Suppose that we want to minimize the administrative procedure required for changing residence. Assume that Carlos (represented as c) has to change his own residence. Obviously, Carlos would like to notify his new residence only once in some public office, say a postoffice (represented as p), and he would like that all the related offices be automatically informed. That is, Carlos does not want to declare his residence again to the tax authority (represented as t), to his bank, and so on.

[2] An alternative definition for γ would be to execute all the projects that occur in any model of α at state s: brave behaviour.

This example involves communication and updates, and can be formulated as follows. The database of the postoffice maintains the information of the residence of the citizen. Every time a person changes residence, he or she has to communicate the new address at any public office (here the postoffice or the tax authority office). Then, it is the responsibility of that office to inform all other adequate offices. To do so, the database of each public office must have an active rule that is triggered when there is a change of residence. This active rule sends a message to all the public offices, like the tax authority, the bank, the working place, etc. In the following, we write $Body \Rightarrow A \wedge B$ as a shorthand for the active rules: $Body \Rightarrow A$ and $Body \Rightarrow B$. Let the initial theory of the postoffice be $\mathcal{T}_p = (P, R)$ where:

$$\begin{aligned} P &= \{rC\} \\ R &= \{NrC \wedge not\ comNrC \wedge not\ t \div NrC \Rightarrow \\ &\qquad (t{:}not\ rC) \wedge (t{:}NrC) \wedge (p{:}comNrC)\} \end{aligned}$$

where rC stands for the current residence of Carlos. The action rule in R states that when Carlos has a new residence (represented as NrC), the tax authority (t) must be notified if the postoffice has not done it yet ($not\ comNrC$) and it is not the case that it was the tax authority office that informed the postoffice of the change of residence ($not\ t \div NrC$). When notifying, the postoffice must declare that the residence of Carlos in no longer valid ($t{:}not\ rC$), it has to communicate the new residence ($t{:}NrC$) and finally it must update its own theory to register this notification ($p{:}comNrC$).

The initial theory $\mathcal{T}_t = (P, R)$ of the tax authority office is similar to the one of the postoffice, where:

$$\begin{aligned} P &= \{rC\} \\ R &= \{NrC \wedge not\ comNrC \wedge not\ p \div NrC \Rightarrow \\ &\qquad (p{:}not\ rC) \wedge (p{:}NrC) \wedge (t{:}comNrC)\}. \end{aligned}$$

Finally, let $\mathcal{T}_c = (P, R)$ be the initial theory of Carlos with:

$$\begin{aligned} P &= \{NrC\} \\ R &= \{NrC \wedge not\ comNrC \Rightarrow \\ &\qquad (p{:}not\ rC) \wedge (p{:}NrC) \wedge (c{:}comNrC)\}. \end{aligned}$$

Let $\mathcal{M} = \{\Psi^0_p, \Psi^0_t, \Psi^0_c\}$ be the multi-agent system at state 0, where $\Psi^0_p = (\mathcal{T}_p, \{\})$, $\Psi^0_t = (\mathcal{T}_t, \{\})$ and $\Psi^0_c = (\mathcal{T}_c, \{\})$. Then, the semantics $\mathcal{S}^0 = \{M^0_p, M^0_t, M^0_c\}$ is:

$$\begin{aligned} &M^0_p = \{\{rC\}\} \qquad M^0_t = \{\{rC\}\} \\ &M^0_c = \{\{NrC, p{:}not\ rC, p{:}NrC, c{:}comNrC\}\}. \end{aligned}$$

The preferred stable model of Carlos contains three (selectable) projects: the firts two denote his intention of proposing the updating of the theory of p with $not\ rC$ and NrC, while the last is an internal project, i.e., Carlos wants to update his own theory with $comNrC$ to register that he has communicated his new residence to the postoffice.

Now, in order to move to the next state, we must determine which selectable projects will be executed. This is carried out by γ. $\gamma(\mathcal{S}^0) = \{U_p^1, U_t^1, U_c^1\}$, where:

$$U_p^1 = \{c{\div}\mathit{not}\ rC, c{\div}NrC\} \quad U_t^1 = \{\}$$
$$U_c^1 = \{c{\div}comNrC\}.$$

Therefore, at state 1 the agents are defined by:

$$\Psi_p^1 = (\mathcal{T}_p, \{U_p^1\}) \quad \Psi_t^1 = (\mathcal{T}_t, \{U_t^1\}) \quad \Psi_c^1 = (\mathcal{T}_c, \{U_c^1\}).$$

