EXPRESSING COLLABORATION AND COMPETITION AMONG ABDUCTIVE LOGIC AGENTS

Anna Ciampolini, Evelina Lamma, Paola Mello and Paolo Torroni
Università degli Studi di Bologna
V.le Risorgimento 2, 40136 Bologna, Italy
{*aciampolini, elamma, pmello, ptorroni*}*@deis.unibo.it*

ABSTRACT

This paper presents a language for coordinating several logic-based agents capable of abductive reasoning. The system is particularly suited for solving problems with incomplete knowledge, where agents may need to make reasonable hypotheses about the domain. We defined a simple declarative language to express agent behavior, and in particular, two forms of coordination: collaboration and competition. An example in the area of medical diagnosis is presented to show the features of the language and the behavior of the proposed architecture.

1 Introduction

The area of logic multi-agent systems is currently a very active research field [1]. The agent concept is systematically used to represent entities with the ability to solve problems, reflecting the results on an environment which might be not completely under their control. Intelligent agents need deductive and pattern-matching capabilities to perform goals and activity requests on them. Recently, a number of works proposed systems where intelligent agents are modeled with logics [15, 6, 7].

In knowledge-intensive (distributed) applications, it is often the case that intelligent agents require some sort of *guess* about a computation (viz., a goal in a logic programming perspective) which cannot be performed (viz., solved) locally since their local knowledge is incomplete or they have to perform some form of hypothetical reasoning. This is very frequent, for instance, in diagnostic systems where each agent has to guess the causes of some symptoms. In his respect, some form of *open* or *abductive* reasoning has to be considered. Abduction has been widely recognized as a powerful mechanism for hypothetical reasoning in presence of incomplete knowledge [5, 10, 12], and also captures other important issues such as reasoning with defaults and beliefs [14, 18].

In a single-agent context, hypotheses are assumed provided that they are consistent with the agent's current knowledge. In a multi-agent setting, where each agent can perform abductive reasoning, different scenarios (and therefore different forms of combinations of the hypotheses raised by each agent) can occur depending on the role of each agent in the computation. In particular, an agent A can be involved in a computation in order to solve a problem in a *collaborative* or *competitive* way with respect to other agents. In the collaborative case, the task assigned to agent A is a sub-problem of the original one which has been split in a *divide and conquer* manner. When each agent is able to perform abductive reasoning, and more than one agent is involved in a computation in a collaborative way, each sub-problem is not completely independent, since the (abductive) explanations separately found by each agent have to be merged and to be consistent with each other. In the competitive case, the same task is assigned to agent A and other (competitive) agents; each agent can find a solution for the task, and one solution, among those found by the competitive agents, is selected.

In this work we present a language for coordinating a system of several logic agents, capable of abductive reasoning. Several autonomous agents, enclosing a local knowledge base, can either autonomously reason using their own local knowledge base, or can ask other agents to cooperate, in a collaborative or competitive way, in order to solve a given goal. The language is supported by *ALIAS* [2], a multi-agent architecture that will be shortly presented in the sequel. Our system uses logics for both modeling agent reasoning (in particular abduction, as in [19]), and expressing communication and coordination between agents (as in [8]). In particular, this work presents a language that, similarly to other proposals ([17]) allows to express communication and coordination among agents, using, unlike them, a declarative style.

2 The Architecture of *ALIAS*

The agent architecture we refer to is *ALIAS* (Abductive LogIc Agents System), a system where several intelligent agents, each enclosing a local knowledge base, can either

