Combining introspection and communication with rationality and reactivity in agents

P. Dell'Acqua[1], F. Sadri[2] and F.Toni[2]

[1] Computing Science Department
Uppsala University
Box 311, S-751 05 Uppsala, Sweden
pier@csd.uu.se

[2] Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ, UK
{fs,ft}@doc.ic.ac.uk

Abstract. We propose a logic-based language for programming agents that can reason about their own beliefs as well as the beliefs of other agents and can communicate with each other. The agents can be reactive, rational/deliberative or hybrid, combining both reactive and rational behaviour. We illustrate the language by means of examples.

1 Introduction

Kowalski & Sadri [12] propose an approach to agents within an extended logic programming framework. In the remainder of the paper we will refer to their agents as KS-agents. KS-agents are *hybrid* in that they exhibit both *rational* (or *deliberative*) and *reactive* behaviour. The reasoning core of KS-agents is a proof procedure that combines forward and backward reasoning. Backward reasoning is used primarily for planning, problem solving and other deliberative activities. Forward reasoning is used primarily for reactivity to the environment, possibly including other agents. The proof procedure is executed within an observe-think-act cycle that allows the agent to be alert to the environment and react to it as well as think and devise plans. Both the proof procedure and the KS-agent architecture can deal with temporal information. The proof procedure (IFF proof procedure [9]) treats both inputs from the environment and agents' actions as *abducibles* (hypotheses).

Barklund et al. [1] and Costantini et al. [6] present an extension to the Reflective Prolog programming paradigm [7, 8] to model agents that are introspective and that communicate with each other. We will refer to this approach as Reflective Prolog with Communication (RPC). In RPC introspection is achieved via the meta-predicate *solve* and communication is achieved via the meta-predicates *tell* and *told*. A communication act is triggered everytime an agent $agent_1$ has a goal of the form $agent_1 : told(agent_2, A)$. This stands both for "$agent_1$ is told by $agent_2$ that A" as well as "$agent_1$ asks $agent_2$ whether A". This goal is solved by $agent_2$ telling $agent_1$ that A, that is, $agent_2 : tell(agent_1, A)$, standing for "$agent_2$ tells $agent_1$ that A". The information is passed from $agent_2$ to $agent_1$ by eventually instantiating A. A main limitation of this approach is that agents cannot tell anything to other agents unless explicitly asked.

The ability to provide agents with some sort of "proactive" communication primitive is widely documented in literature [18–20, 5, 14, 17]. For example, one can model agents that advertise their services so that other agents, possibly with the help of *mediators*, can find agents that provide services for them.

An interesting application of agents is that of a virtual marketplace on the Web where users create autonomous agents to buy and sell goods on their behalf. Chavez & Maes [3], for example, propose a marketplace, where users can create selling and buying agents by giving them a description of the item they want to sell or buy. The main goal of the approach is to help users in the negotiations between buyers and sellers, and to sell the goods better (i.e., at a higher price) than the user would be able to otherwise, by taking advantage of their processing speed and communication bandwidth. Chavez & Maes' agents are (*i*) proactive: "...they try to sell themselves, by going into a marketplace, contacting interested parties (namely, buying agents) and negotiating with them to find the best deal", and (*ii*) autonomous: "...once released into the marketplace, they negotiate and make decisions on their own, without requiring user intervention". Chavez & Maes point out their agents' lack for rationality: "Our experiment demonstrated the need and desire for 'smarter' agents whose decision making processes more closely mimic those of people and which can be directed at a more abstract, motivational level."

In this paper we propose a combination of (a version of) the RPC programming paradigm and KS-agents. In the resulting framework reactive, rational or hybrid agents can reason about their own beliefs as well as the beliefs of other agents and can communicate proactively with each other. In such a framework, the agents' behaviour can be regulated by condition-action rules such as: if I am asked by another, friendly agent about something and I can prove it from my beliefs then I will tell the agent about it.

In the proposed approach, the two primitives for communication in RPC, *tell* and *told*, are treated as abducibles within the cycle of the KS-agent architecture.

The remainder of the paper is structured as follows. In section 2 we give some basic definitions (in particular, we define ordinary and abductive logic programs) and we review the IFF proof procedure. In section 3 we review the KS-agent architecture. In section 4 we review RPC. In sections 5 and 6 we define our approach to introspection and communication in agents, respectively. In section 7 we illustrate the framework by means of an example. In section 8 we conclude and discuss future work.

2 Preliminaries

2.1 Basic definitions

A **logic program** is a set of **clauses** of the form:

$A \leftarrow L_1 \wedge \ldots \wedge L_n \qquad (n \geq 0)$
where every L_i $(1 \leq i \leq n)$ is a literal, A is an atom and all variables are implicitly universally quantified, with scope the entire clause. If $A = p(t)$, with t a vector of terms, then the clause is said to *define* p.

The **completion of a predicate** p [4] defined in a given logic program P by the set of clauses:

$p(t_1) \leftarrow D_1 \quad \ldots \quad p(t_k) \leftarrow D_k \qquad\qquad (k \geq 1)$

is the **iff-definition**:

$p(x) \leftrightarrow [x = t_1 \wedge D_1] \vee \ldots \vee [x = t_k \wedge D_k]$

where x is a vector of variables, all implicitly universally quantified, with scope the entire iff-definition. Any variable in a disjunct D_i which is not in x is implicitly existentially quantified, with scope the disjunct.

If p is not defined in P, then the completion of p is the iff-definition $p(x) \leftrightarrow$ *false*.

The **selective completion** $comp_S(P)$ of a logic program P with respect to a set S of predicates of the language of P is the union of the completions of all predicates in S. Note that the completion of a logic program P [4] is the selective completion of P with respect to all predicates of the language of P, together with Clark's equality theory [4].