The semantics $\mathcal{S}^1 = \{M_p^1, M_t^1, M_c^1\}$ at state 1 is:

$$M_p^1 = \{\{NrC, c{\div}\mathit{not}\ rC, c{\div}NrC, t{:}\mathit{not}\ rC, t{:}NrC,$$
$$p{:}comNrC\}\}$$
$$M_t^1 = \{\{rC\}\} \quad M_c^1 = \{\{NrC, c{\div}comNrC, comNrC\}\}.$$

Since $\gamma(\mathcal{S}^1) = \{U_p^2, U_t^2, U_c^2\}$, where:

$$U_p^2 = \{p{\div}comNrC\} \quad U_t^2 = \{p{\div}\mathit{not}\ rC, p{\div}NrC\}$$
$$U_c^2 = \{\}$$

the agents at state 2 are defined as:

$$\Psi_p^2 = (\mathcal{T}_p, \{U_p^1, U_p^2\}) \quad \Psi_t^2 = (\mathcal{T}_t, \{U_t^1, U_t^2\})$$
$$\Psi_c^2 = (\mathcal{T}_c, \{U_c^1, U_c^2\}).$$

The semantics $\mathcal{S}^2 = \{M_p^2, M_t^2, M_c^2\}$ at state 2 is:

$$M_p^2 = \{\{NrC, comNrC, p{\div}comNrC, c{\div}\mathit{not}\ rC, c{\div}NrC\}\}$$
$$M_t^2 = \{\{NrC, p{\div}\mathit{not}\ rC, p{\div}NrC\}\}$$
$$M_c^2 = \{\{NrC, comNrC, c{\div}comNrC\}\}.$$

Mark that the preferred stable model of every agent at state 2 contains the new residence of Carlos, *NrC*. Since $\gamma(\mathcal{S}^2) = \{\{\},\{\},\{\}\}$ meaning that there are no selectable projects, we have that $\mathcal{S}^3 = \mathcal{S}^2$. Thus, the multi-agent system does not evolve anymore: it has reached a fixed state where the agents do not interact further. This example can be extended in a number of ways, for example to model information filtering: if the postoffice maintains a more detailed profile of the citizen, then it can send the different offices only the information that is relevant for them.

6 Representation of Conflicting Information and Preferences

This example shows how to compute preferred stable models and how, by adding preferences, agents becomes more reactive. The example models a situation where an agent receives conflicting advice from two reliable authorities. Since the agent's expected behaviour is never to do anything that he was advised not to do by a reliable authority, the agent should neither perform the given action nor refuse to do it. Instead, the agent should remain *non-committal* and the outcome of his decision process should therefore be *undefined.*

Suppose that an agent, say Fabio, wants to go live alone (represented as lA). If Fabio receives advice from two reliable authoritative sources, his father and his mother, his expected behaviour is to perform an action recommended by a reliable authority unless it is in conflict with the advice received from another authority. Let $\mathcal{T}_f = (P, R)$ be the initial theory of Fabio.

$$P = \left\{ \begin{array}{ll} dont(lA) \leftarrow father{\div}nolA \wedge \mathit{not}\ do(lA) & (r_1) \\ do(lA) \leftarrow mother{\div}lA \wedge \mathit{not}\ dont(lA) & (r_2) \\ r_1 < r_2 \leftarrow fR & \\ r_2 < r_1 \leftarrow mR & \end{array} \right\}$$

$$R = \left\{ \begin{array}{l} do(lA) \Rightarrow f{:}rentA \\ dont(lA) \Rightarrow f{:}renovateR \end{array} \right\}$$

where *fR* stands for “father's responsibility”, and *mR* for “mother's responsibility”. The active rules in R say that if Fabio wants to live alone, then he must rent an apartment. On the contrary, if he does not want to live alone, then he must renovate his own room. Suppose that at state 1 his mother advises him to live alone, but the father advises not to do so: $U_f^1 = \{mother{\div}lA, father{\div}nolA\}$.

Fabio is unable to choose either $do(lA)$ or $dont(lA)$ and, as a result, does not perform any action whatsoever. In fact, at state 1 Fabio is defined by $\Psi_f^1 = (\mathcal{T}_f, \{U_f^1\})$ and there exist two preferred stable models, $M_f^1 = \{M_1, M_2\}$, where:

$$M_1 = \{dont(lA), lA, nolA, father{\div}nolA, mother{\div}lA,$$
$$f{:}renovateR\}$$
$$M_2 = \{do(lA), lA, nolA, father{\div}nolA, mother{\div}lA,$$
$$f{:}rentA\}.$$

