A Logic Based Asynchronous Multi-Agent System

Pierangelo Dell'Acqua* Ulf Nilsson+ Luís Moniz Pereira†

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

+ Department of Computer and Information Science,
Linköping University, Linköping, Sweden
ulfni@ida.liu.se

† 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 logic programming based asynchronous multi-agent system in which agents can: communicate with one another; update themselves and each other; eliminate contradictory update rules; abduce hypotheses to explain observations, and use them to generate actions. The knowledge base of the agents is comprised of generalized logic programs, integrity constraints, active rules, and of abducibles. We characterize the interaction among agents via an asynchronous transition rule system, and provide a stable models based semantics. An example is developed to illustrate how our approach functions.

1 Introduction

In previous papers [1, 8, 9] we presented a logical formalization of a framework for multi-agent systems where we embedded a flexible and powerful kind of agent. In fact, these agents are rational, reactive, abductive, able to prefer and they can update the knowledge base of other agents (including their own). The knowledge state of each agent is represented by an an abductive logic program in which it is possible to express rules, integrity constraints, active rules, and priorities among rules. This allows the agents to reason, to react to the environment, to prefer among several alternatives, to update both beliefs and reactions, and to abduce hypotheses to explain observations. There we presented a declarative semantics for this kind of agent.[1]

[1] The agent's semantics is based on a characterization of updates given in [3] as a generalization of the stable model semantics of normal logic programs [11]. Such a semantics is generalized to the three-valued case in [4], which enables us to update programs under the well-founded semantics.

These agents were then embedded into a multi-agent system in such a way that the only form of interaction among them was based on the notions of project and update [8]. A project of the form $\alpha{:}C$ of an agent β denotes the intention of β of proposing to update the theory of an agent α with C. Correspondingly, an update of the form $\beta{\div}C$ in the theory of α denotes the intention of β to update the current theory of α with C. It is then up to α whether or not to accept that update. For example, if α trusts β and therefore α is willing to accept it, then α has to update its theory with C. The new information may contradict what α believes and, if so, the new believed information will override what is currently believed by α. β can also propose an update to itself by issuing an internal project $\beta{:}C$.

In [8] the interaction among the agents of the multi-agent system was defined synchronously: at each state of the system all the agents had to synchronize and to execute their projects. Consider a multi-agent system at state s whose agents α, β, and γ have the projects P_α, P_β, and P_γ to be executed. Then, according to the semantics of the multi-agent system, all three agents must synchronize and execute their projects. Thus, if the multi-agent system at state s is:

$$\langle \alpha, \beta, \gamma \rangle \quad \text{with} \quad P_\alpha = \{\beta{:}C_1, \alpha{:}C_2\} \quad P_\beta = \{\} \quad P_\gamma = \{\alpha{:}C_3\}$$

then, at state $s+1$ the system evolves to:

$$\langle \alpha + \{\alpha{\div}C_2, \gamma{\div}C_3\}, \beta + \{\alpha{\div}C_1\}, \gamma \rangle$$

Clearly, the synchronicity assumption of the system is the main limitation of the approach. In this paper, we elaborate over our previous work and set forth a logic based asynchronous multi-agent system, and provide its semantics. In the new framework, the interaction among agents is still centered on the notion of project and update, but it is executed asynchronously. Basically, the asynchronous communication between two agents is achieved through the use of buffers. Each agent, in fact, is equipped with a buffer where the incoming updates (from other agents) are stored. When an agent β wants to execute a project $\alpha{:}C$, β must synchronize with the buffer of α and communicate its projects. In turn, the buffer of α will store those projects in the form of updates. Then, it is up to α when to read its buffer and incorporate its updates into its own theory. In the paper we also treat the problem of inconsistencies that can arise when an agent communicates (via projects) to another agent (including itself) an inconsistency, e.g., when β has two projects of the form: $\alpha{:}A$ and $\alpha{:}not\, A$. Therefore the main contributions of the paper are:

- definition of an asynchronous transition rule system,
- declarative semantics of the multi-agent system (excluding preferences), and
- skeptical contradiction removal technique.

The usefulness of the approach proposed in this paper relies on the general approach to agents, via updates, of which this paper is an incremental advance with respect to communication and contradiction removal.

This approach to asynchronous communication is orthogonal to the declarative semantics of the abductive agents. Thus, we do not take into consideration the ability of preferring in agents because doing so will complicate the presentation of the declarative semantics, while letting unchanged the proposed approach.

In the remainder of the paper, we shall use the following as a working example, where agents can communicate with and update one another, shows among the others the asynchronous interaction of agents.

"Buying a car" story. *In the initial situation Maria wants to buy a car. To have a car one can either buy a car or steal one. Stealing a car is against Maria's principles, so she opts for buying one. Maria has two alternatives: she can either buy a utility car if she is in shortage of money, or buy a sports car if she has enough money to do so. Unfortunately, being unemployed, Maria does not have money at all. Thus, the only possibility is to ask her brother Pedro to lend her money. Meanwhile, her luck changes and she wins the lottery. Finally, depending on whether or not and when Pedro will lend her some money, and when she will be notified from the lottery of the win, she will opt for buying a Fiat or a Ferrari.*

To keep the notation short, we let *lmoney* stand for "to have little money" and *askMoney* for "to ask for borrowed money". We represent Maria with m, Pedro with p, the lottery with l, and the car seller with s.

