Università degli Studi di Bologna
DEIS

An abductive logic programming approach for negotiation and communication in multi-agent systems

Fariba Sadri Francesca Toni Paolo Torroni

April 11, 2002

DEIS Technical Report no. DEIS-LIA-02-001 *LIA Series no. 54*

An abductive logic programming approach for negotiation and communication in multi-agent systems

Fariba Sadri [1] Francesca Toni [1] Paolo Torroni [2]

[1] *Dept. of Computing, Imperial College*
180 Queens Gate
SW7 London, UK
{fs, ft}@ing.unife.it

[2] *DEIS, Università di Bologna*
V.le Risorgimento 2
40136 Bologna, Italy
ptorroni@deis.unibo.it

April 11, 2002

Abstract. In this paper, we present a framework for agent negotiation based on abductive logic programming. The framework extends an existing architecture for logic-based agents by accommodating dialogues for negotiation. We also consider the well-known resource reallocation problem, and show how it can be solved in the proposed framework.

Keywords: *Multi-Agent Systems, Computational Logic, Abduction*

1 Introduction

In multi-agents systems, agents may need to interact in order to exchange resources, when their own are not enough to achieve their goals. Negotiation techniques can be used to solve problems such as resource reallocation, if it is not possible to assume that all agents have a global knowledge about the availability of resources in the environment where they are situated, or if it is not possible to assume that there exists a particular agent that decides the distribution of resources, based on a "global" plan. In fact, there could be no global visibility of the agents' own needs and resources, so that such a "global" plan cannot possibly be generated by a single individual, or the autonomous agent behaviour and the consequent dynamic evolution of the system could make any such plan obsolete before completion.

The focus of this work is the knowledge and the reasoning that is required to build dialogues between negotiating agents, in order to exchange resources that help achieve the agents' objectives. We express such knowledge in a declarative way. In particular, we represent the beliefs of agents as logic programs and integrity constraints, as in abductive logic programming [2], with the dialogue moves treated as abducibles/hypotheses, and dialogue/negotiation policies treated as integrity constraints.

We propose an operational model, and show that it can effectively be used to generate (negotiation) dialogues. We then focus on the resource reallocation problem (r.r.p.), and prove a completeness result by showing how the proposed model can find a solution to the problem, if any solution exists, under the simplifying assumption that during the negotiation process the agents do not change their intentions, and that dialogues are atomic, in that no agent can participate in more than one dialogue at the same time. The framework is based on an agent architecture initially proposed in [3], which we extend an modify to enable dialogue generation. The agent architecture in [3] relies upon executing the IFF abductive proof-procedure defined in [1] as the reasoning engine of the agent. We identify the inadequacy of the IFF for our domain of application, and adopt the procedure in [4] instead.

With respect to our previous work [5, 6], this paper provides a better formalization of the framework for negotiation, and proves some strong results for frameworks where agents that do not change intentions during the negotiation process.

The paper is organized as follows: in Section 2 we introduce the dialogue and negotiation framework, in Section 3 we describe the resource reallocation problem, in Section 4 we define the operational model for the negotiation framework, and in Section 5 we present the formal results. Section 6 concludes.

2 Negotiation framework

In this section, we describe the agent negotiation framework, namely the language for negotiation and the mechanisms to produce individual dialogues and sequences of dialogues.

2.1 Negotiation language

Agents negotiate using a shared language. The language is made of possible utterances, or *dialogue moves*, that are defined as follows:[1]

Definition 1 (*dialogue move*) A *dialogue move* is an instance of a schema

$$tell(X, Y, \mathbf{Subject}, D, T)$$

where X is the *utterer* and Y is the *receiver* of the dialogue move, D is an *identifier* for the dialogue, and T is the *time* when the move is uttered. **Subject** is the *content*

[1] Terms starting with capital or lower-case letters stand for variables or ground terms, respectively.

of the move, expressed in some given *content language*. The identifier of a dialogue uniquely determines the dialogue that includes the dialogue move (see Section 2.3).

A concrete example of a dialogue move is $tell(a, b, \mathbf{request}(\mathbf{give}(nail)), d, 1)$, where **Subject** is $\mathbf{request}(\mathbf{give}(nail))$. Intuitively, this utterance expresses a's request to b at time 1 for a nail, in the course of a dialogue identified by d. We will see that dialogues are composed of moves, that can be seen as abducibles generated by an abductive proof-procedure reasoning with an agent's knowledge base, represented via an abductive logic program.

Definition 2 (*language for negotiation*) A *language for negotiation* $\mathcal{L}$ is a (possibly infinite) set of (possibly non ground) dialogue moves. For a given $\mathcal{L}$, we define two (possibly infinite) subsets of moves, $\mathcal{I}(\mathcal{L})$, $\mathcal{F}(\mathcal{L}) \subseteq \mathcal{L}$, called respectively *initial moves* and *final moves*. Each final move is either *successful* or *unsuccessful*.

A very simple example of language for negotiation is $\mathcal{L}_1$:

$\mathcal{L}_1 = \{\ tell(X, Y, \mathbf{request}(\mathbf{give}(Resource)), D, T),$
$\quad tell(X, Y, \mathbf{accept}(\mathbf{request}(\mathbf{give}(Resource))), D, T)$
$\quad tell(X, Y, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(Resource))), D, T)\ \}$

The initial and final moves are:

$\mathcal{I}(\mathcal{L}_1) = \{tell(X, Y, \mathbf{request}(\mathbf{give}(Resource)), D, T)\}$
$\mathcal{F}(\mathcal{L}_1) = \{\ $ [*succ. moves:*] $tell(X, Y, \mathbf{accept}(\mathbf{request}(\mathbf{give}(Resource))), D, T)$
[*unsucc. moves:*] $tell(X, Y, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(Resource))), D, T)\}$

In this example all moves are either initial or final. This is not the case for a more expressive language, such as the one defined in [6].