autonomously reason on its own local knowledge base or can exhibit a *social* behavior, interacting with other agents by different coordination schemata. A peculiar feature of *ALIAS* agents is that they can solve a problem by means of abductive reasoning.
The inner structure of each *ALIAS* abductive agent, shown in Figure 1, is basically composed of two modules: the *Abductive Reasoning Module* ($\mathcal{ARM}$, for short) and the *Agent Behavior Module* ($\mathcal{ABM}$). Both modules encapsulate a local knowledge base (KB): the *abductive* knowledge base (in the $\mathcal{ARM}$ module) and the *behavior* knowledge base (in the $\mathcal{ABM}$ module). The abductive KB is represented by an abductive logic program (for more details, see Section 3); the behavior KB is a set of logic clauses which describe the behavior of the agent within the environment in a declarative style by means of the *LAILA* language (presented in Section 4). In particular, the social behavior of each agent can be expressed within $\mathcal{ABM}$, by means of explicit communication and collaborative/competitive queries. Each time the agent's behavior requires the abductive explanation of a goal G, an interaction between $\mathcal{ABM}$ and $\mathcal{ARM}$ is needed, in order to locally start the abductive proof for G. It is up to the $\mathcal{ABM}$ to coordinate the computations carried on by different collaborating/competing agents.

Figure 1: The structure of a *ALIAS* agent

Within this framework, a multi-agent application is mapped onto several agents, possibly interacting, either cooperating or competing. Being the behavior of each agent modeled via a logic-based language, the computation is driven by goals to be demonstrated.

3 Abduction in a Multi-Agent Environment

Abduction is a well known hypothetical reasoning technique [10, 11], that allows to find explanations for a given observation, under the *open world* assumption. Hypothetical reasoning could be extremely useful when the knowledge about the problem domain is incomplete: this is the case of multi-agent applications where each agent might have a partial, and possibly incomplete, view of the world. For this reason *ALIAS* agents are equipped with abductive reasoning capabilities, thus being able to support the demonstration of goals even if their knowledge is incomplete. In some cases, however, it could be necessary to involve in a demonstration several agents, in order to obtain an abductive explanation that is consistent with their KBs. In *ALIAS*, agents cooperate in the abductive proof of goals by means of a distributed abductive algorithm. In that case, the produced abductive explanation is a set of hypotheses agreed by all agents. In the following we recall some preliminary concepts on abductive reasoning and introduce the algorithm.

3.1 Abductive Logic Programs

As described in 2, each *ALIAS* agent may enclose (in its $\mathcal{ARM}$ module) an *abductive logic program.*
An abductive logic program is a triple $\langle P, \mathcal{A}, IC\rangle$ where P is a logic program possibly with abducible atoms in clause bodies; $\mathcal{A}$ is a set of *abducible predicates*, i.e., *open* predicates which can be used to form explaining sentences; IC is a set of integrity constraints: each constraint is a denial containing at least one abducible. Given an abductive program $\langle P, \mathcal{A}, IC\rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of atoms Δ (i.e., the abductive explanation of G) which together with P entails G. It is also required that the program $P \cup \Delta$ is consistent with respect to IC. According to [10], negation as default, possibly occurring in clause bodies, can be recovered into abduction by replacing negated literals of the form *not a* with a new positive, abducible atom *not_a* and by adding the integrity constraint $\leftarrow a, not\_a$ to IC. The natural syntactic correspondence between a standard atom and its negation by default is given by the following notion of complement:

$$\bar{l} = \begin{cases} \alpha & \text{if } l = not\_\alpha \\ not\_\alpha & \text{otherwise} \end{cases}$$

where α is an atom.
We suppose that each integrity constraint in IC has at least one abducible in the body. We suppose that abducible predicates have no definition as in [13].

3.2 Extending abductive reasoning to multi-agent systems

In a multi-agent setting, we can equip each abductive agent with a distinct abductive logic program. In particular, in *ALIAS* each agent encloses in its $\mathcal{ARM}$ a local abductive program. Agents can dynamically group into bunches, with the purpose of finding the solution of a given goal in a *collaborative* way. In this perspective the set of program clauses and integrity constraints might differ from agent to agent, we assume that the set of abducible predicates (default predicates included) is the same for all the agents in a bunch. This implies that during the proof of a given goal, if an agent A assumes a new hypothesis h, all the arguing agents (i.e., the agents belonging to the same bunch) must check the consistency of h with their own integrity constraints. These checks could possibly raise new hypotheses, whose global consistency within the bunch have to be recursively checked. Therefore, in *ALIAS*, the abductive ex-

