Enabling Agents to Update their Knowledge and to Prefer

Pierangelo Dell'Acqua[1], Luís Moniz Pereira[2]

[1] Department of Science and Technology
Campus Norrköping, Linköping University
Norrköping, Sweden
pier@itn.liu.se

[2] 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

1 Introduction

In a previous paper [5] we presented a combination of the dynamic logic programming paradigm proposed by J. J. Alferes et al. [1, 10] and a version of KS-agents proposed by Kowalski and Sadri [7]. 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 [8]. 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 [6]. 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 [5] to allow agents to update their knowledge and to prefer among alternative choices.

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 its knowledge base. In our framework, preferences can also be updated, possibly on advice from others. We exhibit examples to show how our approach functions, including how preferring can enhance choice of reactivity in agents. The declarative semantics of this approach to agents can be found in [4].

In the remaining of the paper, we will use the following example (taken from [2]) as a working example. This example, where rules as well as preferences change over time, shows the need to combine preferences and updates in agents, including the updating of preferences themselves.

Happy story. *(0) In the initial situation Maria is living and working everyday in the city. (1) 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. (2) 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. (3) 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.*

To keep the notation short, we follow some abbreviations: 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".

2 Enabling Agents to Update their Knowledge

This section introduces the language and concepts that allow agents to update their knowledge. 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 observed 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 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

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

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*.

Agents interact by exchanging information: for instance, when an agent β communicates C to an agent α. Then, α may or may not believe C. In case α does believe it, α has to update its knowledge accordingly. This kind of agent interaction is formalized via the concepts of project and update. Propositional variables of the form $\alpha{:}C$ (where C is defined below) are called *projects*. $\alpha{:}C$ denotes the intention (of some agent β) of proposing the updating the theory of agent α with C. We assume that projects cannot be negated.

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

Definition 1. *A generalized rule 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.*

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

Definition 2. *An integrity constraint 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.*

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 3. *A generalized logic program* P *is a set of generalized rules and integrity constraints.*

Definition 4. *A query* Q *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.*

Reactivity is a useful ability that agents must exhibit, e.g., if something happens, then the agent needs to perform some action. Agent's reactive behaviour is formalized via the notion of active rules. As active rules will be evaluated bottom-up, we employ a different notation for them to emphasize this aspect.

Definition 5. *An active rule 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.*

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 operate on the state of other agents, e.g., when an agent α proposes to update the theory of another agent β.

Example 1. Suppose that the underlying theory of Maria (represented with m) contains the following active rules:

$$\begin{array}{l} mo \Rightarrow m{:}not\ wk \\ b \Rightarrow m{:}goToBeach \\ t \Rightarrow p{:}bookTravel \end{array}$$

The heads of the first two active rules are project internal to Maria. The first states that if Maria has money, then she wants to update her own theory with *not wk*. The head of the last active rule is an external project: if Maria wants to travel, she proposes to book a travel to Pedro (represented with p).

We assume that for every project $\alpha{:}C$, 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:

$$\begin{array}{l} \alpha{:}(L_0 \leftarrow L_1 \wedge \ldots \wedge L_n) \\ \alpha{:}(false \leftarrow L_1 \wedge \ldots \wedge L_n \wedge Z_1 \wedge \ldots \wedge Z_m) \\ \alpha{:}(L_1 \wedge \ldots \wedge L_n \Rightarrow Z) \\ \alpha{:}(?{-}L_1 \wedge \ldots \wedge L_n) \end{array}$$

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 $false \leftarrow A \wedge \beta{:}B$ in the theory of an agent α enforces the condition that α cannot perform a project $\beta{:}B$ when A holds. The active rule $A \wedge not\beta{\div}B \Rightarrow \beta{:}C$ in the theory of α states to perform project $\beta{:}C$ if A holds and α has not been proposed (by β) to update its theory with B.

When an agent α receives an update $\beta \div C$, α may or may not believe C depending on whether α trusts β. This behaviour, characterized at the semantic level in [4], can be understood with an axiom schema of the form:

$$C \leftarrow \beta \div C \wedge not\ distrust(\beta \div C)$$

in the theory of α.