Definition 3 (*agent system*) An *agent system* consists of:

- a language for negotiation $\mathcal{L}$;
- a finite set A, containing at least two elements, where each $x \in A$ is a ground term, representing the name of an agent, and each agent is equipped with a *knowledge base*. All elements of A are distinct (agent names must be uniquely identifiable).

In the sequel, we will assume that all agents are equipped with $\mathcal{L}_1$.

2.2 Knowledge base of individual agents

Each agent's knowledge base consists of:

- **domain-dependent beliefs** about the world, such as the information about the agent system;
- **domain-independent beliefs**, used to regulate the negotiation dialogues and to represent changes in the agent's own state, such as its ownership of resources;
- information about the **resources** the agent owns before the dialogues start;
- the agent **goal**;
- the agent's **intention**, i.e., in this paper, the *plan* that the agent intends to carry out in order to achieve its goal; an intention is a tuple $< Plan$, $Needed\_resources$, $Missing\_resources >$, where $Plan$ is a sequence of actions, $Needed\_resources$ is a set of resources needed to perform the actions, and that the agent owns from the start, and $Missing\_resources$ is a set of resources that the agent needs to obtain before the plan becomes feasible;
- the past (time-stamped) **utterances**;

We will refer to the knowledge of agent x at time t as $\mathcal{K}_{x,t}$.[2] For example, the knowledge $\mathcal{K}_{a,0}$ of some agent a at time 0 (standing for the initial time) can be:

- $agents(\{a, b, c\})$;
- all the rules and constraints defined in the following sections (i.e., policy regulations, predicate definitions, and the rules and constraints for the dialogue cycle);
- $have(picture, 0)$;
- $goal(hung(picture))$
- $plan(< hit(nail),\ hang(picture) >, \{picture\}, \{nail, hammer\})$;
- empty: as there has not been a dialogue yet at time 0;

Note that the goal and the intention are not time-stamped. We assume that plans in intentions are given initially and do not change.
At any time the missing resources of an agent can be inferred from the agent knowledge base. In particular, let us consider an agent x with an intention $\mathcal{I}_x$. Let $\mathcal{R}(\mathcal{I}_x)$ be the set of resources required to carry out the plan in $\mathcal{I}_x$ (e.g., $\mathcal{R}(\mathcal{I}_x) = \{picture,$ $nail,\ hammer\}$). Let R be a resource and let $\mathcal{R}_{x,t}$ be the resources the agent owns

[2] We will drop sometimes x and/or t, if clear from the context.

at any time t. Then, we assume that at time t one and only one of the following predicates is entailed by $\mathcal{K}_x$:

- $a(R,t)$: x has R and does not need it ($R \in \mathcal{R}_{x,t} \wedge R \notin \mathcal{R}(\mathcal{I}_x)$)
- $m(R,t)$: x does not have R but needs it($R \notin \mathcal{R}_{x,t} \wedge R \in \mathcal{R}(\mathcal{I}_x)$)
- $n(R,t)$: x does have R and does need it($R \in \mathcal{R}_{x,t} \wedge R \in \mathcal{R}(\mathcal{I}_x)$)
- $i(R,t)$: x does not have R and does not need it($R \notin \mathcal{R}_{x,t} \wedge R \notin \mathcal{R}(\mathcal{I}_x)$)

Intuitively, a, m, n, i stand for *available*, *missing*, *needed*, and *indifferent*, respectively. Formal definitions of "have" and "need" will be given in Section 2.3. We only assume here that resources are considered not to be owned if they have been "promised" to another agent, i.e., as a consequence of the acceptance of another agent's request, even if the actual delivery has not yet been carried out. Indeed, here we are not concerned with the execution of a plan, and we assume that agents will obtain the resources they have been promised by the time the plan needs to be executed.

2.3 Dialogues and negotiation policies

For a given agent $x \in A$, where A is equipped with a language for negotiation $\mathcal{L}$, we define the sets $\mathcal{L}_x^{in}$, of all dialogue move schemata of which x is the receiver, but not the utterer; and $\mathcal{L}_x^{out}$, of all dialogue move schemata of which x is the utterer, but not the receiver. Note that we do not allow for agents to express utterances to themselves. Then, negotiation policies can be specified by sets of dialogue constraints:

Definition 4 (*dialogue constraint*) Given an agent system A, equipped with a language for negotiation $\mathcal{L}$, and an agent $x \in A$, a *dialogue constraint* for x is a (possibly non-ground) if-then rule of the form: $p(D,T) \wedge C \Rightarrow \exists\, T'(\hat{p}(D,T') \wedge T' > T)$, where

- $p(D,T) \in \mathcal{L}_x^{in}$ and $\hat{p}(D,T') \in \mathcal{L}_x^{out}$,
- the utterer of $p(D,T)$ is the receiver of $\hat{p}(D,T')$, and
- C is a conjunction of literals in the language of the knowledge base of x. [3]

[3]Note that C in general might depend on several time points, possibly but not necessarily including T; thus, we do not indicate explicitly any time variable for C.