An **integrity constraint** is an implication of the form:

$L_1 \wedge \ldots \wedge L_n \Rightarrow A \qquad (n \geq 0)$

where $L_1, \ldots, L_n$ are literals and A is an atom, possibly *false*. All variables in an integrity constraint are implicitly universally quantified, with scope the entire integrity constraint.

An **abductive logic program** [10] is a triple $\langle P, \mathcal{A}, I \rangle$, where P is a logic program, $\mathcal{A}$ a set of predicates in the language of P, and I a set of integrity constraints. The predicates in $\mathcal{A}$ are referred to as the **abducible predicates** and the atoms built from the abducible predicates are referred to as **abducibles**. $\overline{\mathcal{A}}$ is the set of non-abducible predicates in the language of P.

Without loss of generality (see [10]), we can assume that abducible predicates have no definitions in P. Abducibles can be thought of as hypotheses that can be used to extend the given logic program in order to provide an "explanation" for given queries (or observations). Explanations are required to "satisfy" the integrity constraints. Different notions of explanation and satisfaction have been used in the literature. The simplest notion of satisfaction is consistency of the explanation with the program and the integrity constraints.

In the sequel, constant, function and predicate symbols may be written as any sequence of characters in `typewriter` style. Variables may be written as any sequence of characters in *italic* style starting with a lower-case character. Thus, for example, `buys` may be a predicate symbol, `Tom` may be a constant symbol, while x and *city* are variables.

Example 1. Let the abductive logic program $\langle P, \mathcal{A}, I \rangle$ be given as follows

$$P = \left\{ \begin{array}{l} \mathtt{has}(x,y) \leftarrow \mathtt{buys}(x,y) \\ \mathtt{has}(x,y) \leftarrow \mathtt{steals}(x,y) \\ \mathtt{honest(Tom)} \end{array} \right\}$$

$$\mathcal{A} = \left\{ \mathtt{buys}, \mathtt{steals} \right\}$$

$$I = \left\{ \mathtt{honest}(x) \wedge \mathtt{steals}(x,y) \Rightarrow \mathtt{false} \right\}.$$

Then, given the observation $G = \mathtt{has}(\mathtt{Tom}, \mathtt{computer})$, the set of abducibles $\{\mathtt{buys}(\mathtt{Tom}, \mathtt{computer})\}$ is an explanation for G, satisfying I, whereas the set $\{\mathtt{steals}(\mathtt{Tom}, \mathtt{computer})\}$ is not, because it is inconsistent with I.

2.2 The IFF proof procedure

The IFF proof procedure [9] is a rewriting procedure, consisting of a number of inference rules, each of which replaces a formula by one which is equivalent to it in a theory T of iff-definitions. We assume $T = comp_{\overline{\mathcal{A}}}(P)$, for some given abductive logic program $\langle P, \mathcal{A}, I \rangle$. The basic inference rules are: [1]

1. **Unfolding**: given an atom $\mathtt{p}(t)$ and a definition
 $\mathtt{p}(x) \leftrightarrow D_1 \vee \ldots \vee D_n$ in T,
$\mathtt{p}(t)$ is replaced by $(D_1 \vee \ldots \vee D_n)\theta$, where θ is the substitution $\{x/t\}$.
2. **Propagation**: given an atom $\mathtt{p}(s)$ and an integrity constraint
 $L_1 \wedge \ldots \mathtt{p}(t) \ldots \wedge L_n \Rightarrow A$,
a new integrity constraint $L_1 \wedge \ldots t = s \ldots \wedge L_n \Rightarrow A$ is added.
3. **Logical simplification**:
 $[B \vee C] \wedge E$ is replaced by $[B \wedge E] \vee [C \wedge E]$ (**splitting**)
 $\mathtt{not}\, A \wedge B \Rightarrow C$ is replaced by $B \Rightarrow C \vee A$ (**negation elimination**)
 $B \wedge \mathtt{false}$ is replaced by $\mathtt{false}$, $B \vee \mathtt{false}$ is replaced by B, and so on.
4. **Equality rewriting**: applies equality rewrite rules (see [9]) simulating the unification algorithm of [16] and the application of substitutions.

Given an **initial goal** G (a conjunction of literals, whose variables are free), a **derivation** for G is a sequence of formulae $F_1 = G \wedge I, \ldots, F_m$ such that each **derived goal** F_{i+1} is obtained from F_i by applying one of the inference rules, as follows: unfolding - to atoms that are either conjuncts in F_i or conjuncts in bodies of integrity constraints in F_i; propagation - to atoms that are conjuncts in F_i and integrity constraints in F_i; equality rewriting and logical simplification. Every negative literal $\mathtt{not}\, A$ as a conjunct in the initial goal as well as in any derived goal is rewritten as an integrity constraint $A \Rightarrow \mathtt{false}$.

Every derivation relies upon some control strategy. Some strategies are preferable to others. E.g., splitting should always be postponed as long as possible, because it is an explosive operation.

[1] The full IFF proof procedure includes two additional inference rules, case analysis and factoring. In this paper, we omit these rules for simplicity.

Let $F_1 = G \wedge I, \ldots, F_n = N \vee Rest$ be a derivation for G such that $N \neq \texttt{false}$, N is some conjunction of literals and integrity constraints, and no inference step can be applied to N which has not already been applied earlier in the derivation. Then, $F_1, \ldots, F_n$ is a **successful derivation**. An **answer extracted from** N is a pair $(\mathcal{D}, \sigma)$ such that

- σ' is a substitution replacing all free and existentially quantified variables in N by variable-free terms in the underlying language and σ' satisfies all equalities and disequalities in N,
- $\mathcal{D}$ is the set of all abducible atoms that are conjuncts in $N\sigma'$ and σ is the restriction of σ' to the variables in G.

The full proof procedure is proven sound and complete with respect to a given semantics [9], based on Kunen's three-valued completion semantics.