For example, M_1 is a preferred stable model since:

$$\mathcal{PR} = P \cup \{father{\div}nolA, mother{\div}lA\} \cup \{nolA, lA\}$$
$$Reject(P, \{U_f^1\}, 1, M_1) = Unsup(\mathcal{PR}, M_1) = \{\}$$
$$Unpref(\mathcal{PR}, M_1) = \{\}$$
$$\mathcal{X} = \mathcal{PR} \qquad Default(\mathcal{PR}, M_1) = \{\mathit{not}\ do(lA), \ldots\}$$
$$Project(R, M_1) = \{f{:}renovateR\}.$$

Thus, we have that: $M_1 = \mathrm{least}(\mathcal{X} \cup Default(\mathcal{PR}, M_1) \cup Project(R, M_1))$. According to the definition of γ, as there exists no executable project that belongs to all the preferred stable models of Fabio at state 1, no project is executed: Fabio is not reactive at all in this situation.

Afterwards, consider the case where Fabio's parents separate and the judge assigns responsibility over Fabio to the mother: $U_f^2 = \{judge{\div}mR\}$. Now the situation changes since the second priority rule in R gives preference to the mother's wishes. In fact, the knowledge state of Fabio at state 2 is defined by $\Psi_f^2 = (\mathcal{T}_f, \{U_f^1, U_f^2\})$ that has a single preferred stable model: $M_f^2 = \{M_3\}$ with:

$$M_3 = \{r_2 < r_1, mR, do(lA), lA, nolA, father{\div}nolA, \\ mother{\div}lA, judge{\div}mR, f{:}rentA\}.$$

Since it holds that:

$$\mathcal{PR} = P \cup \{father{\div}nolA, mother{\div}lA, judge{\div}mR\} \\ \cup \{nolA, lA, mR\}$$

$$Reject(P, \{U_f^1, U_f^2\}, 2, M_3) = Unsup(\mathcal{PR}, M_3) = \{\}$$

$$Unpref(\mathcal{PR}, M_3) = \{r_1\}$$

$$\mathcal{X} = \mathcal{PR} - \{r_1\}$$

$$Default(\mathcal{PR}, M_3) = \{not\ dont(lA), \ldots\}$$

$$Project(R, M_3) = \{f{:}rentA\}$$

we have that: $M_3 = \text{least}(\mathcal{X} \cup Default(\mathcal{PR}, M_3) \cup Project(R, M_3))$. Now, since $f{:}rentA$ holds in every preferred stable model in M_f^2 (there exists only one), it can be executed. Thus, Fabio can rent an apartment and happily go live alone.

7 Updating preferences

The first stages in the "sad story" of Example 1 can be modeled as follows. For simplicity, we adopt unique numbers for rules, instead of the rules themselves, in the priority relation, and we let c stand for "living in the city", mt for "settling on a mountain", b for "living by the beach", t for "traveling", wk for "work", vac for "vacations", and mo for "possessing money". Let the initial theory $\mathcal{T}_m = (P, R)$ of Maria be:

$$R = \left\{ \begin{array}{l} mo \Rightarrow m{:}not\ wk \\ b \Rightarrow m{:}goToBeach \\ mt \Rightarrow m{:}goToMountains \end{array} \right\}$$

$$P = \left\{ \begin{array}{ll} (1) & c \leftarrow not\ mt \wedge not\ b \wedge not\ t \\ (2) & wk \\ (3) & vac \leftarrow not\ wk \\ (4) & mt \leftarrow not\ c \wedge not\ b \wedge not\ t \wedge mo \\ (5) & b \leftarrow not\ c \wedge not\ mt \wedge not\ t \wedge mo \\ (6) & t \leftarrow not\ c \wedge not\ mt \wedge not\ b \wedge mo \\ & 1 < 4 \leftarrow wk \\ & 1 < 5 \leftarrow wk \\ & 1 < 6 \leftarrow wk \\ & 4 < 6 \leftarrow vac \\ & 5 < 6 \leftarrow vac \\ & 6 < 1 \leftarrow vac \end{array} \right\}$$

Suppose that Maria receives the information from some agent l that she has received some money, that is, $U_m^1 = \{l{\div}mo\}$. Thus, as Maria does not distrust agent l, at state 1 she believes mo and therefore she proposes the internal update $m{:}not\ wk$ by triggering the first active rule in R. At state 2 Maria has two preferred stable models:

$$\{mt, vac, mo, l{\div}mo, m{\div}not\ wk, 4 < 6, 4 < 1, 5 < 6, \\ 5 < 1, 6 < 1\}$$
$$\{b, vac, mo, l{\div}mo, m{\div}not\ wk, 4 < 6, 4 < 1, 5 < 6, \\ 5 < 1, 6 < 1\}$$

and therefore she is unable to take any action at all. Suppose now Maria is advised by a friend f to prefer mountains to beaches, that is, $U_m^2 = \{f{\div}(4{<}5{\leftarrow}vac)\}$. At state 3 Maria has a single preferred stable model and can go to the mountains.