$$(\mathcal{N}.IC1)\ \ tell(X, a, \mathbf{request}(\mathbf{give}(R)), D, T) \wedge\ have(R, T)\ \wedge\ \neg need(R, T)$$
$$\Rightarrow \exists\ T'(tell(a, X, \mathbf{accept}(\mathbf{request}(\mathbf{give}(R))), D, T1)\ \wedge T1 > T)$$

$$(\mathcal{N}.IC2)\ \ tell(X, a, \mathbf{request}(\mathbf{give}(R)), D, T) \wedge\ have(R, T)\ \wedge\ need(R, T)$$
$$\Rightarrow \exists\ T'(tell(a, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R))), D, T1)\ \wedge T1 > T)$$

$$(\mathcal{N}.IC3)\ \ tell(X, a, \mathbf{request}(\mathbf{give}(R)), D, T) \wedge\ \neg have(R, T)$$
$$\Rightarrow \exists\ T'(tell(a, X, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(R))), D, T1)\ \wedge T1 > T)$$

Figure 1. *policy regulations* $\mathcal{N}$

Any variables in a dialogue constraint not explicitly quantified are implicitly universally quantified over the constraint. The move $p(T)$ is referred to as the *trigger*, $\hat{p}(T')$ as the *next move* and C as the *condition* of the dialogue constraint.

Intuitively, the dialogue constraints of an agent x express x's negotiation policies. The intuitive meaning of a dialogue constraint $p(T) \wedge C \Rightarrow \exists\ T'(\hat{p}(T') \wedge T' > T)$ of agent x is as follows: if at a certain time T in a dialogue some other agent y utters a move $p(T)$, then the corresponding instance of the dialogue constraint is triggered and, if the condition C is entailed by the knowledge base of x, then x will utter $\hat{p}(T')$, with y as receiver, at a later time T' that is available for executing communicative actions (utterances). The instantiation of T' is left to the agent cycle (see Section 4.2).

A negotiation policy can be seen as a set of properties that must be satisfied at all times, by enforcing (uttering) the conclusion of the constraints that represent it, whenever they are triggered, and the conditions in their premise are entailed by $\mathcal{K}$. In this sense, constraints behave like active rules in databases and integrity constraints in abductive logic programming [2]. Indeed, as we will see in Section 4.2, the knowledge of an agent can be represented by an abductive logic program. In Figure 1 we show the negotiation policy regulations $\mathcal{N}$, inspired by [5]. a is the name of the agent which has $\mathcal{N}$ in its knowledge base.

Since we will prove some results for $\mathcal{N}$, we introduce the following notion:

Definition 5 ($\mathcal{N}$-*agent*) We call $\mathcal{N}$-agent an agent x equipped with the language $\mathcal{L}_1$ for negotiation, whose dialogue constraints in $\mathcal{K}_x$ are all and only those of $\mathcal{N}$, and such that $\mathcal{K}_x$ defines the predicates $have/2$ and $need/2$ in such a way that the following property holds: for every resource R and every time T, either $have(R, T)$ or $\neg have(R, T)$, and either $need(R, T)$ or $\neg need(R, T)$ are entailed from $\mathcal{K}_x$).

In section 4.2 we use abductive logic programming to represent $\mathcal{K}$. In that context, the predicates "have" and "need" can be defined by the following logic program.

$have(R,T) \leftarrow have(R,0) \ \wedge 0 < T \wedge \neg[gave\_away(R,T1), 0 < T1 \leq T]$
$have(R,T) \leftarrow obtained(R,T1) \ \wedge T1 < T \wedge \neg[gave\_away(R,T2), T1 < T2 \leq T]$
$obtained(R,T) \leftarrow tell(X,a,accept(request(give(R))),D,T)$
$gave\_away(R,T) \leftarrow tell(a,X,accept(request(give(R))),D,T)$
$need(R,T) \leftarrow have(R,T) \ \wedge plan(P,N,M) \ \wedge R \in N \cup M$[4]

A dialogue is defined as follows.

Definition 6 (*dialogue*) A *dialogue* d between two agents x and y is a set of ground dialogue moves, $\{p_0, p_1, p_2, \ldots\}$, such that, for a given set of time lapses $0 \leq t_0 < t_1 < t_2 < \ldots$:

1. $\forall\ i \geq 0$, p_i is uttered at time t_i;
2. $\forall\ i \geq 0$, if p_i is uttered by agent x (viz. y), then p_{i+1} (if any) is uttered by agent y (viz. x);
3. $\forall\ i > 0$, p_i can be uttered by agent $\alpha \in \{x,y\}$ only if there exists a (grounded) dialogue constraint $p_{i-1} \wedge C \Rightarrow p_i \wedge t_{i-1} < t_i \in \mathcal{K}_\alpha$ such that $\mathcal{K}_{\alpha,t_{i-1}} \wedge p_{i-1} \vdash C$;
4. $\forall\ i \geq 0$, the p_i share a unique dialogue identifier d.

A dialogue $\{p_0, p_1, \ldots p_m\}$, $m \geq 0$, is *terminated* if p_m is a ground final move, namely p_m is a ground instance of an utterance in $\mathcal{F}(\mathcal{L})$.
By condition 1, a dialogue is in fact a *sequence* of moves. By condition 2, agents alternate utterances in a dialogue. By condition 3, dialogues are generated by the dialogue constraints, together with the given knowledge base to determine whether the condition of triggered dialogue constraints is entailed. By condition 4, the dialogue moves of the same dialogue share the same dialogue identifier.
In this paper we propose a concrete agent framework that can produce dialogues, according to definition 6. The kind of dialogue that is relevant to our purposes is that started with a request of a resource R. Such dialogues will be initiated at time

[4] Obviously $need(R,T)$ implies $have(R,T)$, but we adopt both notations for the sake of readability. With respect to Section 2.2, we have the following correspondence, for a given agent x, time T and resource R:

- $\mathcal{K}_{x,T} \vdash \mathbf{m}(R) \iff \mathcal{K}_{x,T} \nvdash have(R,T) \ \wedge \mathcal{K}_{x,T} \vdash plan(P,N,M) \ \wedge R \in N \cup M$
- $\mathcal{K}_{x,T} \vdash \mathbf{n}(R) \iff \mathcal{K}_{x,T} \vdash need(R,T) \ \wedge \mathcal{K}_{x,T} \vdash have(R,T)$
- $\mathcal{K}_{x,T} \vdash \mathbf{a}(R) \iff \mathcal{K}_{x,T} \vdash have(R,T) \ \wedge \mathcal{K}_{x,t} \nvdash need(R,T)$
- $\mathcal{K}_{x,T} \vdash \mathbf{i}(R) \iff \mathcal{K}_{x,T} \nvdash have(R,T) \ \wedge \mathcal{K}_{x,t} \nvdash need(R,T)$

t by an agent x such that $\mathcal{K}_{x,t} \vdash \mathbf{m}(R)$.