Example 2. Let $\langle P, \mathcal{A}, I \rangle$ be the abductive logic program in example 1. Then, $comp_{\overline{\mathcal{A}}}(P)$ is

$$\left\{ \begin{array}{l} \texttt{has}(x,y) \leftrightarrow \texttt{buys}(x,y) \vee \texttt{steals}(x,y) \\ \texttt{honest}(x) \leftrightarrow x = \texttt{Tom} \end{array} \right\}.$$

The following is a (successful) derivation for the goal $G = \texttt{has}(\texttt{Tom}, \texttt{computer})$:

$F_1 = G \wedge I$

$F_2 = G \wedge [x = \texttt{Tom} \wedge \texttt{steals}(x,y) \Rightarrow \texttt{false}]$ (by unfolding)

$F_3 = G \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}]$ (by equality rewriting)

$F_4 = [\texttt{buys}(\texttt{Tom},\texttt{computer}) \vee \texttt{steals}(\texttt{Tom},\texttt{computer})] \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}]$ (by unfolding)

$F_5 = [\texttt{buys}(\texttt{Tom},\texttt{computer}) \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}]] \vee [\texttt{steals}(\texttt{Tom},\texttt{computer}) \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}]].$ (by splitting)

The answer $(\mathcal{D} = \{\texttt{buys}(\texttt{Tom},\texttt{computer})\}, \sigma = \{\})$ can be extracted from the first disjunct of F_5. An additional successful derivation is $F_1, \ldots, F_9$ with

$F_6 = [\texttt{buys}(\texttt{Tom},\texttt{computer}) \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}]] \vee [\texttt{steals}(\texttt{Tom},\texttt{computer}) \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}] \wedge [y = \texttt{computer} \Rightarrow \texttt{false}]]$ (by propagation)

$F_7 = [\texttt{buys}(\texttt{Tom},\texttt{computer}) \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}]] \vee [\texttt{steals}(\texttt{Tom},\texttt{computer}) \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}] \wedge \texttt{false}]]$ (by equality rewriting)

$F_8 = [\texttt{buys}(\texttt{Tom},\texttt{computer}) \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}]] \vee \texttt{false}$ (by logical simplification)

$F_9 = [\texttt{buys}(\texttt{Tom},\texttt{computer}) \wedge [\texttt{steals}(\texttt{Tom},y) \Rightarrow \texttt{false}]]$ (by logical simplification)

from which the same answer $(\mathcal{D}, \sigma)$ as above can be extracted.

Note that in this example, unfolding and equality rewriting in the conditions of the integrity constraint are performed before any other operation. In general, many of the operations involving the integrity constraints can be done at compile time, and the simplified (more efficient) version of the integrity constraint conjoined to the initial goal.

3 Kowalski-Sadri agents

Every KS-agent can be thought of as an abductive logic program, equipped with an initial goal. The abducibles are *actions* to be executed as well as *observations* to be performed. Updates, observations, requests and queries are treated uniformly as goals. The abductive logic program can be a temporal theory. For example, the event calculus [13] can be written as an abductive logic program:

$$\begin{aligned}\texttt{holds\_at}(p,t_2) \leftarrow\ & \texttt{happens}(e,t_1) \wedge (t_1 < t_2) \wedge \\ & \texttt{initiates}(e,p) \wedge \texttt{not}\ \texttt{broken}(t_1,p,t_2)\end{aligned}$$

$$\texttt{broken}(t_1,p,t_2) \leftarrow \texttt{happens}(e,t) \wedge \texttt{terminates}(e,p) \wedge (t_1 < t < t_2).$$

The first clause expresses that a property p holds at some time t_2 if it is initiated by an event e at some earlier time t_1 and is not broken (i.e. persists) from t_1 to t_2. The second clause expresses that a property p is broken (i.e. does not persist) from a time t_1 to a later time t_2 if an event e that terminates p happens at a time t between t_1 and t_2.

The predicate `happens` is abducible, and can be used to represent both observations, as events that have taken place in the past, or events scheduled to take place in the future. An integrity constraint

$$I_1 \quad \texttt{happens}(e,t) \wedge \texttt{preconditions}(e,t,p) \wedge \texttt{not}\ \texttt{holds\_at}(p,t) \Rightarrow \texttt{false}$$

expresses that an event e cannot happen at a time t if the preconditions p of e do not hold at time t.

The predicates `preconditions`, `initiates` and `terminates` have application-specific definitions, e.g.

$$\texttt{preconditions}(\texttt{carry\_umbrella},t,p) \leftarrow p = \texttt{own\_umbrella}$$

$$\texttt{preconditions}(\texttt{carry\_umbrella},t,p) \leftarrow p = \texttt{borrowed\_umbrella}$$

$$\texttt{initiates}(\texttt{rain},\texttt{raining})$$

$$\texttt{terminates}(\texttt{sun},\texttt{raining}).$$

Additional integrity constraints might be given to represent reactive behaviour of intelligent agents, e.g.

$$I_2 \quad \texttt{happens}(\texttt{raining},t) \Rightarrow \texttt{happens}(\texttt{carry\_umbrella},t+1)$$

or to prevent concurrent execution of actions (events)

$$\texttt{happens}(e_1,t) \wedge \texttt{happens}(e_2,t) \Rightarrow e_1 = e_2.$$

The basic "engine" of a KS-agent is the IFF proof procedure, executed via the following cycle:

To cycle at time t,

(i) observe any input at time t,
(ii) record any such input,
(iii) resume the IFF procedure by propagating the inputs,
(iv) continue applying the IFF procedure,
using for steps (iii) and (iv) a total of r units of time,
(v) select an atomic action which can be executed at time $t + r + 2$,
(vi) execute the selected action at time $t + r + 2$ and record the result,
(vii) cycle at time $t + r + 3$.