3 Enabling Agents to Prefer

Preferring is an important ability that agents must exhibit to select among alternative choices. Consider for instance the following example.

Example 2. Let Q be the underlying theory of Maria.

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

As Q has a unique 2-valued model $\{c, wk\}$, Maria decides to live in the city (c). Things change if we add the rule mo (indicating that Maria possesses money) to Q. In this case, Q has four models: $\{c, mo, wk\}$, $\{mt, mo, wk\}$, $\{b, mo, wk\}$ and $\{t, mo, wk\}$, and thus c is no longer true. Maria is now unable to decide where to live.

To incorporate the ability of preferring, we extend the agent language to make it possible to express preferences among the rules of the agent itself. Let $<$ be a binary predicate symbol whose set of constants includes all the generalized rules.

Definition 6. *A priority rule 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 7. *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 [5] 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.

Example 3. Let P be the following prioritized logic program:

$$P = Q \cup \{mo\} \cup \left\{ \begin{array}{l} 1 < 4 \leftarrow wk \\ 1 < 5 \leftarrow wk \\ 1 < 6 \leftarrow wk \\ 4 < 6 \leftarrow vac \\ 5 < 6 \leftarrow vac \\ 6 < 1 \leftarrow vac \end{array} \right\}$$

For simplicity, we adopt unique numbers for rules, instead of the rules themselves, in the priority relation $<$. The first three priority rules in P say that rule 1 is preferable to the rules 4, 5 and 6 if wk holds. If vac holds, then the rules 4 and 5 are both preferable to rule 6 which in turn is preferable to rule 1. As wk holds while vac does not, the first three priority rules are used to characterise the preferred model: $\{c, mo, wk\}$. Thus, by reducing the number of models of a program, priority rules allow us to select among several alternative models. Not always the priority rules allow us to select exactly one model. This may happen when the priority rules do not specify a complete linear order among the alternative choices. Suppose for instance that wk is false, then vac would hold and P would have two models: $\{mt, mo, wk\}$ and $\{b, mo, wk\}$.

As priority rules are generalized rules, preferences can also be updated. For example, we can update the priority rules of the program P in the previous example by updating with the rule $4 < 5 \leftarrow vac$. In this case, P will have a unique model (also in case vac holds).

4 Agents and Multi-Agent Systems

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

Definition 8. *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 9. *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 $\alpha \div C \in U$ then Ψ_α 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 10. *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 11. *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 proposes 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.

Example 4. This example formalizes the happy story as described in Section 1. Let the initial theory $\mathcal{T} = (P, R)$ of Maria be:

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

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

At state 0, 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 does not work and therefore decides to take a vacation. Unfortunately, the theory of Maria has two models:

$$\{mt, vac, mo, 4 < 6, 4 < 1, 5 < 6, 5 < 1, 6 < 1\}$$
$$\{b, vac, mo, 4 < 6, 4 < 1, 5 < 6, 5 < 1, 6 < 1\}$$

and therefore Maria is unable to take any action neither living by the beach nor settling up on a mountain. Suppose now that 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 the theory of Maria has a single preferred model:

$$\{mt, vac, mo, 4 < 5, 4 < 6, 4 < 1, 5 < 6, 5 < 1, 6 < 1\}$$

and Maria can happily settle up on the mountains.

5 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, and is able to prefer. 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 [4]. We are also investigating possible generalizations to the multi-agent system to make it not synchronous, and dynamic in a way that the agents can enter and leave the 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. P. Dell'Acqua and L. M. Pereira. Preferring and updating with multi-agents. Deductive Databases and Knowledge Management (DDLP'2001). To appear.
5. 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.
6. 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.
7. 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.
8. E. Lamma, F. Riguzzi, and L. M. Pereira. Strategies in combined learning via logic programs. *Machine Learning*, 38(1/2):63–87, 2000.
9. 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.
10. 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.
11. A. Y. Levy and D. S. Weld. Intelligent internet systems. *Artificial Intelligence*, 118:1–14, 2000.