Definition 7 (*request dialogue*) A *request dialogue* with respect to a resource R of an agent x is a dialogue d, consisting of a set $\{p_0, p_1, p_2, \ldots\}$ between x and some other agent $y \in A$ such that, for some $t \geq 0$,

- $p_0 = tell(x, y, request(give(R)), d, t)$, and
- $\mathcal{K}_{x,t} \vdash \mathbf{m}(R)$.

In the sequel, by *dialogue* we will mostly mean a *request dialogue*. In general, after a terminated request dialogue d with respect to a resource R and an intention $\mathcal{I}$ of some agent x, the agent might have been successful or not in obtaining the resource R. Let us assume that x and y are both $\mathcal{N}$-agents. Suppose d started at time t and terminated at time $t1$, and let $t' > t1$. If d is successful, the knowledge of agent x that initiated d changes in the following way:

$$\mathcal{K}_{x,t} \vdash \mathbf{m}(R) \overset{d}{\leadsto} \mathcal{K}_{x,t'} \vdash \mathbf{n}(R)$$

while the knowledge of the other agent, y, changes in the following way:

$$\mathcal{K}_{y,t} \vdash \mathbf{a}(R) \overset{d}{\leadsto} \mathcal{K}_{y,t'} \vdash \mathbf{i}(R)$$

If d has been unsuccessful, then both agents' knowledge about resources does not change, and x might decide to engage in a new dialogue to obtain R from a different agent.

While dialogues can be characterized by the initial and final states of the knowledge of the participants, negotiation policies can also be characterized by the possible dialogues (with consequent changes in states) that they generate:

Definition 8 (*self-interested agent*) An agent x is called *self-interested* if, for all request dialogues d between agents starting x and y at time t and terminating at time $t1 < t'$, with respect to a resource R:

- for all $\hat{R} \in \mathcal{R}_t(\mathcal{I}_x)$, $\hat{R} \neq R$, then $\mathcal{K}_{x,t} \vdash \mathbf{s}(\hat{R})$ if and only if $\mathcal{K}_{x,t'} \vdash \mathbf{s}(\hat{R})$, where $\mathbf{s}$ can be a, i, n, or m;
- if $\mathcal{K}_{x,t} \vdash \mathbf{m}(R)$ then either $\mathcal{K}_{x,t'} \vdash \mathbf{m}(R)$ or $\mathcal{K}_{x,t'} \vdash \mathbf{n}(R)$
- if $\mathcal{K}_{x,t} \vdash \mathbf{a}(R)$ then either $\mathcal{K}_{x,t'} \vdash \mathbf{a}(R)$ or $\mathcal{K}_{x,t'} \vdash \mathbf{i}(R)$
- if $\mathcal{K}_{x,t} \vdash \mathbf{n}(R)$ then $\mathcal{K}_{x,t'} \vdash \mathbf{n}(R)$;
- if $\mathcal{K}_{x,t} \vdash \mathbf{i}(R)$ then $\mathcal{K}_{x,t'} \vdash \mathbf{i}(R)$;

the first bullet expresses the fact that agents negotiate only one resource at a time; the other bullets that it is self interested (it retains the resources that it needs) and does not change its intention, as we assumed in the beginning.

$\mathcal{N}$-agents are an example of self-interested agents.

2.4 The agent dialogue cycle

In order to obtain all the resources missing in a plan, a single dialogue is not enough, in general. A *sequence of dialogues* is needed, for an agent to attempt to obtain all the resources that she requires.In order to produce sequences of dialogues, agents run a *dialogue cycle*, having the following properties:

- no agent is asked twice for the same resource within the dialogue sequence;
- if a resource is not obtained from one agent, then it is asked from some other agent, if any;
- if a resource is not obtained after asking all agents, then the agent dialogue cycle terminates with **failure**.

The idea behind the third property is that the agent will not carry on asking for the other resources, since, at least one resource in the current intention cannot be obtained, the plan in the intention will not possibly be executable. One cycle with these properties is defined in [5]. Below we give a modified version of it.

Definition 9 (*agent dialogue cycle*) Given an initial intention I of agent x, containing a set of missing resources Rs, the *agent dialogue cycle* is the following:

1. set $SD = \emptyset$;
2. until $Rs = \emptyset$ repeat
   (a) select a resource $R \in Rs$ and update Rs to $Rs - \{R\}$;
   (b) until a successful dialogue d is found, between x and some other agent, repeat
      i. select an agent $y \in A \setminus \{x\}$ such that there does not exist in SD a request dialogue between x and y with respect to R;
      ii. if such y does not exist, then **failure** and restore Rs to $Rs \cup \{R\}$;
      iii. construct a request dialogue d between x and y with respect to R;

(c) add d to SD;

3. **success**.

In Section 4 we will show an operational model that gives a concrete implementation of such agent dialogue cycle.