The cycle starts at time t by observing and recording any inputs from the environment (steps (i) and (ii)). Steps (i) and (ii) are assumed to take one unit of time each. Then, the proof procedure is applied for r units of time (steps (iii) and (iv)). The amount of resources r available in steps (iii) and (iv) is bounded by some predefined amount n. By decreasing n the agent is more *reactive*, by increasing n the agent is more *rational*. Propagation is applied first (step (iii)), in order to allow for an appropriate reaction to the inputs. Afterwards, an action is selected and executed, taking care of recording the result (steps (v) and (vi)). Steps (v) and (vi) conjoined are assumed to take one unit of time. Selected actions can be thought of as outputs into the environment, and observations as inputs from the environment. ¿From every agent's viewpoint, the environment contains all other agents.

Selected actions correspond to abducibles in an answer extracted from a disjunct in a derived goal in a derivation. The disjunct represents an *intention*, i.e. a (possibly partial) plan executed in stages. A sensible action selection strategy may select actions from the same disjunct (intention) at different iterations of the cycle. Failure of a selected plan is obtained via logical simplification, after having propagated `false` into the selected disjunct.

Actions that are generated in an intention may have times associated with them. The times may be absolute, for example $\texttt{happens}(\texttt{ring\_bell}, 3)$, or may be within a constrained range, for example $\texttt{happens}(\texttt{step\_forward}, t) \wedge (1 < t < 10)$. In step (v), the selected action will either have an absolute time equal to $t + r + 2$ or a time range compatible with an execution time at $t + r + 2$. In the latter case, recording of the result of the execution instantiates the time of the action.

Integrity constraints provide a mechanism not only for constraining explanations and plans, for example, as in I_1, but also for allowing reactive, condition-action type of behaviour, for example, as in I_2.

4 Reflective Prolog with communication

4.1 Reflective Prolog

Reflective Prolog (RP) [7, 8] is a metalogic programming language that extends the language of Horn clauses [11, 15] to include higher-order-like features.

The language is that of Horn clauses except that terms are defined differently in order to include *names* that are intended to represent at the meta-level the expressions of the language itself. The alphabet of RP differs from the usual alphabet of Horn clauses by making a distinction between variables and *metavariables* and by the presence of *metaconstants* in RP. Metavariables can only be substituted with names of sentences of RP, and metaconstants are intended as names for constants, function and predicate symbols of RP. If `c` is a constant, a function or a predicate symbol, then we write $'\mathtt{c}$ as a convenient way to represent the metaconstant that names `c`. Similarly, if $'\mathtt{c}$ is a metaconstant, then its name is $''\mathtt{c}$, and so on.

Furthermore, the alphabet of RP contains the (unary) predicate symbol `solve`. This allows us to extend at the *meta-level* the intended meaning of predicates (partially) defined at the *object-level*. Clauses defining `solve` will be referred to as *meta-level clauses*, and clauses defining object-level predicates will be referred to as *object-level clauses*. Metavariables are written as any sequences of characters in *italic* style starting in the upper-case.

Compound terms and atoms are represented at the meta-level as *name terms*. For example, the name of the term $\mathtt{f}(\mathtt{a}, x)$ is the name term $'\mathtt{f}('\mathtt{a}, 'x)$, where $'\mathtt{f}$ and $'\mathtt{a}$ are the metaconstants that name the function symbol `f` and constant `a`, respectively, and $'x$ stands for the name of the value of the variable x.

The intended connection between the object-level and the meta-level of RP is obtained by means of the following (inter-level) reflection axioms:

$A \leftarrow \mathtt{solve}('A)$ and $\mathtt{solve}('A) \leftarrow A$, for all atoms A.

The first asserts that whenever an atom of the form $\mathtt{solve}('A)$ is provable at the meta-level, then A is provable at the object-level. These axiom schemata are not explicitly present in any given program but rather they are simulated within the modified SLD-resolution underlying RP.

Suppose, for example, that we want to express the fact that an object `obj` satisfies all the relations in a given `class`. If we want to formalise that statement at the object-level, then for every predicate `q` in `class` we have to write the clause `q(obj)`. Instead, we may formalise our statement at the meta-level as:

$\mathtt{solve}(P('\mathtt{obj})) \leftarrow \mathtt{belongs\_to}(P, \mathtt{class})$,

where P is a metavariable ranging over the names of predicate symbols.

4.2 Communication

RPC is an extension of Reflective Prolog to accommodate communication between agents [7, 8]. Agents are seen as logic programs. In an agent setting, `solve` may be seen as representing a given agent's beliefs, for example $\mathtt{agent}_1 : \mathtt{solve}('A)$, for some atom A, stands for "$\mathtt{agent}_1$ believes A". RPC allows for two communication acts, expressed via the two meta-predicates $\mathtt{tell}(X, Y)$ and $\mathtt{told}(X, Y)$. An atom $\mathtt{tell}('\mathtt{agent}_2, 'A)$ in the theory representing $\mathtt{agent}_1$ stands for "$\mathtt{agent}_1$ tells $\mathtt{agent}_2$ that A holds". An atom $\mathtt{told}('\mathtt{agent}_2, 'A)$ in the theory representing $\mathtt{agent}_1$ stands for "$\mathtt{agent}_1$ is told by $\mathtt{agent}_2$ that A holds". The latter atom can also be interpreted as "$\mathtt{agent}_1$ *asks* $\mathtt{agent}_2$ whether A holds", i.e. the predicate `told` can be used to express queries of agents to other agents.

Communication acts are formalized in RPC by means of (inter-theory) reflection axioms based on the predicate symbols `tell` and `told`:

$[\mathtt{agent}_1 : \mathtt{told}('\mathtt{agent}_2,'A)] \leftarrow [\mathtt{agent}_2 : \mathtt{tell}('\mathtt{agent}_1,'A)]$

for all atoms A, and agent names $'\mathtt{agent}_1$ and $'\mathtt{agent}_2$. The intuitive meaning of each such axiom is that every time an atom of the form $\mathtt{tell}('\mathtt{agent}_1,'A)$ can be derived from a theory $\mathtt{agent}_2$ (which means that $\mathtt{agent}_2$ wants to communicate proposition A to $\mathtt{agent}_1$), the atom $\mathtt{told}('\mathtt{agent}_2,'A)$ is consequently derived in the theory $\mathtt{agent}_1$ (which means that proposition A becomes available to $\mathtt{agent}_1$). These axioms are not explicitly present in any given program but rather they are simulated within the modified SLD-resolution underlying RPC.