2 Logic Programming Framework

Typically, an agent can hold positive and negative information, and it can update its own knowledge with respect to the new incoming information. Thus the language of an agent should be expressive enough to represent both positive and negative information. In order to represent negative information in logic programs, we need a language that allows default negation $not\, A$ not only in premises of clauses but also in their heads[2], i.e., 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*. For each objective atom A we assume a complementary propositional variable of the form $not\ A$, called a *default atom*. Objective atoms and default atoms are generically called *atoms*.

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. Projects can be negated. A *negated project* of the form $not\ \alpha{:}C$ denotes the intention of the agent of not proposing the updating of the theory of agent α with C.

[2] For further motivation and intuitive reading of logic programs with default negations in the heads see [3].

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. Updates can be negated. A *negated update* of the form *not* $\beta{\div}C$ in the theory of an agent α indicates that agent β does not have the intention to update the theory of agent α with C. Atoms, updates and negated updates are generically called *literals*.

Definition 1. Let $\mathcal{K}$ be a set of propositional variables consisting of objective atoms and projects such that the objective atom *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:

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

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 (with $L_0 \neq \mathit{false}$) is an atom and every L_i $(1 \leq i \leq n)$ is a literal 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 $\mathit{head}(r)$ to indicate L_0, $\mathit{body}(r)$ to indicate the conjunction $L_1 \wedge \ldots \wedge L_n$, $\mathit{body}_{pos}(r)$ to indicate the conjunction of all objective atoms and updates in $\mathit{body}(r)$, and $\mathit{body}_{neg}(r)$ to indicate the conjunction of all default atoms and negated updates in $\mathit{body}(r)$. 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: $\mathit{false} \leftarrow L_1 \wedge \ldots \wedge L_n \wedge Z_1 \wedge \ldots \wedge Z_m$ $(n \geq 0, m \geq 0)$, where every L_i $(1 \leq i \leq n)$ is a literal, and every Z_j $(1 \leq j \leq m)$ is a project or a negated 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 when updating an agent's theory.

Definition 4 (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 a literal 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 5 (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 a literal, and Z is a project or a negated 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 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 is a project that is either internal or external. An *internal project* operates on the state of the agent itself (self-update), 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. A negated project that occurs in the head of an active rule denotes the intention (of some agent) not to perform that project at the current state.

Example 1. Suppose that the underlying theory of Maria contains the following active rules:

$$R = \left\{ \begin{array}{l} buyCar \wedge not\ lmoney \Rightarrow p:askMoney \\ sportsCar \Rightarrow s:buyFerrari \\ utilityCar \Rightarrow s:buyFiat \\ stealCar \Rightarrow m:stealFerrari \end{array} \right\}$$

The heads of the first three active rules are projects external to Maria. The first active rule states that if Maria wants to buy a car and she does not have any money at all, then she asks her brother Pedro to lend her money. The second rule states that if Maria prefers to buy sports cars, then she buys a Ferrari from a car seller. The head of the last rule is a project internal to Maria. If she decides to steal a car, then she will steal a Ferrari.

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

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

Note that projects and negated projects can only occur in the heads of active rules and in the body of integrity constraints.

Example 2. 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. An active rule $A \wedge not \beta \div B \Rightarrow \beta{:}C$ in the theory of an agent α instructs it to perform project $\beta{:}C$ if A holds and agent β has not wanted to update the theory of α with B.

Definition 6 (Abductive logic program). An abductive logic program is a pair $(P, \mathcal{A})$, where P is a set of generalized rules and integrity constraints, and $\mathcal{A}$ is a set of atoms in the language $\mathcal{L}_{\mathcal{K}}$. The atoms in $\mathcal{A}$ are referred to as the abducibles.

Abducibles can be thought of as hypotheses that can be used to extend the given abductive logic program in order to provide an "explanation" for given queries. Explanations are required to meet all the integrity constraints in P. Abducibles may also be defined in P by generalized rules as the result of a self-update which adopts an abducible as a fact.

Example 3. Let $(P, \mathcal{A})$ be the following abductive logic program underlying the theory of Maria, where $\mathcal{A} = \{buyCar, stealCar\}$ and

$$P = \left\{ \begin{array}{l} sportsCar \leftarrow money \\ utilityCar \leftarrow lmoney \wedge not\ money \\ lmoney \leftarrow money \\ haveCar \leftarrow buyCar \\ haveCar \leftarrow stealCar \\ false \leftarrow stealCar \end{array} \right\}$$

The first generalized rule in P states that if Maria has money, then she prefers to buy sports cars. The third rule states that having money implies having a little money as well. The last rule is an integrity constraint that prevents Maria from stealing cars. For example, the goal $?{-}haveCar$ of having a car has one abductive explanation, i.e., *buyCar*. Stealing a car is not an abductive explanation because it does not satisfy the integrity constraint.

3 Abductive Agents

This section presents the conception of abductive agent. The initial knowledge of an agent is modeled by the notion of initial theory.

Definition 7 (Initial theory). The initial theory $\mathcal{T}$ of an agent α is a tuple $(P, \mathcal{A}, R)$, where $(P, \mathcal{A})$ is an abductive logic program and R is a set of active rules.