2.5 Properties of $\mathcal{K}$: determinism and exhaustiveness

An agent should ideally produce one and only one move in response to any non-final move of other agents. The following definitions pave the way towards building agents with such behaviour.

Definition 10 (*determinism of* $\mathcal{K}$) An agent knowledge base $\mathcal{K}_x$ is *deterministic* iff for each incoming dialogue move $p(t)$ which is a ground instance of a schema in $\mathcal{L}_x^{in}$, there exists at most one outgoing dialogue move $\hat{p}(t')$ which is a ground instance of a schema in $\mathcal{L}_x^{out}$ such that $p(t) \wedge C \Rightarrow \hat{p}(t') \wedge t' > t \in \mathcal{K}_x$ and $\mathcal{K}_x \wedge p(t)$ entails C.

In order for a dialogue not to end before a final move is reached, it is important that an agent be "exhaustive":

Definition 11 (*exhaustiveness of* $\mathcal{K}$) An agent knowledge base $\mathcal{K}_x$ is *exhaustive* iff for each dialogue move $p(t)$ which is a ground instance of a schema in $\mathcal{L}_x^{in} \setminus \mathcal{F}(\mathcal{L})$, there exists at least one dialogue move $\hat{p}(t+1)$ which is a ground instance of a schema in $\mathcal{L}_x^{out}$ such that $p(t) \wedge C \Rightarrow \hat{p}(t+1) \in \mathcal{K}_x$ and $\mathcal{K}_x \wedge p(t)$ entails C.

We can present here a first result about $\mathcal{N}$-agents. Its proof is sketched in Appendix.

Theorem 1 If x is an $\mathcal{N}$-agent, then $\mathcal{K}_x$ is exhaustive and deterministic.

3 A simple resource reallocation problem

A resource reallocation problem can be defined as follows.

Definition 12 (*resource reallocation problem – r.r.p.*) Given an agent system A,

- the *r.r.p. for an agent* $x \in A$ is the problem of finding a knowledge base $\mathcal{K}_{x,t'}$, $t' > t$ such that $\nexists\, R | \mathcal{K}_{x,t'} \vdash \mathbf{m}(R)$. If such $\mathcal{K}_{x,t'}$ is found, we say that the r.r.p. for x is solved at time t'.

- the *r.r.p. for the agent system* A is the problem of solving the r.r.p. for every agent in A at the same time. The *r.r.p.* for A is *solved* if the required sets of knowledge bases is found.

A resource reallocation problem is *solved* if there is no agent in the system with a missing resource. It is called *solvable* if there are enough resources within the system (i.e., owned by the agents in the system) for all agents to carry out their intentions – after, possibly, a number of exchanges of ownership of resources – *unsolvable* otherwise.

Definition 13 (*solvable resource reallocation problem*) For a system A, let $\mathcal{I}_x$ be the intention of agent $x \in A$, $\mathcal{R}_{x,0}$ the set of initially owned resources, and $\mathcal{M}_{x,0}$ be the set of initially missing resources in $\mathcal{I}_x$ ($\mathcal{M}_{x,0} = \mathcal{R}_0(\mathcal{I}_x) \setminus \mathcal{R}_{x,0}$). Let $\mathcal{M} = \bigcup_{x \in A} \mathcal{M}_x$, and $\mathcal{R}_0 = \bigcup_{x \in A} \mathcal{R}_{x,0}$. The resource reallocation problem for A is *solvable* iff $\mathcal{M} \subseteq \mathcal{R}_0$, it is *unsolvable* otherwise.

We will see that the resource reallocation problem can be solved by means of sequences of request dialogues between pairs of agents, $x, y \in A$, such that at some t, and for some R, $\mathcal{K}_{x,t} \vdash \mathbf{m}(R)$. The aim of the dialogue is for agent x to (consistently) modify its knowledge base and obtain, at $t' > t$, when the dialogue is terminated, $\mathcal{K}_{x,t'} \vdash \mathbf{n}(R)$.

4 An operational model for the agent reasoning and dialogue cycle

We give an operational model of the negotiation framework, consisting of an abductive logic program for $\mathcal{K}$, an abductive proof-procedure for agent reasoning, and a definition of an agent cycle. We also re-define the concept of agent with respect to this new framework.

The proof-procedure that we propose is the one proposed in [4], henceforth called the ST proof-procedure. This is a modification of the IFF proof-procedure [1].

The ST proof procedure modified the Fung-Kowalski one in its handling of negation. The modification resulted in achieving a "negation as failure" behaviour within the abductive proof-procedure more appropriate for the production rule/active rule behaviour needed from the integrity constraints. [4] contains a detailed description of the ST procedure. Here we summarize it.

We presume notions of clause, logic program, completion of a predicate, iff-definition and selective completion.

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*.[5] All variables in an integrity constraint are implicitly universally quantified, with scope the entire integrity constraint.

An **abductive logic program** [2] 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.

4.1 The ST proof procedure

The ST proof procedure 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 $T = comp_{\overline{\mathcal{A}}}(P)$ (in T, abducible predicates are not completed), for some given abductive logic program $\langle P, \mathcal{A}, I \rangle$. Initially all implications in I are marked (by means of the $*$ symbol). The basic inference rules are: [6]

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**) $B \wedge \mathtt{false}$ is replaced by $\mathtt{false}$, $B \vee \mathtt{false}$ is replaced by B, and so on.

4. **Equality rewriting**: applies equality rewrite rules (see [3]) simulating the unification algorithm and the application of substitutions.