$$\{mt, vac, mo, l{\div}mo, m{\div}not\ wk, f{\div}(4{<}5{\leftarrow}vac), 4 < 5, \\ 4 < 6, 4 < 1, 5 < 6, 5 < 1, 6 < 1\}$$

8 Conclusion and Future Work

We have presented a logical framework of a multi-agent system in which each agent can communicate with and update other agents, can react to the environment, is able to prefer and to abduce hypotheses in order to explain observations made, which in turn may trigger reactions.

Applications in which our agent technology can have a significant potential to contribute, and that we are considering, are internet applications, e.g., information integration and web-site management. To this aim we have designed an agent architecture [11] and we are going to start implementing it. The problem of information integration consists in how to integrate data from multiple heterogeneous sources. The goal of a data integration system is to provide a uniform interface to a several data sources. An example of information integration is the task of providing information about movies from data sources on the web (for example, cf. [13]).

The goal of web site management applications is the flexible construction and modification of web sites. Web sites contain and integrate several pieces of data that are linked together into a navigational structure. One possible approach in these applications is to declaratively represent web sites, that is, to logically represent the kind of data and the structural aspects of web sites. One advantage would be the ability to automatically reconstruct and to enforce integrity constraints on web sites. The underlying idea is that such sites restructure themselves depending on the usage patterns, and can adapt themselves according to the profile of individual users. If we represent declaratively a web site, then we can reason about the structure of the site, for example, to see if the integrity constraints hold any time the site is updated (e.g., no dangling pointers, an employee's homepage should point to the department's homepage, etc.).

In the near future, we plan to develop a proof procedure for updating and preferring reasoning and to prove it correct and complete with respect to the declarative semantics presented in this paper. We are also investigating possible generalizations to the multi-agent system to make it not syncronous, and dynamic in a way that the agents can enter and leave the system.

Currently, we are working on an implementation of our agent framework. The framework is implemented as follows: its logical parts (e.g., logical reasoning, preferring, updating, etc.) are implemented in XSB Prolog [5], while its non-logical parts (e.g., agent communication, user interface, etc.) are implemented in Java. We then use InterProlog [4] to interface Java and XSB system.

References

[1] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinski, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *J. Logic Programming*, 45(1-3):43–70, 2000.

[2] J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego, I. P. de Guzmn, G. Brewka, and L. M. Pereira, editors, *Logics in AI, Procs. JELIA'00*, LNAI 1919, pages 345–360, Berlin, 2000. Springer.

[3] J. J. Alferes, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, and P. Quaresma. Preliminary exploration on actions as updates. Proc. of Joint Conf. on Declarative Programming (APPIA-GULP-PRODE'99).

[4] InterProlog. The system is available at: http://www.declarativa.com/InterProlog

[5] XSB-Prolog. The system is available at: http://xsb.sourceforge.net

[6] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.

[7] P. Dell'Acqua and L. M. Pereira. Updating agents. In S. Rochefort, F. Sadri and F. Toni (eds.), Procs. of the ICLP'99 Workshop on Multi-Agent Systems in Logic (MASL'99), 1999.

[8] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *ICLP'88*, pages 1070–1080. MIT Press, 1988.

[9] R. A. Kowalski and F. Sadri. Towards a unified agent architecture that combines rationality with reactivity. In D. Pedreschi and C. Zaniolo, editors, *Logic in Databases, Intl. Workshop LID'96*, LNCS 1154, pages 137–149. Springer, 1996.

[10] E. Lamma, F. Riguzzi, and L. M. Pereira. Strategies in combined learning via logic programs. *Machine Learning*, 38(1/2):63–87, 2000.

[11] J. A. Leite, J. J. Alferes, and L. M. Pereira. Minerva - A Dynamic Logic Programming Agent Architecture. To appear in: ATAL01 - 8th Int. Workshop on Agent Theories, Architectures, and Languages, 2001.

[12] J. A. Leite and L. M. Pereira. Iterated logic program updates. In J. Jaffar, editor, *Proc. Joint Intl. Conf. Symp. on Logic Programming 1998*, pages 265–278, Cambridge, Mass., 1998. MIT Press.

[13] A. Y. Levy and D. S. Weld. Intelligent internet systems. *Artificial Intelligence*, 118:1–14, 2000.