$(P, \mathcal{A})$ 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, or when it abduces new hypotheses to explain observations. The new knowledge is represented in the form of an updating program, and the new hypotheses in the form of a (finite) set $\triangle \subseteq \mathcal{A}$ of abducibles, possibly negated.

Definition 8 (Updating program). An updating program U is a finite set of updates.

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. 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 set $S = \{0, 1, \ldots, m, \ldots\}$. We call the elements $s \in S$ *states*.

Definition 9 (Agent α at state s). Let $s \in S$ be a state. An agent α at state s, written as Ψ_α^s, 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 by an empty sequence of updating programs, that is $\Psi_\alpha^0 = (\mathcal{T}, \{\})$. 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 0, either from other agents or as self-updates. In general, an agent α at state s is defined by $\Psi_\alpha^s = (\mathcal{T}, \{U^1, \ldots, U^s\})$, where each U^i is the updating program containing the updates that α has received at state $i-1$. 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 of Abductive Agents

This section introduces the declarative semantics of abductive agents. In the remainder of the paper, by (2-valued) *interpretation* M of $\mathcal{L}_\mathcal{K}$ we mean any set of propositional variables from $\mathcal{L}_\mathcal{K}$ such that for any A in $\mathcal{L}_\mathcal{K}$ precisely one A or *not* A belongs to M.

Definition 10 (Default assumptions). Let $(P, \mathcal{A})$ be an abductive logic program and M an interpretation of P. Let $\triangle \subseteq \mathcal{A}$ be a set of abducibles. The set of default assumptions[3] is:

$$\mathit{Default}(P, \triangle, M) = \{\mathit{not}\ A \mid A \text{ is an objective atom}, A \notin \triangle, \mathit{not}\ A \notin \triangle, \text{ and } \nexists r \in P \text{ such that } \mathit{head}(r) = A \text{ and } M \models \mathit{body}(r)\}.$$

A generalized rule r either already in P or proposed via an update in U^i is rejected at state s by a model M if there exists a generalized rule r' proposed via a subsequent update (or proposed at the same state as r, i.e., when $i = j$) in U^j by any agent α, 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[4]. r can also be rejected if there exists a current hypothesis $L \in \triangle$ that is the complement of the head of r.

Definition 11 (Rejected generalized rule). Let $(P, \mathcal{A})$ be an abductive logic program and M an interpretation of P. Let $s \in S$ be a state of an agent, $\mathcal{U} = \{U^i \mid i \in S \text{ and } i > 0\}$ a sequence of updating programs and $\triangle \subseteq \mathcal{A}$ a set

[3] For simplicity, we assume positive abducibles false by default.
[4] *distrust/1* is a reserved predicate which can itself be updated.

of abducibles. The set of rejected generalized rules at state s is:

$$
\begin{aligned}
&\mathit{RejectGr}(P,\mathcal{U},\triangle,s,M) = \\
&\{r \in P \mid r \text{ is a generalized rule and } \exists\, \alpha{\div}r' \in U^i \text{ such that } 1 \le i \le s, \\
&\qquad \mathit{head}(r) = \mathit{not}\ \mathit{head}(r'), M \models \mathit{body}(r') \text{ and } M \models \mathit{not}\ \mathit{distrust}(\alpha{\div}r')\} \\
&\cup \{r \in P \mid r \text{ is a generalized rule and } \exists\, L \in \triangle \text{ such that } \mathit{head}(r) = \mathit{not}\ L\} \\
&\cup \{r \mid r \text{ is a generalized rule, } \exists\, \beta{\div}r \in U^i \text{ and } \exists\, \alpha{\div}r' \in U^j \text{ such that } i \le j \le s, \\
&\qquad \mathit{head}(r) = \mathit{not}\ \mathit{head}(r'), M \models \mathit{body}(r') \text{ and } M \models \mathit{not}\ \mathit{distrust}(\alpha{\div}r')\} \\
&\cup \{r \mid r \text{ is a generalized rule, } \exists\, \beta{\div}r \in U^i, 1 \le i \le s \text{ and } \exists\, L \in \triangle \text{ such that} \\
&\qquad \mathit{head}(r) = \mathit{not}\ L\}.
\end{aligned}
$$

The idea behind the updating process is that newer rules reject older ones in a way that contradictions can never arise. Thus, contradictions could only ever arise between rules introduced in the same state. The ability to reject a rule r by another one r' proposed in the same updating program as r (that is, when $i = j$) prevents the having contradictory projects in the same state[5]. This means that contradictory rules from the same state reject one another, *skeptical contradiction removal*. Any agent α can prevent any type of updates from an agent β via the use of *distrust* in the theory of α, e.g., $\mathit{distrust}(\beta{\div}C) \leftarrow \mathit{liar}(\beta)$.

Active rules can also be rejected in a way similar to those of generalized rules.