5. **Negation rewriting**: a *marked* implication $*[not\, A_1 \wedge \ldots \wedge not\, A_m \rightarrow A]$, $m \geq 1$, is replaced by the disjunction $[provable(A_1)] \vee \ldots \vee [provable(A_m)] \vee [*[A_1 \rightarrow false] \wedge \ldots \wedge *[A_m \rightarrow false] \wedge A]$.
   $*[true \rightarrow A]$ is automatically rewritten as A.

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

[5] Note that the constraints in $\mathcal{N}$ do not conform to this syntax. But they can be written to do so. A constraint $Q(x) \Rightarrow \exists\, yR(x,y)$ can be rewritten as $Q(x) \Rightarrow T(x)$ together with a definition $T(y) \leftarrow R(x,y)$. In the remainder of the paper we assume that $\mathcal{N}$ is in the required form.

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

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, logical simplification, and negation rewriting. Every negative literal `not` A as a conjunct in the initial goal as well as in any derived goal is rewritten as an implication $A \Rightarrow \texttt{false}$.

Every derivation relies upon some control strategy. Some strategies are preferable to others. E.g., if splitting is applied whenever possible, then derived goals are disjunctions. We assume that the rules are always applied in the left most disjunct, thus simulating a depth-first search strategy.

[4] gives a detailed characterization of the notions of extracted and computed answers. For simplicity and because of lack space here we simplify these notions to that of the "computed abducibles". In the application to agent negotiation, these abducibles will be communication actions potentially executed by the agent.

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 N does not contain $provable(A)$ for any A. The **abducibles computed from** N are 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.

Henceforth, in the remainder of the paper, the notion of entailment refers to the application of the ST proof-procedure. In particular when we write $\mathcal{K}_{x,t} \vdash Goal$ we will consider $\mathcal{K}$ to be an abductive logic program, and $Goal$ a goal in ALP that is entailed by $\mathcal{K}$ according to the ST procedure.

4.2 Abductive agents

The agent architecture that we propose for this framework is inspired by that of Kowalski-Sadri agents [3]. Every KS-agent can be thought of as an abductive logic program, equipped with a (possibly empty) initial goal. The abducibles are *actions* to be executed as well as *observations* to be performed. In our case, actions are *communicative actions*, namely dialogue moves uttered by the agent, and observations are dialogue moves uttered by other agents. Updates, observations, requests and queries are treated uniformly as goals. The abductive logic program can be a temporal theory.

To cycle at time t,

(i) observe any input at time t,

(ii) record such input, if any,

(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 $t' > t + r$, if any,

(vi) execute the selected action at time t' and record the result,

(vii) cycle at time $t' + 1$.

Figure 2. *KS agent cycle*

The basic "engine" of a KS-agent is the IFF proof procedure of [1], executed via the cycle shown in Figure 2.

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. 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.

IN our extension of he cycle we make the simplifying assumption that time is only consumed by the application of the proof procedure, and not by the decisions about whether or not to process inputs and whether or not to output an action.

In our particular case of abductive agents for negotiation dialogues, the cycle must be extended in order to enforce *atomicity*, to tackle concurrent dialogues, and *interleaving*, used to produce dialogues that conform to the definition given in Section 2.3. By *atomicity* we mean that each agent is involved in at most one dialogue at any time; therefore

- a move b observed during the step i of the cycle is recorded (in step ii) only if the agent is not already involved in another dialogue, or if there is an ongoing dialogue d and b is a move of d, and
- the agent can utter a move a only if there is no ongoing dialogue (it is the first utterance of a new dialogue), or there is an ongoing dialogue d, and a is part

To cycle at time t,

- (i) **[observe] observe any input** at time t, and put it in *input*;
  - $(i.1)$ **[filter observations]** if *input* $= \emptyset$, then go to (iv);
  - $(i.2)$ **[filter observations]** if *ongoing(D)* and either $\nexists\, b \in$ `buffer` such that $b = tell(Y, x, Subject, D, T)$ or $waiting = false$, then go to step (iv);
- (ii) **[record input]** if *ongoing(D)* then **record an input** $b \in$ *input* such that $b = tell(Y, x, Subject, D, T)$
- (ii') **[record input]** if *ongoing(D)* is false for all D (reply to a new dialogue D'): **record an input** $b \in$ *input* (say, $b = tell(Y, x, Subject, D', T)$), and set *ongoing(D)* to *ongoing(D')*;
  - $(ii.1)$ **[update dialogue status]** if b is a final move then set *ongoing(D)* to false, otherwise set *waiting* to false;
- (iii) **[think] resume** the KS procedure by propagating the inputs, if any;
- (iv) **[think] continue applying the KS procedure**, for a total of $\boldsymbol{r}$ units of time;
  - $(iv.1)$ **[filter actions]** if there does not exist an atomic action which can be executed at time $t' = t + r$, or if $waiting = true$, then go to step (vii);
  - $(iv.2)$ **[filter actions]** if *ongoing(D)* and there does not exist an atomic action $a = tell(x, Y, Subject, D, T)$ which can be executed[7] at time $t' = t + r$, then go to step (vii);
- (v) **[select action]** if *ongoing(D)* then **select an atomic action** $a = tell(x, Y, Subject, D, T)$ such that a is an abducible computed by the proof-procedure which can be executed at time $t' = t + r$ and instantiate T to t';
- (v') **[select action]** if *ongoing(D)* is false for all D (start a new dialogue D'): **select an atomic action** $a = tell(x, Y, Subject, D', T)$ such that a is an abducible computed by the proof-procedure which can be executed at time $t' = t + r$, instantiate T to t' and set *ongoing(D)* to *ongoing(D')*;
  - $(v.1)$ **[update dialogue status]** if a is a final move then set *ongoing(D)* to false, otherwise set *waiting* to true;
- (vi) **[act] execute the selected action** at time t' and record the result;
- (vii) **[cycle] cycle** at time $t' + 1$.