These two predicates are intended to model the simplest and most neutral form of communication among agents, with no implication about provability (or truth) of what is communicated, and no commitment about how much of its information an agent communicates and to whom. An agent may communicate to another agent everything it can derive (in its associated theory), or only part of what it can derive, or it may even lie, that is, it communicates something it cannot derive.

The intended connection between `tell` and `told` is that an agent may receive from another agent (by means of `told`) only the information the second agent has explicitly addressed to it (by means of `tell`). Thus, an agent can regulate its interaction with other agents by means of appropriate clauses defining the predicate `tell`.

What use an agent makes of any information given to it by others is entirely up to the agent itself. Thus, the way an agent communicates with others is not hard-wired in the language. Rather, it is possible to define in a program different behaviours for different agents or different behaviours of one agent in different situations. (Several examples of the use of agents in RPC can be found in [1, 6].) For example, the following clauses:

$\mathtt{Bob} : [\mathtt{solve}(X) \leftarrow \mathtt{reliable}(A) \wedge \mathtt{told}(A, X)]$

$\mathtt{Bob} : [\mathtt{solve}(X) \leftarrow \mathtt{told}('\mathtt{John},\ '\mathtt{not}\ X)]$

express that an agent `Bob` trusts every agent that is reliable but distrusts `John`. The clause:

$\mathtt{Bob} : [\mathtt{tell}(A, X) \leftarrow \mathtt{agent}(A) \wedge \mathtt{solve}(X)]$

says that the agent `Bob` tells the others whatever he can prove.

RP and its extension RPC rely upon a Prolog-style control strategy and therefore do not allow for agents to tell other agents or ask for information proactively.

5 Adding introspection to KS-agents

In this paper, following the framework of KS-agents, agents are represented as abductive logic programs rather than ordinary logic programs as in RPC. The abductive logic program, in its completed form, is used embedded within a KS-agents' cycle.

In order to accommodate within agents beliefs about different agents, we give solve an additional argument rather than using labels as in RPC. The atom $\texttt{solve}(Agent, A)$ stands for "$Agent$ believes A". We will assume that solve can only take names of atoms as second argument. By convention, $\texttt{solve}(A)$ will be an abbreviation for $\texttt{solve}(Agent, A)$ within the program P of $Agent$ itself.

Instead of incorporating the reflection principle linking object- and meta-level within the proof procedure, as in RP, we incorporate it via (a finite set of) clauses to be added to the given (abductive) logic program. (A similar approach is presented in [2].)

Let $\langle P, \mathcal{A}, I\rangle$ be an abductive logic program and let $C = H \leftarrow B$ be a clause in P. Then, we define a reflection principle $\mathcal{RP}$:

$$\mathcal{RP}(C) = \begin{cases} \mathcal{O}(C) & \text{if } C \text{ is an object-level clause} \\ \mathcal{M}(C) & \text{if } C \text{ is a meta-level clause} \end{cases}$$

where

- If $H = \texttt{p}(t)$, then

$$\mathcal{O}(C) = \left\{ \texttt{solve}(X) \leftarrow X = \,'\!\texttt{p}('t) \wedge B \right\}.$$

Note that necessarily $\texttt{p} \notin \mathcal{A}$, since we assume, without loss of generality, that abducible predicates are not defined in P.

- If $H = \texttt{solve}('\texttt{p}('t))$ and $\texttt{p} \notin \mathcal{A}$, then

$$\mathcal{M}(C) = \{\texttt{p}(x) \leftarrow x = t \wedge B\}.$$

- If $H = \texttt{solve}(X('t))$, then

$$\mathcal{M}(C) = \left\{ \texttt{p}(t) \leftarrow X = \,'\!\texttt{p} \wedge B \mid \text{for every}\, \texttt{p} \notin \mathcal{A} \text{ and } \texttt{p} \neq \texttt{solve} \right\}.$$

- If $H = \texttt{solve}(X)$, then

$$\mathcal{M}(C) = \left\{ \texttt{p}(x) \leftarrow X = \,'\!\texttt{p}('x) \wedge B \mid \text{for every}\, \texttt{p} \notin \mathcal{A} \text{ and } \texttt{p} \neq \texttt{solve} \right\}.$$

$\mathcal{RP}(P)$ is given by the union of $\mathcal{RP}(C)$ for all $C \in P$.

► Let $\langle P, \mathcal{A}, I\rangle$ be an abductive logic program. The **abducibility set** of $\mathcal{A}$, written as $\Gamma(\mathcal{A})$, is the set of clauses:

$$\Gamma(\mathcal{A}) = \left\{ \texttt{solve}(X) \leftarrow X = \,'\!\texttt{a}('x) \wedge \texttt{a}(x) \mid \text{for all } \texttt{a} \in \mathcal{A} \right\}.$$

The abducibility set of an abductive logic program allows a meta-level representation of provability of abducible predicates.

The intended connection among object-level, meta-level and abducible atoms is captured by the following definition.

► Let $\langle P, \mathcal{A}, I\rangle$ be an abductive logic program.
The **associated program** of P with respect to $\mathcal{A}$, written as $\Delta(P, \mathcal{A})$, is defined as:

$$\Delta(P, \mathcal{A}) = P \cup \mathcal{RP}(P) \cup \Gamma(\mathcal{A}).$$

The **associated integrity constraints** of I with respect to $\mathcal{A}$, written as $\Delta(I, \mathcal{A})$, is defined as:

$$\Delta(I, \mathcal{A}) = I \cup \{\texttt{solve}('\texttt{p}('x)) \Rightarrow \texttt{p}(x) \mid \text{for all } \texttt{p} \in \mathcal{A}\}.$$

The **meta-abductive logic program** associated with $\langle P, \mathcal{A}, I\rangle$ is:

$$\langle \Delta(P, \mathcal{A}),\ \mathcal{A},\ \Delta(I, \mathcal{A})\rangle.$$

The addition of the new integrity constraints in $\Delta(I, \mathcal{A})$ allows the agent to propagate (and thus compute the consequences of) any new information it receives about abducible predicates in whatever (meta-level or object-level) form, without any need to alter the original set of integrity constraints, I, or the program P.