planation of a goal is a set of abduced hypotheses agreed by all agents in the bunch.
In order to perform abduction in a multi-agent environment we need to introduce some mechanism to support agent bunches, local abduction and global consistency checks. The algorithm we have adopted in the current implementation (*DAA*, Distributed Abduction Algorithm), discussed in [2], is based on a proof procedure, defined originally in [10] by Eshgi and Kowalski and further refined by Kakas and Mancarella [13], which is correct with respect to the abductive semantics defined in [4]. The proof procedure presented in [13] extends the basic resolution mechanism adopted in logic programming by introducing the notion of *abductive* and *consistency* derivation. Intuitively, an *abductive* derivation is the usual logic programming derivation suitably extended in order to consider abducibles. When an abducible atom h is encountered during this derivation, it is assumed, provided this is consistent. The consistency check of a hypothesis, then, starts the second kind of derivation. The *consistency* derivation verifies that, when the hypothesis h is assumed and added to the current set of hypotheses, any integrity constraint containing h fails (i.e., the bodies of all the integrity constraints containing h are false). During this latter procedure, when an abducible L is encountered, in order to prove its failure, an abductive derivation for its complement, $\overline{L}$, is attempted. The *DAA* algorithm extends the Kakas and Mancarella approach in the sense of distribution: now knowledge is distributed among several agents. In particular, while abductive derivation is limited to the local KB, consistency derivations have to be coordinated within the pool of logic agents of the current bunch.
It is worth to notice, however, that the *ALIAS* architecture is not strictly related to the Kakas-Mancarella abductive proof procedure. The same high-level features of the system could exploit different abduction algorithms. In particular, the major drawback of the current approach is that it applies only to ground predicates, thus limiting the real exploitation of the system. Therefore, as a future work, we plan to experiment other abductive proof procedures, such as, for instance, *SLDNFA* [9] in order to test the system in real applications. Moreover, the system could be extended to other forms of inference, such as, for instance, inductive reasoning. To this purpose, we plan also to integrate into the *ALIAS* architecture agents capable to learn, following an inductive approach.

4 The Coordination Language

In *ALIAS*, agent behavior is expressed in the *Language for AbductIve Logic Agents* (LAILA, for short). This language allows to model agent actions and interactions in a logic programming style. In particular we will focus on agent social behavior, and especially on how each agent can request and coordinate proofs of goals to other agents in the system. In the following subsections we will describe the syntax and the operational semantics of LAILA.

4.1 Syntax of LAILA

The syntax of LAILA is given as a BNF grammatics. Let us consider a system composed by $n+1$ agents. Each agent encapsulates a LAILA program describing its behavior.
Let $\mathcal{V}$ be the vocabulary of the language:

$$\mathcal{V} = \{\leftarrow, \&, ;, >, \downarrow, \mathbf{A_0}, \ldots, \mathbf{A_n}, \mathbf{a_0}, \ldots, \mathbf{a_k}, \mathbf{not}, \mathbf{true}\}$$

where:

- $\leftarrow$ is an implication operator;
- & is the *collaborative* coordination operator;
- ; is the *competitive* coordination operator;
- $>$ is the *communication* operator;
- $\downarrow$ is the *down-reflection* operator;
- A_i, $i = 0, \ldots n$ is the identifier of the $(i+1)$-th agent in the system;
- a_j, $j = 0, \ldots k$ is a ground atom (either abducible or not) occurring in the program.

A LAILA program is a set of L-clause. A L-clause is defined as follows:

$$\begin{array}{lcl}
L-clause & ::= & Atom \leftarrow Body. \\
Body & ::= & Formula; Body | Formula \\
Formula & ::= & SingleFormula \& Formula | \\
 & & SingleFormula \\
SingleFormula & ::= & \mathbf{true} | \downarrow Literal | CommFormula \\
CommFormula & ::= & Agent > Literal \\
Literal & ::= & Atom | \mathbf{not}\ Atom \\
Atom & ::= & \mathbf{a_1} | \mathbf{a_2} | \ldots | \mathbf{a_k} \\
Agent & ::= & \mathbf{A_0} | \mathbf{A_1} | \ldots | \mathbf{A_n}
\end{array}$$