Figure 3. *Extended agent cycle*

of it.

By *interleaving* we mean that dialogue moves must be interleaved within the same dialogue (condition 2 of Def. 6): therefore, an agent will not accept two dialogue moves of a same dialogue from another agent, unless he utters something in between.

In order to enforce these properties, we introduce the following data structures:

- an input buffer, *input*, that contains all the observable predicates (incoming dialogue moves) that have not yet been recorded by the agent; recording an input involves adding it to the current (initial or derived) goal handled by the proof-procedure;
- a predicate *ongoing/1*, that contains the identifier of the ongoing dialogue, if any; initially, *ongoing(D)* is false for all D;
- a predicate *waiting/0*, initially false, that is true if the agent is expecting a dialogue move from another agent within an ongoing dialogue.

The revised agent cycle is shown for an agent x in Figure 3. The proof-procedure referred to within the cycle is the ST proof-procedure. The original KS agent cycle is extended with filtering/updating steps according to the state of the dialogue, for what concerns both the recording of an incoming dialogue move and the possible uttering of an outgoing move. It is worth noticing that such modifications do not interfere with the reasoning part of the agent, and in particular to the application of the proof-procedure.
Figure 4 shows the abductive logic program implementing he dialogue cycle.

The two integrity constraints are used to start the negotiation process (IC.1) from the missing resources of the agent's plan, and to determine failure (IC.2) once the agent has failed to get a specific resource after asking all the agents in the system (but itself). The implementation of the dialogue cycle concludes the operational definition of the framework for agent negotiation, whose components, the abductive agents, can be defined as follows:

Definition 14 (*abductive agent*) An *abductive agent* is an agent whose *knowledge base* $\mathcal{K}$ is expressed in the form of an abductive logic program, equipped with the *ST abductive proof-procedure* to reason upon $\mathcal{K}$; and with the *dialogue cycle* and the *extended agent cycle* both defined in Section 4.2. An *abductive* $\mathcal{N}$*-agent* is an $\mathcal{N}$-agent that is also an abductive agent.

Definition 15 (*exhaustive and deterministic agent*) An abductive agent x is exhaustive and deterministic if, at each time t, x produces one and only one ground

$$
\begin{array}{ll}
(\textbf{def.1}) & ask\_for(M) \leftarrow ask(R) \ \wedge R \in M \ \wedge get(R) \ \wedge R' is M \setminus \{R\} \ \wedge ask\_for(R') \\
& ask\_for(M) \leftarrow ask(R) \ \wedge R \in M \ \wedge fail\_to\_get(R) \\
(\textbf{def.2}) & get(R) \leftarrow tell(X, a, accept(request(give(R))), D, T) \\
(\textbf{def.3}) & ask(R) \leftarrow agents(A) \ \wedge ask\_agents(A, R) \\
(\textbf{def.4}) & ask\_agents(A, R) \leftarrow A = \emptyset \\
& ask\_agents(A, R) \leftarrow X \in A \ \wedge tell(a, X, request(give(R)), D, T) \ \wedge get(R) \\
& ask\_agents(A, R) \leftarrow X \in A \ \wedge tell(a, X, request(give(R)), D, T) \\
& \qquad \wedge tell(X, a, refuse(request(give(R))), D, T') \\
& \qquad \wedge A' = A \setminus \{X\} \ \wedge ask\_agents(A', R) \\
(\textbf{def.5}) & fail\_to\_get(R) \leftarrow agents(A) \ \wedge refused\_by\_all(R, A) \\
(\textbf{def.6}) & refused\_by\_all(R, A) \leftarrow A = \emptyset \\
& refused\_by\_all(R, A) \leftarrow X \in A \\
& \qquad \wedge tell(X, a, refuse(request(give(R))), D, T) \\
& \qquad \wedge A' = A \setminus \{X\} \ \wedge refused\_by\_all(R, A') \\
(\textbf{IC.1}) & plan(\_, \_, M) \Rightarrow ask\_for(M) \\
(\textbf{IC.2}) & fail\_to\_get(R) \Rightarrow \bot
\end{array}
$$

Figure 4. *Implementation of the dialogue cycle*

move, in response to any non-final move of other agents.

5 Formal results

We prove that the framework described in Section 4 is capable of generating dialogues, provided that the agents in the system are exhaustive and deterministic with respect to the negotiation language. In fact, if an agent is exhaustive and deterministic, then at each step of the dialogue the agent will produce *exactly one* reply to a (non final) move made by the other agent. If both agents involved in a dialogue are exhaustive and deterministic, then at each step exactly one agent is guaranteed to produce only one dialogue move, unless a final move is made, and we can produce a dialogue that conforms to the definition given in Section 2.3. All proofs are sketched in Appendix.

Lemma 1 (*entailment of dialogue move*) If x is an abductive agent, and $\mathcal{K}_x$ is *exhaustive* and *deterministic* with respect to a given language for negotiation $\mathcal{L}$, then x is an exhaustive and deterministic agent.

Corollary 1 (*exhaustive and deterministic abductive $\mathcal{N}$-agents*) If x is an *abductive $\mathcal{N}$-agent*, then x is exhaustive and deterministic.

Theorem 2 (*production of dialogues*) Let x and y be two agents sharing a language $\mathcal{L}$ for negotiation. Suppose x and y are both deterministic and exhaustive. Let $S = \{p_1, p_2, \ldots, p_n\}$ be a sequence of utterances between x and y (i.e., for each p_i the utterer is either x or y and the receiver is either x or y) such that the p_i all share the same identifier d. Then S is a dialogue according to Definition 6.