6 Adding communication to KS-agents

In this paper, we interpret $\texttt{tell}(X, Y)$ and $\texttt{told}(X, Y)$ as abducible predicates in meta-abductive logic programs. As for `solve` we can give `tell` and `told` an additional argument instead of introducing labels, to represent communication between agents. For simplicity, we will abbreviate $\texttt{tell}(\mathit{Agent}, X, Y)$ (resp. `told`) within the program P of *Agent* itself as $\texttt{tell}(X, Y)$. As with `solve`, we will assume that `tell` and `told` take only names of atoms as their last argument.

Example 3. Let $\texttt{agent}_1$ be represented by the abductive logic program $\langle P, \mathcal{A}, I\rangle$ with:

$$P = \left\{ \begin{array}{l} \texttt{solve}(X) \leftarrow \texttt{told}(A, X) \\ \texttt{desire}(y) \leftarrow y = \texttt{car} \\ \texttt{good\_price}(p, x) \leftarrow p = 0 \end{array} \right\}$$

$$\mathcal{A} = \{\texttt{tell}, \texttt{told}, \texttt{offer}\}$$

$$I = \{\texttt{desire}(x) \wedge \texttt{told}(A, '\texttt{good\_price}('p, 'x)) \Rightarrow \texttt{tell}(A, '\texttt{offer}('p, 'x))\}.$$

Namely, $\texttt{agent}_1$ believes anything it is told (by any other agent), and it desires to have a car. The third clause in P says that anything that is free is at a good price. Moreover, if the agent desires something and it is told (by some other agent) of a good price for it, then it makes an offer to the other agent, by telling it. Note that a more accurate representation of the integrity constraint should include time.

The corresponding meta-abductive logic program is

$$\Delta(P,\mathcal{A}) = P \cup \left\{ \begin{array}{l} \texttt{solve}(X) \leftarrow X = {}'\texttt{desire}('y) \wedge y = \texttt{car} \\ \texttt{solve}(X) \leftarrow X = {}'\texttt{good\_price}('p,'x) \wedge p = 0 \\ \texttt{desire}(x) \leftarrow X = {}'\texttt{desire}('x) \wedge \texttt{told}(A,X) \\ \texttt{good\_price}(p,x) \leftarrow X = {}'\texttt{good\_price}('p,'x) \wedge \texttt{told}(A,X) \\ \texttt{solve}(Y) \leftarrow Y = {}'\texttt{tell}('A,'X) \wedge \texttt{tell}(A,X) \\ \texttt{solve}(Y) \leftarrow Y = {}'\texttt{told}('A,'X) \wedge \texttt{told}(A,X) \\ \texttt{solve}(Y) \leftarrow Y = {}'\texttt{offer}('p,'x) \wedge \texttt{offer}(p,x) \end{array} \right\}$$

$$\mathcal{A} \quad = \{\, \texttt{tell}, \texttt{told}, \texttt{offer} \,\}$$

$$\Delta(I,\mathcal{A}) \; = I \cup \left\{ \begin{array}{l} \texttt{solve}('\texttt{tell}('A,'X)) \Rightarrow \texttt{tell}(A,X) \\ \texttt{solve}('\texttt{told}('A,'X)) \Rightarrow \texttt{told}(A,X) \\ \texttt{solve}('\texttt{offer}('p,'x)) \Rightarrow \texttt{offer}(p,x) \end{array} \right\}.$$

Given an abductive logic program $\langle P, \mathcal{A}, I \rangle$, corresponding to some agent, and the associated meta-abductive logic program $\langle \Delta(P,\mathcal{A}), \mathcal{A}, \Delta(I,\mathcal{A}) \rangle$, the agent's cycle applies the IFF procedure with

$$T = comp_{\overline{\mathcal{A}}}(\Delta(P,\mathcal{A})).$$

The equality rewriting rule needs to be modified to take into account equality between names.

Example 4. Let $\langle P, \mathcal{A}, I \rangle$ be the abductive logic program of example 3. Assume that $\texttt{agent}_1$ has the input observation:

$$\texttt{told}('\texttt{agent}_2,'\texttt{good\_price}('50,'\texttt{car})),$$

meaning that $\texttt{agent}_1$ has been told by $\texttt{agent}_2$ of a good price (of 50) for a car. Then, the initial goal G is:

$$\texttt{told}('\texttt{agent}_2,'\texttt{good\_price}('50,'\texttt{car})).$$

A computed answer for G is

$$\begin{array}{l} \mathcal{D} = \{\texttt{told}('\texttt{agent}_2,'\texttt{good\_price}('50,'\texttt{car})), \texttt{tell}('\texttt{agent}_2,'\texttt{offer}('50,'\texttt{car}))\} \\ \sigma \; = \{\}. \end{array}$$

Within the KS-agent architecture, the following requirements are met:

- KS-agents are known by their symbolic names.
- When a KS-agent sends a message, it directs that message to a specific addressee.
- When a KS-agent receives a message, it knows the sender of that message.
- Messages may get lost.