A computation can be started by a query, defined as follows:

$$Query \quad ::= \quad ?Body$$

In order to help the reader in understanding the sense of L-clauses, we anticipate here two simple examples. Let us consider, for instance, the following LAILA competitive query, formulated by agent A_0:

$$? \downarrow \texttt{g1} \ ; \ \texttt{A1} > \texttt{g2}$$

It means that A_0 must either perform a local abductive derivation for g_1, *or* ask agent A_1 to demonstrate goal g_2. Le us consider, now, the following collaborative query, given by agent A_0:

$$? \ \texttt{A1} > \texttt{g3} \ \& \ \texttt{A2} > \texttt{g4}$$

It means that agent A_0 asks agent A_1 to prove g_3 and A_2 to prove goal g_4.

4.2 Operational Semantics of LAILA

In this section we present LAILA operational semantics. A LAILA program P is a collection of L-clauses, possibly distributed among a set of agents. In the following, $A_0, \ldots, A_n$ denote agents in the system; g denotes a single formula, G a composition of formulae; $\delta_1, \ldots \delta n$ denote conjunctions of abduced hypotheses; L is a literal; h denotes an atomic formula. Given a formula F, let us denote by:

$$b(F) = \{A | A \text{ is an agent in a commun. formula} \in F\}$$

In other words, $b(F)$ represents the set of agents involved in message exchanges by F. For instance, given the formula F: $A_1 > g_1;\ A_2 > g_2\ \&\ A_3 > g_3$, $b(F)$ is $\{A_1, A_2, A_3\}$. Let us introduce the definition of a successful top-down derivation.

Definition 1 (Successful top-down derivation) *Let P be a program and G a formula. a* top-down derivation *for G in P can be traced in terms of (possibly infinite) sequences of steps: $A \vdash_{\delta_{in,i}, \delta_{out,i}} G_i$, where A is an agent, δ_{in} and δ_{out} are sets of abduced hypotheses, and G_i is a formula. Each step is obtained by applying within A a suitable inference rule starting from the set δ_{in} of hypotheses, and possibly producing a new set of hypotheses δ_{out}. The first step of a top-down derivation starts from an empty set $\delta_{in,0}$. A* top-down *derivation is successful if, at some step k, the null formula is derived. The set $\delta_{out,k}$ represents the obtained abductive explanation associated with the successful derivation.*

In the following:

- $A_i \overset{abd}{\models}_{\delta_i \delta_j} s$, where s is a set of atoms, denotes the local abductive proof of the conjunction of all atoms in s, perfomed by agent A_i (whose meaning is given by the adopted abductive proof procedure);
- $B \overset{cons}{\models}_{\delta_i \delta_j} s$, where s is a set of atoms, denotes the consistency check of the conjunction of all atoms in s, with respect to the integrity constraints of all agents in bunch B.

Let us give the set of inference rules modeling the operational behavior of the system.

Definition 2 (True formula)

$$\frac{}{A \vdash_{\delta,\delta} \texttt{true}}$$

Definition 3 (Down reflection formula)

$$\frac{A \overset{abd}{\models}_{\delta_1,\delta_2} \{L\}}{A \vdash_{\delta_1,\delta_2} \downarrow L}$$

Therefore, the goal $\downarrow g$ starts a local abductive derivation for g.

Definition 4 (Communication formula)

$$\frac{A_1 \vdash_{\delta_1,\delta_2'} L \ \wedge\ \{A_0, A_1\} \overset{cons}{\models}_{\delta_1,\delta_2} \delta_2'}{A_0 \vdash_{\delta_1,\delta_2} A_1 > L}$$

Definition 5 (Collaborative formula)

$$\frac{A_0 \vdash_{\delta_1,\delta_2'} f \ \wedge\ A_0 \vdash_{\delta_1,\delta_2''} F \ \wedge\ \{A_0\} \cup b(f\&F) \overset{cons}{\vdash}_{\delta_1,\delta_2} \delta_2' \cup \delta_2''}{A_0 \vdash_{\delta_1,\delta_2} f\&F}$$

Thus, the following query, (formulated, for instance, by agent A_0):

$$?\ \mathtt{A_1 > q_1\ \&\ A2 > q_2}$$

has the following effects:

- A_0 asks A_1 to solve q_1; if q_1 succeeds, N ($N > 0$) abductive explanations $\delta_{1,i}$ ($i \in [1, \ldots N]$) for q_1 could be obtained.
- A_0 asks A_2 to solve q_2 ; if q_2 succeeds, M ($M > 0$) abductive explanations $\delta_{2,j}$ ($j \in [1, \ldots M]$) for q_2 could be obtained.