Definition 12 (Rejected active rule). Let $(P, \mathcal{A}, R)$ be the initial theory of an agent and M an interpretation of P. 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. The set of rejected active rules at state s is:

$$
\begin{aligned}
&\mathit{RejectAr}(R,\mathcal{U},s,M) = \\
&\{r \in R \mid \exists\, \alpha{\div}r' \in U^i \text{ such that } 1 \le i \le s, \mathit{head}(r) = \mathit{not}\ \mathit{head}(r'), \\
&\qquad M \models \mathit{body}(r') \text{ and } M \models \mathit{not}\ \mathit{distrust}(\alpha{\div}r')\} \\
&\cup \{r \mid \exists\, \beta{\div}r \in U^i, r \text{ is an active rule and } \exists\, \alpha{\div}r' \in U^j \text{ such that } i \le j \le s, \\
&\qquad \mathit{head}(r) = \mathit{not}\ \mathit{head}(r'), M \models \mathit{body}(r') \text{ and } M \models \mathit{not}\ \mathit{distrust}(\alpha{\div}r')\}
\end{aligned}
$$

As the head of an active rule is a project and not an atom, active rules can only be rejected by active rules. Rejecting an active rule r makes r not triggerable even if its body is true in the model. Thus, by rejecting active rules we make the agent less reactive. Triggering an active rule means to execute the project occurring in its head. The set of projects selected to be executed is the set containing the projects of all the active rules that are triggered by a model M.

Definition 13 (Selected projects). Let R be a set of active rules and M an interpretation. The set of selected projects is:

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

[5] However, at a certain state it is still possible that there exist no models because of undefinedness caused by odd loops, e.g., $a \leftarrow \mathit{not}\ a$.

The following definition introduces the notion of abductive stable model of an agent α at a state s with set of hypotheses $\triangle$. Given the initial theory $\mathcal{T}$ of α, a sequence of updating programs $\mathcal{U}$ and the hypotheses $\triangle$ assumed at state s by α, an abductive stable model of α 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 hypotheses in $\triangle$, and all those rules whose updates are neither distrusted nor rejected. The abductive stable model contains also the selected projects.

Definition 14 (Abductive stable model of agent α at state s with hypotheses $\triangle$). Let $s \in S$ be a state. Let $\Psi_\alpha^s = (\mathcal{T},\mathcal{U})$ be agent α at state s and M an interpretation such that $\mathit{false} \notin M$. Assume that $\mathcal{T} = (P, \mathcal{A}, R)$ and $\mathcal{U} = \{U^i \mid i \in S \text{ and } i > 0\}$. Let $\triangle \subseteq \mathcal{A}$ be a set of abducibles. M is an abductive stable model of agent α at state s with hypotheses $\triangle$ iff:

$M = \text{least}(\mathcal{X} \cup \mathit{Default}(\mathcal{Y}, \triangle, M) \cup \mathit{Project}(\mathcal{Z}, M))$, where:

$\mathcal{Y} = P \ \cup\ \bigcup_{1\le i\le s} U^i \ \cup \triangle\ \cup \{r \mid r \text{ is a generalized rule or an integrity constraint and } \exists\, \alpha{\div}r \in \bigcup_{1\le i\le s} U^i \text{ such that } M \models \mathit{not\ distrust}(\alpha{\div}r)\}$

$\mathcal{X} = \mathcal{Y} - \mathit{RejectGr}(P,\mathcal{U}, \triangle, s, M)$

$\mathcal{Z} = R\ \cup\ \{r \mid r \text{ is an active rule and } \exists\, \alpha{\div}r \in \bigcup_{1\le i\le s} U^i \text{ such that } M \models \mathit{not\ distrust}(\alpha{\div}r)\} - \mathit{RejectAr}(R,\mathcal{U}, s, M)$.

The definition of abductive stable model semantics is based on the stable model semantics. The use of *least*(...) for generalized logic programs derives from the following two equivalent definitions [3]. A 2-valued interpretation M is a stable model of a generalized logic program P:
- if M is the least model of the Horn theory $P \cup M^-$, i.e. $M = \mathit{least}(P \cup M^-)$
- if $M = \{L \mid L \text{ is an atom and } P \cup M^- \vdash L\}$.

When there are neither projects nor updates nor hypotheses (i.e., $\triangle = \{\}$), the semantics reduces to the update semantics of Alferes et al. [2, 3]. Our semantics complements it with contradiction removal, abducibles, mutual and self-updates by means of active rules and projects, plus queries, within a society of agents.

Definition 15 (Abductive explanation of agent α at state s for query Q). Let $s \in S$ be a state and $\Psi_\alpha^s = (\mathcal{T},\mathcal{U})$ agent α at state s. Let Q be a query. An abductive explanation of agent α at state s for Q is any subset $\triangle$ of $\mathcal{A}$ such that there exists an abductive stable model M of α at s with hypotheses $\triangle$ and $M \models Q$.

Note that at state s an agent α may have several abductive explanations for a query Q.

Example 4. Let $\Psi_m^0 = (\mathcal{T}_m, \{\})$ be the theory of Maria at state 0 with $\mathcal{T}_m = (P, \mathcal{A}, R)$, where $(P, \mathcal{A})$ is is the abductive logic program of Example 3 and R is the set of active rules of Example 1. Suppose that Maria at state 0 has the query $?{-}\mathit{haveCar}$. Then, $M1 = \{\mathit{buyCar}, \mathit{haveCar}, p\,{:}\mathit{askMoney}\}$[6] is the unique

[6] Note that we do not write default atoms in models. Thus, a model $\{a, \mathit{not}\, b\}$ is written as $\{a\}$.

abductive stable model of Maria at state 0. In fact, we have that:

$$\triangle = \{buyCar\} \qquad \mathcal{Y} = P \cup \triangle \qquad \mathcal{X} = \mathcal{Y} \qquad \mathcal{Z} = R$$
$$Default(\mathcal{Y}, \triangle, M1) = \{not\ money, not\ lmoney, not\ utilityCar, not\ stealCar\}$$
$$Project(\mathcal{Z}, M1) = \{p : askMoney\}$$

Thus, it holds that $M1 = least(\mathcal{X} \cup Default(\mathcal{Y}, \triangle, M1) \cup Project(\mathcal{Z}, M1))$. The interpretation $M2 = \{stealCar, haveCar, m{:}stealFerrari\}$ is not an abductive stable model of Maria because it does not satisfy the integrity constraints in P.