Both $\texttt{tell}$ and $\texttt{told}$ are treated as actions: everytime (the cycle of) an agent $\texttt{agent}_1$ selects a communicative action, i.e., an action of the form

$$\texttt{told}('\texttt{agent}_2, A) \qquad \text{or} \qquad \texttt{tell}('\texttt{agent}_2, A),$$

$\texttt{agent}_1$ will attempt to execute it. If the attempt is successful, the record

$$\texttt{told}('\texttt{agent}_2, A) \qquad \text{or} \qquad \texttt{tell}('\texttt{agent}_2, A)$$

is conjoined to the goals of agent $\texttt{agent}_1$, as an additional input. If the attempt is not successful, the record

$$\texttt{told}('\texttt{agent}_2, A) \Rightarrow \texttt{false} \qquad \text{or} \qquad \texttt{tell}('\texttt{agent}_2, A) \Rightarrow \texttt{false}$$

is conjoined to the goals of agent $\texttt{agent}_1$, as an additional input. In the next section we will formalise an example showing how proactive communication is achieved by executing the proof procedure within cycle.

7 Example

The following example demonstrates the running of the proof procedure within cycle, action selection, proactive communication whereby one agent volunteers information to another, and how during the planning phase such information can help in the choice of intention.

The example is as follows: an agent wishes to register for a conference, let us say Jelia, that is to take place in Paris on the 10th and to make travel arrangements to go to Paris on the 10th (for simplicity we omit the month). So the agent's original goal is the conjunction:

$\texttt{register(Jelia)} \wedge \texttt{travel(Paris, 10)}$.

The agent has the following program, P:

C_1 $\quad \texttt{travel}(\mathit{city}, \mathit{date}) \leftarrow \texttt{go}(\mathit{city}, \mathit{date}, \texttt{train})$

C_2 $\quad \texttt{travel}(\mathit{city}, \mathit{date}) \leftarrow \texttt{go}(\mathit{city}, \mathit{date}, \texttt{plane})$

C_3 $\quad \texttt{early\_registration(Jelia)} \leftarrow \texttt{send\_form}(\texttt{Jelia}, \mathit{date}) \wedge (\mathit{date} < 3)$

C_4 $\quad \texttt{go}(\mathit{city}, \mathit{date}, \mathit{means}) \leftarrow \texttt{book}(\mathit{city}, \mathit{date}, \mathit{means})$

C_5 $\quad \texttt{book}(\mathit{city}, \mathit{date}, \mathit{means}) \leftarrow$
$\qquad \texttt{told}('\texttt{ticket\_agent}, '\texttt{available}('\mathit{city}, '\mathit{date}, '\mathit{means})) \wedge$
$\qquad \texttt{tell}('\texttt{ticket\_agent}, '\texttt{reserve}('\mathit{city}, '\mathit{date}, '\mathit{means}))$

C_6 $\quad \texttt{solve}(X) \leftarrow \texttt{told}('\texttt{ticket\_agent}, X)$.

Clauses C_1 and C_2 say that one travels to a *city* on a given *date* if one goes there on that *date* by train or by plane. C_3 says that the deadline for early registration for Jelia is the 3rd. C_4 says that one goes to a *city* on a given *date* by some *means* if one makes a booking for that journey. C_5 says that one makes a booking if the ticket_agent confirms availability of ticket and one makes a reservation. Note that in a more thorough representation we would represent the transaction time of when a booking is made (which must be before the time of travel). In that case we will have an extra argument in tell and told that represents such transaction times. We will ignore this issue here for simplicity.

The agent has the following set of integrity constraints I:

$$I_1 \quad \texttt{register}(\mathit{conference}) \Rightarrow \texttt{early\_registration}(\mathit{conference})$$
$$I_2 \quad \texttt{go}(\mathit{city}, \mathit{date}, \mathit{means}) \wedge \texttt{strike}(\mathit{city}, \mathit{date}, \mathit{means}) \Rightarrow \texttt{false}.$$

I_1 states the agent's departmental policy that anyone registering at a conference must take advantage of early registration. I_2 states that one cannot use a mode of transport which is subject to a strike.

Finally, the agent has the following set of abducibles $\mathcal{A}$:

$\{\texttt{send\_form}, \texttt{tell}, \texttt{told}, \texttt{register}, \texttt{available}, \texttt{reserve}, \texttt{strike}\}$.

Now suppose that the cycle of the agent starts at time 1 and that the agent does not observe any input. Thus, the cycle effectively starts at step (iv) by applying the IFF proof procedure (using $\langle \Delta(P, \mathcal{A}), \mathcal{A}, \Delta(I, \mathcal{A})\rangle$ which we do not show here) to the goal obtained by conjoining the original goal and the integrity constraints $\Delta(I, \mathcal{A})$. By repeatedly applying the IFF proof procedure, the goal is transformed (within the initial cycle or within some later iteration, depending on the resource parameter r) into

$\texttt{register}(\texttt{Jelia}) \wedge \texttt{travel}(\texttt{Paris}, 10) \wedge \texttt{send\_form}(\texttt{Jelia}, \mathit{date}) \wedge (\mathit{date} < 3)$.

At this point cycle can select (at step (v)) the action $\texttt{send\_form}(\texttt{Jelia}, \mathit{date})$ to perform if time is less than 3. If the agent has been too slow and time 3 has already passed the agent has failed its goals. Suppose the agent succeeds. Note that at any time any other agent can send this agent a message. So suppose the ticket agent sends a message that trains are on strike in Paris on the 10th, i.e. at some iteration of cycle at step (i) the agent receives the following input

$\texttt{told}('\texttt{ticket\_agent}, '\texttt{strike}('\texttt{Paris}, '10, '\texttt{train}))$.

In that iteration of cycle this information is propagated (step (iii)) and the simplified constraint

$\texttt{go}(\texttt{Paris}, 10, \texttt{train}) \Rightarrow \texttt{false}$

is added to the goal. Meanwhile the sub-goal $\texttt{travel}(\texttt{Paris}, 10)$ is unfolded (step (iv)) into

$\texttt{go}(\texttt{Paris}, 10, \texttt{train}) \vee \texttt{go}(\texttt{Paris}, 10, \texttt{plane})$.