The abductive explanation for the query is therefore a set of hypotheses δ including both $\delta_{1,i}$ and $\delta_{2,j}$ ($i \in [1, \ldots N]$, $j \in [1, \ldots M]$) , such that it is consistent in the bunch $\{A_0, A_1, A_2\}$. If either $A_1 > q_1$ or $A_2 > q_2$ fail, the query Q fails.

Definition 6 (Competitive formula)

$$\frac{(A_0 \vdash_{\delta_1,\delta_2'} g \ \vee\ A_0 \vdash_{\delta_1,\delta_2''} G) \ \wedge\ \delta_2 \in \{\delta_2', \delta_2''\}}{A_0 \vdash_{\delta_1,\delta_2} g; G}$$

For instance, let us consider the following *competitive* query, formulated by agent A_0:

$$?\ \mathtt{A_1 > q_1\ \ ;\ A2 > q_2}$$

it causes:

- A_0 asks A_1 to solve q_1; if q_1 succeeds, N ($N > 0$) abductive explanations $\delta_{1,i}$ ($i \in [1, \ldots N]$) for q_1 could be obtained.
- A_0 asks A_2 to solve q_2 in the bunch $\{A_0, A_2\}$; if q_2 succeeds, M ($M > 0$) abductive explanations $\delta_{2,j}$ ($j \in [1, \ldots M]$) for q_2 could be obtained.

The resulting abductive explanation is either $\delta_{1,i}$ ($i \in [1, \ldots N]$) or $\delta_{2,j}$ ($j \in [1, \ldots M]$). It is worth to notice that in this case the selection rule for the abductive explanation δ is not specified: it could be either purely non-deterministic or it could follow a different criterium (*e.g.*, priority among the components of the query). If both q_1 and q_2 fail, the competitive query fails.

Definition 7 (Consistency check)

$$\frac{\forall A_i \in B \quad A_i \overset{abd}{\models}_{\delta_1,\delta_2^i} \delta \ \wedge \ B \overset{cons}{\models}_{\delta_1,\delta_2} \ \bigcup_i \delta_2^i}{B \overset{cons}{\models}_{\delta_1,\delta_2} \ \delta}$$

$$\frac{}{B \overset{cons}{\models}_{\delta_1,\delta_1} \ \delta}$$

Finally, the semantics of atomic formulas is described, as usual, by the following inference rule:

Definition 8 (Atomic formula)

$$\frac{\exists h' \leftarrow G \ \in A \quad \wedge \exists \ \ \theta = mgu(h,h') \quad \wedge \quad A \vdash_{\delta_1,\delta_2} \{G\}\theta}{A \vdash_{\delta_1,\delta_2} h}$$

5 An Example

The domain of medical diagnosis is particularly suited for providing examples for both the collaborative and the competitive case. Let us consider for instance a group of medical doctors, each one expert in a particular area (e.g. gastroenterology, ematology, etc.) who have to collaborate in order to formulate a diagnosis for a given set of symptoms. Each expert can be modeled by an abductive agent whose task is to find an hypothesis (i.e., a disease) as an explanation for a sub-part of the symptoms (i.e., those relevant for his/her area) given as observations to the agent. In some cases, the hypotheses raised by an agent can generate an inconsistency with other hypotheses raised by a collaborative agent. For instance, a certain symptom s, let us say low blood pressure, does not occur when the disease d is present, e.g., d is a disease associated with hypertension.
As an example of the competitive case, let us consider again a group of medical doctors, each one expert in a particular area who have to find a diagnosis for the symptom. Each expert can be modeled by an abductive agent whose task is to find an hypothesis (i.e., a diagnosis) as an explanation for the symptoms given as observations to the agents. Different diagnosis can thus be proposed by the different agents, and the *best* one can be chosen according to some policy (for instance, the one with major incidence, or the most plausible one, with respect to the clinical history of the patient).
Now, let us suppose agent A_0 (representing a patient) wants to query agents A_1 and A_2 (both representing medical specialists) about some symptoms `s1` and `s2`, observed on himself.
Let us consider agent A_1. His knowledge is modeled by the following L-clauses:

$$\mathcal{ABM}: \quad \mathtt{s1} \leftarrow\downarrow \mathtt{s1}. \qquad \mathcal{ARM}: \quad \begin{array}{l} \mathtt{s1} \leftarrow \mathtt{d1}. \\ \leftarrow \mathtt{d1}, \mathtt{s2}. \end{array}$$

where d_1 represents a certain disease.

Finally, let the agent A_2 knowledge be the following:

$$\mathcal{ABM}: \quad \begin{array}{l} \mathtt{s1} \leftarrow\downarrow \mathtt{s1}. \\ \mathtt{s2} \leftarrow\downarrow \mathtt{s2}. \end{array} \qquad \mathcal{ARM}: \quad \begin{array}{l} \mathtt{s1} \leftarrow \mathtt{d3}. \\ \mathtt{s2} \leftarrow \mathtt{d2}. \end{array}$$

where d_2 and d_3 represent diseases.
Both agents could give explanations for symptom s_1. In respect to symptom s_2, though, only agent A_2 is able to formulate a diagnosis. Therefore agent A_0, the patient, formulates the following query:

$$? \ (\mathtt{A1} > \mathtt{s1} \ ; \ \mathtt{A2} > \mathtt{s1}) \ \& \ \mathtt{A2} > \mathtt{s2}$$

A_0's $\mathcal{ABM}$ interprets the query and sends three different messages (*ask*) to request demonstrations to other agents:

(1) A_0 asks A_1 s_1 in bunch $B_1' = \{A_0, A_1\}$, which means that A_1 has to demonstrate $s1$ in bunch B_1'. A_1 down-reflects s_1 to the local $\mathcal{ABM}$ module. This computation succeeds, producing the abductive explanation δ_1' = {`d1, not s2`}.
(2) A_0 asks A_2 s_1 in bunch $B_1'' = \{A_0, A_2\}$, which means that A_1 has to demonstrate `s1` in the bunch B_1''. A_1 down-reflects s_1 to the local $\mathcal{ARM}$ module. This computation succeeds producing the abductive explanation δ_1'' = {`d3`}.
(3) A_0 asks A_2 s_2 in bunch $B_2 = \{A_0, A_2\}$, which means that A_1 has to demonstrate `s2` in bunch B_2 the query. A_2 maps s_2 into the abductive query s_2 for the local $\mathcal{ABM}$ module This computation succeeds producing the abductive explanation δ_2 = {`d2`}.

The three computations start in parallel and have to be coordinated according to the meaning of the collaborative / competitive operators in the query raised by A_0. In particular, after one or both of the competitive computations (computations 1 and 2) ends, the agent selects one of the two δ produced. We have two different cases:

(a) δ_1' = {`d1, not s2`} is selected. The collaborative composition with δ_2 follows, in order to check their consistency and to produce a temporary hypothesis which is the union of the two solutions: $\delta = \delta_1' \cup \delta_2$ = {`d1, not s2, d2`}. If it was inconsistent, $\mathcal{ABM}$ would trigger a backtracking mechanism in order to find another solution from `(`A_1`>s1;` A_2`>s1)`. This is not the case, therefore A_0 issues the creation of a last bunch $B = \{A_0, A_1, A_2\}$, to which A_0 will submit as a query δ = {`d1, not s2, d2`}, in order to test it consistency, and to generate a solution to the whole query (i.e., a final Δ). Unfortunately, this δ fails because of A_2's rule: `s2` $\leftarrow$ `d2`, therefore a backtracking, again, is needed (in this case, it leads to case b).
(b) δ_1'' = {`d3`} is selected. The collaborative composition with δ_2 follows, producing: $\delta = \delta_1'' \cup \delta_2$ = {`d3, d2`}, which is consistent. A_0 issues the creation of a bunch $B = \{A_0, A_2\}$, to which A_0 will submit as a query δ = {`d3, d2`}, which succeeds.