5 Multi-Agent Systems

A multi-agent system consists of a set of agents acting concurrently and a number of transition rules that characterize the global behaviour of the system. Communication among agents is asynchronous, and modelled via buffers. A *buffer* is a sequence (possibly empty) of updating programs $U_1, \ldots, U_i$ ($i \geq 0$), written as $[U_1, \ldots, U_i]$. Each agent is equipped with a buffer. We write $\Psi^s_\alpha[U_1, \ldots, U_i]$ to indicate an agent α at state s with buffer $[U_1, \ldots, U_i]$. We use $' \mid '$ to indicate concurrency and assume it commutative and associative. Thus, we write $\Psi^{s1}_{\alpha_1}[U_1, \ldots, U_i] \mid \Psi^{s2}_{\alpha_2}[V_1, \ldots, V_j] \mid \Psi^{s3}_{\alpha_3}[W_1, \ldots, W_k]$ to indicate three agents α_1, α_2, and α_3 acting concurrently.

Definition 16 (Multi-agent system). A multi-agent system consists of a number of abductive agents $\alpha_1, \ldots, \alpha_n$ ($n \geq 2$) acting concurrently:

$$\Psi^{s1}_{\alpha_1}[U_1, \ldots, U_i] \mid \ \ldots \ \mid \Psi^{sn}_{\alpha_n}[V_1, \ldots, V_j]$$

together with the transition rules EXTP, INTP, and INCUP (defined below). The initial configuration of the multi-agent system is: $\Psi^0_{\alpha_1}[\,] \mid \ \ldots \ \mid \Psi^0_{\alpha_n}[\,]$.

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. Distinct agents in $\mathcal{M}$ may have different sets $\mathcal{A}$ of abducibles and they may be at different states.

The global behaviour of the multi-agent system is characterized by the transition rules EXTP, INTP, and INCUP. The intuition is that the interaction among agents occurs in two steps. Suppose that the projects of an agent α_1 to an agent α_2 are $\{\alpha_2{:}C_1, \ldots, \alpha_2{:}C_n\}$. To execute these projects α_1 synchronizes with the buffer of α_2 which will receive the updates $\{\alpha_1 \div C_1, \ldots, \alpha_1 \div C_n\}$. Then α_2 will read these updates from its buffer and will move on to the new state by incorporating them into its theory. In the following we write P_α to indicate the set of all executable[7] projects of an agent α.

[7] The projects of an agent that are executable is defined in Def. 19.

EXTP Rule for executing external projects of an agent α_1.

If $\exists\ (\alpha_2{:}A) \in P_{\alpha_1}$:

$$\Psi^{s1}_{\alpha_1}[U_1,\ldots,U_i] \mid \Psi^{s2}_{\alpha_2}[V_1,\ldots,V_j] \to \Psi^{s1}_{\alpha_1}[U_1,\ldots,U_i] \mid \Psi^{s2}_{\alpha_2}[V_1,\ldots,V_j,V_{j+1}]$$

where $\alpha_1 \neq \alpha_2$, $i \geq 0$, $j \geq 0$ and $V_{j+1} = \{\alpha_1 \div C \mid \text{for every project } \alpha_2{:}C \in P_{\alpha_1}\}$.

EXTP can be executed when the agent α_1 has at least one executable project of the form $\alpha_2{:}A$ (i.e., $\exists\ (\alpha_2{:}A) \in P_{\alpha_1}$).

INTP Rule for executing internal projects of an agent α.

If $\exists\ (\alpha{:}A) \in P_\alpha$:

$$\Psi^s_\alpha[U_1,\ldots,U_i] \to \Psi^s_\alpha[U_1,\ldots,U_i,U_{i+1}]$$

where $i \geq 0$ and $U_{i+1} = \{\alpha \div C \mid \text{for every project } \alpha{:}C \in P_\alpha\}$.

INCUP Rule for incorporating updating programs into the theory of an agent α from its buffer. Let $\Psi^s_\alpha = (\mathcal{T}, \{U^1,\ldots,U^s\})$.

$$\Psi^s_\alpha[U_1,\ldots,U_i] \to \Psi^{s+1}_\alpha[U_2,\ldots,U_i]$$

where $i \geq 1$ and $\Psi^{s+1}_\alpha = (\mathcal{T}, \{U^1,\ldots,U^s,U^{s+1}\})$ with $U^{s+1} = U_1$.

At state s the agent α incorporates the first updating program U_1 from its buffer and moves on to a new state $s+1$.

To ensure fairness we assume that the following two conditions hold:

1. Every agent capable of executing a project $\alpha{:}C$ indefinitely, will do so eventually. This guarantees that any agent β has the possibility to write into the buffer of an agent α and that β will do so.
2. Each agent will read its buffer sooner or later.