The information about the strike will be used to remove the first possibility (i.e the first disjunct) leaving only

$\texttt{go}(\texttt{Paris}, 10, \texttt{plane})$

which will become the agent's intention. This will be unfolded into the plan

$\texttt{told}('\texttt{ticket\_agent}, '\texttt{available}('\texttt{Paris}, '10, '\texttt{plane})) \wedge$

$\texttt{tell}('\texttt{ticket\_agent}, '\texttt{reserve}('\texttt{Paris}, '10, '\texttt{plane}))$.

The appropriate actions will be selected during some iterations of cycle.

For lack of space we have ignored the cycle of the ticket_agent.

8 Conclusions

We have presented an approach to agents that can reason about their own beliefs as well as beliefs of other agents and that can communicate with each other. The approach results from the combination of the approach to agents in [12] and the approach to meta-reasoning and communication in [6, 1]. We have illustrated the approach by means of a number of examples.

The approach needs to be extended in a number of ways. For simplicity, we have ignored the treatment of time. However, time plays an important role in most agent applications and should be explicitly taken into account.

We have considered only two communication performatives, `tell` and `told`. Existing communication languages, e.g. [5], consider additional performatives, e.g. `deny`, `achieve` and `unachieve`. We are currently investigating whether some of these additional performatives could be defined via communication protocols, as definitions and integrity constraints within our framework.

The primitive `told` is used to express both *active* request for information and *passive* communication. The two roles should be separated out, possibly with the addition of a third predicate `ask`, distinguished from `told`. Then the predicate `told` could be defined in terms of `ask` and `tell`, rather than be an abducible. For example, an agent $\mathtt{agent}_1$ is told of X by another agent $\mathtt{agent}_2$ if and only if $\mathtt{agent}_2$ tells $\mathtt{agent}_1$ of X or $\mathtt{agent}_1$ actively asks $\mathtt{agent}_2$ about X and $\mathtt{agent}_2$ gives a positive answer.

We have implicitly assumed that different agents share the same content language. However, this assumption is not essential. Indeed, "translator agents" could be defined, acting as mediators between agents with different content languages. This is possible by virtue of the metalogic features of the language.

Acknowledgments

The authors are grateful to the anonymous referees for their comments. The second and third authors are supported by the UK EPSRC project "Logic-based multi-agent systems".

References

1. J. Barklund, S. Costantini, P. Dell'Acqua, and G. A. Lanzarone. Metareasoning agents for query-answering systems. In Troels Andreasen, Henning Christiansen, and Henrik Legind Larsen, editors, *Flexible Query-Answering Systems*, pages 103–122. Kluwer Academic Publishers, Boston, Mass., 1997.
2. J. Barklund, S. Costantini, P. Dell'Acqua, and G. A. Lanzarone. Reflection Principles in Computational Logic. Submitted to J. of Logic and Computation, 1997.
3. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In B. Crabtree and N. Jennings, editors, *Proc. 1st Intl. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 75–90. The Practical Application Company, 1996.
4. K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*. Plenum Press, New York, 1978.
5. P. R. Cohen and H. J. Levesque. Communicative actions for artificial agents. In V. Lesser, editor, *Proc. 1st Intl. Conf. on Multiagent Systems*, AAAI Press, pages 65–72. MIT Press, 1995.

6. S. Costantini, P. Dell'Acqua, and G. A. Lanzarone. Reflective agents in metalogic programming. In Alberto Pettorossi, editor, *Meta-Programming in Logic*, LNCS 649, pages 135–147, Berlin, 1992. Springer-Verlag.
7. S. Costantini and G. A. Lanzarone. A metalogic programming language. In G. Levi and M. Martelli, editors, *Proc. 6th Intl. Conf. on Logic Programming*, pages 218–33, Cambridge, Mass., 1989. MIT Press.
8. S. Costantini and G. A. Lanzarone. A metalogical programming approach: Language, semantics and applications. *Int. J. of Experimental and Theoretical Artificial Intelligence*, 6:239–287, 1994.
9. T. H. Fung and R. Kowalski. The IFF proof procedure for abductive logic programming. *J. Logic Programming*, 33(2):151–165, 1997.
10. A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. Gabbay, C. Hogger, and A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, UK, 1998.
11. R. A. Kowalski. Predicate logic as a programming language. In J. L. Rosenfeld, editor, *Information Processing, 1974*, pages 569–574, Amsterdam, 1974. North-Holland.
12. R. A. Kowalski and F. Sadri. Towards a unified agent architecture that combines rationality with reactivity. In Dino Pedreschi and Carlo Zaniolo, editors, *Logic in Databases, Intl. Workshop LID'96*, LNCS 1154, pages 137–149, Berlin, 1996. Springer-Verlag.
13. R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.
14. Y. Labrou and T. Finin. Semantics and conversations for an agent communication language. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 234–242, San Francisco, 1997. Morgan Kaufmann.
15. John W. Lloyd. *Foundations of Logic Programming, Second Edition*. Springer-Verlag, Berlin, 1987.
16. Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM TOPLAS*, 4:258–282, 1982.
17. M. P. Singh. Towards a formal theory of communication for multiagent systems. In *Proc. 12th Intl. Joint Conf. on Artificial Intelligence*, pages 69–74, Sydney, Australia, 1991. Morgan Kaufmann.
18. B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Communicating rational agents. In B. Nebel and L. Dreschler-Fischer, editors, *KI-94: Advances in Artificial Intelligence*, LNAI 861, pages 202–213, Berlin, 1994. Springer-Verlag.
19. E. Verharen and F. Dignum. Cooperative information agents and communication. In P. Kandzia and M. Klusch, editors, *Cooperative Information Agents*, LNAI 1202, pages 195–208, Berlin, 1997. Springer-Verlag.
20. G. Wagner. Multi-level security in multiagent systems. In P. Kandzia and M. Klusch, editors, *Cooperative Information Agents*, LNAI 1202, pages 272–285, Berlin, 1997. Springer-Verlag.