Therefore, the only possible solution for the initial query is supported by the abductive explanation δ = {`d3, d2`} as a solution to the initial query.

6 Conclusion And Future Work

In this work we presented a language for expressing communication and coordination among logic-based agents in a declarative style. Agents are thought to interact within a system, ALIAS, whose current implementation allows distributed abduction to be performed among dynamically grouped agents [2, 3].
In the future, we intend to improve the implementation of ALIAS in order to support the coordination language, and to extend it to cope with other abductive proof procedures and other forms of reasoning, e.g. inductive reasoning. Our intention is also to test the system in a real world case, in particular in the field of medical diagnosis.

6.1 Acknowledgements

This work has been supported by M.U.R.S.T. Project on *Intelligent agents: interaction and knowledge acquisition.*

REFERENCES

[1] S. Rochefort, F. Sadri, and F. Toni, eds., *Proc. Int. Workshop on Multi-Agent Systems in Logic Programming.* In conjunction with ICLP'99, Las Cruces, New Mexico, 1999.

[2] A. Ciampolini, E. Lamma, P. Mello and P. Torroni. An Implementation for Abductive Logic Agents. In *Proc. AI*IA99*, Springer-Verlag LNAI 1792 (to appear).

[3] A. Ciampolini, E. Lamma, P. Mello and P. Torroni. Rambling Abductive Agents in *ALIAS.* In [1].

[4] A. Brogi, E. Lamma, P. Mancarella, and P. Mello. A Unifying View for Logic Programming with Non-Monotonic Reasoning. In *Theoretical Computer Science*, Vol. 184, 1–49, North Holland, 1997.

[5] P. T. Cox and T. Pietrzykowski. Causes for events: Their computation and applications. In *Proc. CADE-86*, 608, 1986.

[6] P. Dell'Acqua, and L. M. Pereira. Updating Agents. In [1].

[7] P. Dell'Acqua, F. Sadri, and F. Toni. Combining Introspection and Communication with Rationality and Reactivity in Agents. In U. Furbach and L. Farinas del Cerro eds., *Proc. 6th European Workshop on Logics in Artificial Intelligence*, Springer Verlag LNAI 1489, 17–32 (1998)

[8] P. Dell'Acqua, F. Sadri, and F. Toni. Communicating Agents. In [1].

[9] M. Denecker and D. De Schreye SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, Elsevier, 1998.

[10] K. Eshgi and R. A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proc. 6th Int. Conf. on Logic Programming*, 234. MIT Press, 1989.

[11] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[12] A. C. Kakas, and P. Mancarella. Generalized stable models: a semantics for abduction. In *Proc. 9th European Conf. on Artificial Intelligence.* Pitman Pub., 1990.

[13] A. C. Kakas, and P. Mancarella. On the relation between Truth Maintenance and Abduction. In *Proc. PRICAI90*, 1990.

[14] R. A. Kowalski. Problems and promises of computational logic. In *Proc. Symp. on Computational Logic*, 1-36. Springer-Verlag, Nov. 1990.

[15] R. A. Kowalski, and F. Sadri. From Logic Programming to Multi-Agent Systems. In *Annals of Mathematics and Artificial Intelligence*, 1999 (to appear).

[16] N. R. Jennings, M. J. Wooldridge, eds., *Agent Technology.* Springer-Verlag, 1998.

[17] Y. Labrou, and T. Finin. A semantics approach to KQML – a general purpose communication language for software agents. In *Proc. 3rd Int. Conf. on Information and Knowledge Management*, 1994.

[18] D. L. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27. Elsevier, 1988.

[19] F. Sadri, and F. Toni. Abduction with Negation as Failure for Active Databases and Agents. In *LNAI, Proc. AI*IA99.* Springer-Verlag, 1999 (to appear).

[20] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.