Remarks:

1. According to the transition rules above, the multi-agent system evolves non-deterministically depending on which transition rule is used and on which agents it is applied to.
2. Note also that in EXTP and INTP neither agent changes its state, only in INCUP agent α moves to a new state $s+1$ by incorporating the updating program U_1 into its own theory.
3. If we want to make the agent α_1 aware of the projects it has executed, then we can employ the following rule.

If $\exists\ (\alpha_2{:}A) \in P_{\alpha_1}$:

$$\Psi^{s1}_{\alpha_1}[U_1,\ldots,U_i] \mid \Psi^{s2}_{\alpha_2}[V_1,\ldots,V_j] \to \Psi^{s1}_{\alpha_1}[U_1,\ldots,U_i,U_{i+1}] \mid \Psi^{s2}_{\alpha_2}[V_1,\ldots,V_j,V_{j+1}]$$

where $\alpha_1 \neq \alpha_2$, $i \geq 0$, $j \geq 0$, $V_{j+1} = \{\alpha_1 \div A \mid \text{for every project } \alpha_2{:}A \in P_{\alpha_1}\}$ and $U_{i+1} = \{\alpha_1 \div exec(\alpha_2{:}A) \mid \text{for every project } \alpha_2{:}A \in P_{\alpha_1}\}$.

4. Other communication protocols can be programmed into a 'postmaster' agent to which every message is sent and then distributed accordingly.
5. Limitation: in this approach an agent α cannot execute two projects $\alpha_1{:}A_1$ and $\alpha_2{:}A_2$ on distinct agents α_1 and α_2 simultaneously. This ability would require the agent to synchronize with the buffer of the other two agents, which in practice is unrealistic.

The next example illustrates the use of transition rules.

Example 5. Let $\Psi_m^0 = (\mathcal{T}_m, \{\})$ be the theory of Maria at state 0.

If $P_l = \{m{:}money\}$ by EXTP:

$$\Psi_l^0[\,] \mid \Psi_m^0[\,] \rightarrow \Psi_l^0[\,] \mid \Psi_m^0[\{l \div money\}]$$

If $P_p = \{m{:}lmoney\}$ by EXTP:

$$\Psi_p^0[\,] \mid \Psi_m^0[\{l \div money\}] \rightarrow \Psi_p^0[\,] \mid \Psi_m^0[\{l \div money\}, \{p \div lmoney\}]$$

Then, by INCUP:

$$\Psi_m^0[\{l \div money\}, \{p \div lmoney\}] \rightarrow \Psi_m^1[\{p \div money\}]$$

with $\Psi_m^1 = (\mathcal{T}_m, \{U^1\})$, where $U^1 = \{l \div money\}$.

The multi-agent system evolves by applying transition rules either by buffering updates (via EXTP and INTP) or by incorporating updating programs from the buffer of an agent into its theory (via INCUP). Any time INCUP is employed by an agent α, both α and the multi-agent system $\mathcal{M}$ will move to a new state.

Definition 17 (Multi-agent system at state s). Let $\alpha_1, \ldots, \alpha_n$ $(n \geq 2)$ be abductive agents. Then, the multi-agent system at state 0 is:

$$\mathcal{M}^0 = \Psi_{\alpha_1}^0[\,] \mid \ \ldots \ \mid \Psi_{\alpha_n}^0[\,]$$

Suppose that the multi-agent system is at state s $(s \geq 0)$:

$$\mathcal{M}^s = \Psi_{\alpha_1}^{s_1}[U_1, \ldots, U_h] \mid \ \ldots \ \mid \Psi_{\alpha_n}^{s_n}[V_1, \ldots, V_j]$$

Then, if a transition rule INTP or EXTP is employed, the multi-agent system remains at the same state s. Instead, if the transition rule INCUP is employed by an agent α_i, then the system will move to a new state $s+1$:

$$\mathcal{M}^{s+1} = \Psi_{\alpha_1}^{s_1}[U_1, \ldots, U_h] \mid \ \ldots \ \mid \Psi_{\alpha_i}^{s_i+1}[T_2, \ldots, T_l] \mid \ \ldots \ \mid \Psi_{\alpha_n}^{s_n}[V_1, \ldots, V_j]$$

Note that, if $\mathcal{M}^s = \Psi_{\alpha_1}^{s_1}[U_1, \ldots, U_h] \mid \ \ldots \ \mid \Psi_{\alpha_n}^{s_n}[V_1, \ldots, V_j]$ then we have that $s = s_1 + \ldots + s_n$. The hypotheses $\triangle$ abduced by an agent α at state s are by default discarded at state $s+1$. Thus, α normally does not have memory of what it has assumed. If we want instead to model an agent α that is able to enrich its experience by assuming hypotheses during the process of proving queries and explaining observations, and is able to adopt previously assumed hypotheses, we can equip the theory of α with active rules of the form: $a \Rightarrow \alpha{:}a$, for whatever abducible $a \in \mathcal{A}$ desired. In this way, when α abduces a the project $\alpha{:}a$ will be triggered, and at the next state its theory will be updated with a itself.

6 Semantics of Multi-Agent Systems

We can now present the semantics $\mathcal{S}$ of a multi-agent system $\mathcal{M}$ embedding abductive agents. The role of $\mathcal{S}$ is to characterize the relationship among the agents of $\mathcal{M}$. $\mathcal{S}$ is defined as the set of abductive stable models of each agent in $\mathcal{M}$, after a non-deterministic transition.

Definition 18 (Semantics of MAS). Let $\mathcal{M}^s$ be the multi-agent system $\Psi^{s_1}_{\alpha_1}[U_1,\ldots,U_h] \mid \ldots \mid \Psi^{s_n}_{\alpha_n}[V_1,\ldots,V_j]$ at state s. Suppose that $Q^{s_i}_{\alpha_i}$ is the query of α_i at state s_i. Let $M^{s_i}_{\alpha_i}$ be the set of abductive stable models of α_i at state s_i each of which with set of hypotheses any abductive explanation $\triangle^{s_i}_{\alpha_i}$ of α_i for $Q^{s_i}_{\alpha_i}$, for every $1 \leq i \leq n$. Then, the semantics of $\mathcal{M}^s$ is $\mathcal{S}^s = \{M^{s_1}_{\alpha_1},\ldots,M^{s_n}_{\alpha_n}\}$.

In order to apply the transition rules, we have to define what is the set of executable projects for each agent. To do this, we use the function γ below. Let $M^{s_i}_{\alpha_i}$ be the set of abductive stable models of an agent α_i at state s_i:

$$\gamma(M^{s_i}_{\alpha_i}) = \{\alpha_j{:}C \mid (\alpha_j{:}C) \in \bigcap M^{s_i}_{\alpha_i}\}$$

γ characterizes the projects $\alpha_j{:}C$ that belong to every model of α_i at state s_i, that is: $(\alpha_j{:}C) \in \bigcap M^{s_i}_{\alpha_i}$. The projects in $\gamma(M^{s_i}_{\alpha_i})$ are the executable projects of α_i. The intuition is that α_i at state s_i can have several abductive stable models, each of which may trigger distinct active rules (and therefore each model will contain distinct selected projects). According to γ the projects that will be executed are the selected projects that occur in every model of α_i at state s_i. Thus, the definition of γ characterizes a cautious behaviour for agents.[8] Note that when $j = i$ we have a self-update: the agent chooses to update its own theory.

Definition 19 (Executable projects). Let $\mathcal{M}^s$ be the multi-agent system $\Psi^{s_1}_{\alpha_1}[u_1,\ldots,u_h] \mid \ldots \mid \Psi^{s_n}_{\alpha_n}[v_1,\ldots,v_j]$ at state s and $\mathcal{S}^s = \{M^{s_1}_{\alpha_1},\ldots,M^{s_n}_{\alpha_n}\}$ its semantics. Then, the set of executable project of an agent α_i at state s_i is: $P_{\alpha_i} = \gamma(M^{s_i}_{\alpha_i})$, for every $1 \leq i \leq n$.

7 "Buying a Car" Story

This section describes the example "Buying a Car Story" of Section 1 via a multi-agent M, with four agents: Maria (m), Pedro (p), the lottery (l) and the car seller (s). To keep the example simple, we illustrate how the system works mainly from the perspective of Maria. At state 0, $\mathcal{M}$ is:

$$\mathcal{M}^0 = \Psi^0_m[\,] \mid \Psi^0_p[\,] \mid \Psi^0_l[\,] \mid \Psi^0_s[\,]$$

[8] An alternative definition for γ would be to execute all the projects that occur in any model of α_j at state s: brave behaviour. Alternatively, one may introduce transition rules operating on distinct definitions of executable projects as a way to combine different forms of agent behaviour.

where $\Psi_m^0 = (\mathcal{T}_m, \{\})$ is defined in Example 4. Having Maria the goal $?{-}haveCar$, the unique abductive stable model of Maria at state 0 is $M1$ (cf. Example 4), that is $M_m^0 = \{M1\}$. Consequently, the set of executable projects of Maria is $P_m = \{p\!:\!askMoney\}$. Suppose that the executable projects of the remaining agents are: $P_p = \{\}$, $P_l = \{m\!:\!money\}$ and $P_s = \{\}$.

At state 0 the multi-agent system can evolve in different ways depending to which agents the transition rules are applied. In the first scenario, consider the case where the rule EXTP is applied to Maria. Then the system evolves to:

$$\mathcal{M}^0 = \Psi_m^0[\,] \mid \Psi_p^0[\{m \div askMoney\}] \mid \Psi_l^0[\,] \mid \Psi_s^0[\,]$$

and if INCUP is then applied to Pedro, to:

$$\mathcal{M}^1 = \Psi_m^0[\,] \mid \Psi_p^1[\,] \mid \Psi_l^0[\,] \mid \Psi_s^0[\,]$$

Assume that at state 1 the executable projects of Pedro are $P_p = \{m\!:\!lmoney\}$ and that EXTP is first applied to Pedro and then INCUP is applied to Maria. Then, the system evolves to a new state:

$$\mathcal{M}^2 = \Psi_m^1[\,] \mid \Psi_p^1[\,] \mid \Psi_l^0[\,] \mid \Psi_s^0[\,]$$

with $\Psi_m^1 = (\mathcal{T}_m, \{U^1\})$, where $U^1 = \{p \div lmoney\}$. As the unique abductive stable model of Maria at this state is $M2 = \{utilityCar, p \div lmoney, lmoney, s\!:\!buyFiat\}$ and the executable projects of Maria are $P_m = \{s\!:\!buyFiat\}$, the system evolves to:

$$\mathcal{M}^2 = \Psi_m^1[\,] \mid \Psi_p^1[\,] \mid \Psi_l^0[\,] \mid \Psi_s^0[\{m \div buyFiat\}]$$

Things go differently if other transition rules are used. Consider a second scenario where at state 0 the rule EXTP is applied to the lottery, INCUP to Maria and finally EXTP to Maria again. Then the system would have evolved to:

$$\mathcal{M}^2 = \Psi_m^1[\,] \mid \Psi_p^1[\,] \mid \Psi_l^0[\,] \mid \Psi_s^0[\{m \div buyFerrari\}]$$

8 Concluding Remarks

We have presented a logical framework of a multi-agent system in which each agent can communicate with and update other agents, and is able to abduce hyphoteses to explain observations. In Section 4 we have provided the declarative semantics of this kind of agent. Here we also treat the problem of inconsistencies that can arise when an agent simultaneously communicates contradictory information to another agent or itself. The interaction among agents has been characterized via an asynchronous transition rule system based on buffering in Section 5. In Section 6 we have given a stable model based semantics of our multi-agent system.

We believe that the theory of the agents construed is one rich evolvable basis, and suitable for engineering configurable, dynamic, self-organizing and self-evolving agent societies. Within the proposed multi-agent system framework we can represent groups, teams, coalitions of agents implicitly based on the internal mental states of its members. It is advocated, especially in open multi-agent systems (cf. [5, 12, 13]), that there is a need to make the organisational elements as well as the formalisation of the agent interactions of a multi-agent system externally visible rather than being embedded in the mental state of each agent, i.e., it is required to explicitly represent the organisational structure and the agent interactions. For example, we may formalize a group structure as an abstract description of a group that identifies all the roles and interactions that can occur within a group. This will allow us to precisely define concepts like permission, obligation, responsibility, social laws, requirements and roles of the society, etc. We are investigating how to explicitly represent organisational structures in our framework, and how to animate them with agents in a way that each agent will automatically have a view (perhaps partial) of the organisational structure and the externally visible events. We believe that this can be achieved through the concept of *organisational reflection*[9]. In fact, we may embed the theory of agents with various degrees of reflection abilities to make an agent able to introspect various aspects of the organisation where it belongs. In this way, an agent may only have a partial view of the entire organisational structure. Thus, the members of a society will have information of it (e.g., they will know its rules, requirements and laws), and they will have more information when reasoning about other agents' actions. In addition, the agents will be able to reason upon organisational structures and eventually try to modify them. In this way organisational structures will not be rigid, but flexible and can evolve with the agents' intervention.

Another interesting line of research concerns the investigation of invariants and other properties of the multi-agent system. In general, the behavior of the agents depends on the order of arrival of messages, and detecting these invariants will allow us to guarantee (to some degree) the behavior of the entire system.

Currently, we are working on an implementation of of our agent framework, implemented as follows: its logical parts (e.g., logical reasoning, updating, abducing, etc.) are implemented in XSB Prolog [7], while its non-logical parts (e.g., agent communication, user interface, etc.) are implemented in Java. We then use InterProlog [6] to interface Java and the XSB system.

Acknowledgements

L. M. Pereira acknowledges the support of POCTI project 40958 "FLUX - FleXible Logical Updates" and P. Dell'Acqua a Research Grant by the *Stiftelsen Lars Hiertas Minne*.

[9] This term was first introduced by J. Ferber and O. Gutknecht [10], but it is used here with a different meaning.

References

1. J. J. Alferes, P. Dell'Acqua, and L. M. Pereira. A compilation of updates plus preferences. 8th European Conf. on Logics in Artificial Intelligence (Jelia02). To appear.
2. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic logic programming. In A. Cohn and L. Schubert, editors, *KR'98*, pages 98–109. Morgan Kaufmann, 1998.
3. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The J. of Logic Programming*, 45(1-3):43–70, September/October 2000.
4. 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).
5. A. Artikis and G. Pitt. A formal model of open agent societies. Proc. of Autonomous Agents, 2001.
6. InterProlog. Available at http://www.declarativa.com/InterProlog/default.htm.
7. XSB-Prolog. Available at http://xsb.sourceforge.net.
8. P. Dell'Acqua and L. M. Pereira. Preferring and updating in abductive multi-agent systems. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Engineering Societies in the Agents' World (ESAW 2001)*, LNAI 2203, pages 57–73. Springer-Verlag, 2001.
9. P. Dell'Acqua and L. M. Pereira. Preferring and updating with multi-agents. In: 9th Int. DDLP-Workshop on Deductive Databases and Knowledge Management, which was held in the Stream Content Management of the 14th International Conference on Applications of Prolog (INAP2001), Tokyo, 2001.
10. J. Ferber and O. Gutnecht. A meta-model for the analysis and design of organisations in multi-agent systems. In *Int. Conf. on Multi-Agent Systems (ICMAS-98)*, pages 128–135. IEEE Computer Society, 1998.
11. 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.
12. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.
13. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, LNCS 1957, pages 127–141, Berlin, 2001. Springer-Verlag.