By Theorem 2, the self-interested abductive $\mathcal{N}$-agents are able to produce dialogues. We will call a system composed of abductive $\mathcal{N}$-agents an $\mathcal{N}$-system.

If A is an $\mathcal{N}$-system and there exists a solution to the r.r.p. associated with A, then there exists a set of sequence of dialogues that solves the problem. If all the dialogue sequences are allowed to terminate, a set of sequences of dialogues that solves A's r.r.p. will be actually generated by the agents in A. Formally:

Theorem 3 (*weak completeness of an $\mathcal{N}$-system*) Let A be an $\mathcal{N}$-system. If the r.r.p. for A is solvable, then there exists a set of sequences of dialogues amongst the agents in A ending at time t such that for each $x \in A$, $\mathcal{K}_{x,t} \not\vdash \mathbf{m}R$.

6 Conclusions

We have presented a framework for agent dialogue and negotiation based on abductive reasoning. Based on earlier work [5], we have given a formal definition of dialogue, and presented a domain where it can effectively be used, that of negotiation for resource reallocation. In order to provide an operational model for the framework, we introduced a modified version of the agent cycle of [3], with respect to the proof-procedure introduced in [4], rather than the IFF that [3] chooses to adopt. Our model is able to produce sequences of negotiation dialogues that conform to the given definition, and a specific instance of a negotiation framework that we define, i.e. an $\mathcal{N}$-system, is able to find solutions to the resource reallocation problem, if any.

Acknowledgements

We would like to thank Bob Kowalski for his interest in this work and for helpful discussions and suggestions about the dialogue cycle. This work has been partially supported by the EU funded project IST-2001-32530, Societies Of ComputeeS (SOCS), and by the Marco Polo research grant (University of Bologna).

References

[1] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 1997.

[2] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. *Handbook of Logic in AI and Logic Programming*, 5:235–324, 1998.

[3] R. A. Kowalski and F. Sadri. From logic programming to multi-agent systems. *Annals of Mathematics and AI*, 1999.

[4] F. Sadri and F. Toni. Abduction with negation as failure for active and reactive rules. In E. Lamma and P. Mello, editors, *Proceedings AI*IA 99, 6th Congress of the Italian Association for Artificial Intelligence*, number 1792 in LNAI, pages 49–60. Springer Verlag, 2000.

[5] F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: agent varieties and dialogue sequences. In *Proceedings ATAL'01, Seattle, WA*, volume to appear of *LNAI, Intelligent Agents VIII (best paper award)*. Springer Verlag, 2001.

[6] F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogues and negotiation: an abductive approach. In *Proceedings AISB'01 Convention, York, UK*, March 2001.

Appendix: proof sketches

Proof of Theorem 1 (sketch). This result follows from the property that at each time t and for each resource R either $have(R,t)$ holds or $\neg have(R,t)$ holds, and either $needs(R,t)$ holds or $\neg needs(R,t)$ holds. In fact, let x be an $\mathcal{N}$-agent. The conditions of $\mathcal{N}.IC1$, $\mathcal{N}.IC2$, and $\mathcal{N}.IC3$ are respectively $C_1(R,T) = have(R,T) \wedge \neg need(R,T)$, $C_2(R,T) = have(R,T) \wedge need(R,T)$, and $C_3(R,T) = \neg have(R,T)$. It is easy to see that all the conditions are mutually exclusive, and therefore $\mathcal{K}_x$ is deterministic. On the other hand, $\neg C_1 \wedge \neg C_2 = \neg[have(R,T) \ \wedge \ \neg need(R,T)] \wedge \neg[have(R,T) \wedge need(R,T)] = [\neg have(R,T) \vee need(R,T)] \wedge [\neg have(R,T) \vee \neg need(R,T)] = \neg have(R,T) = C_3$, and similarly for the other two pairs of constraints. Since the only incoming non-final move in $\mathcal{L}_1$ is $tell(X,x,request(give(R)),D,T)$, that is the trigger of all constraints in $\mathcal{N}$, $\mathcal{K}_x$ is exhaustive.

Proof of Lemma 1 (sketch). Each constraint of $\mathcal{K}_x$ is a dialogue constraint with a trigger $p_t \in \mathcal{L}_x^{in}$. Such p_t becomes true in $\mathcal{K}_x$ if and only if x receives an utterance, p_t, at a time t, that is in $\mathcal{L}_x^{in}$. Since $\mathcal{K}_x$ is exhaustive, for all possible utterances in $\mathcal{L}_x^{in}$ uttered at time T there exists an IC that is fired, giving an utterance $p_{t'}$ intended to be uttered by agent x at time t'. Since $\mathcal{K}_x$ is deterministic, no other $\bar{p}_{t'}$, different from $p_{t'}$, intended to be uttered at time t', is entailed.

Proof of Theorem 2 (sketch). By Lemma 1, for a given utterance (trigger) $p_t \in \mathcal{L}_x^{in} \cap \mathcal{L}_y^{out}$, which is timestamped t, there exists one and only one $p_{t+1} \in \mathcal{L}_x^{out} \cap \mathcal{L}_y^{in}$, which is timestamped $t' = t + r$, for a given r. x's agent cycle will therefore utter $p_{t'}$ at time t'. Therefore the agents will produce a dialogue as defined in 2.3.

Proof of Theorem 3 (sketch). Let us consider an agent system A, associated with a solvable resource reallocation problem, and for A a set of terminated sequence of dialogues, $\Sigma = \{d_{1(t_1)}, \ldots, d_{m(t_m)}\}$. Let us assume that Σ starts at time $t = t1$ and terminates at time $t' = t_m$.

If all sequences of dialogues in Σ have successfully terminated, the resource reallocation problem for A has been solved. Otherwise, there exists (at least) one agent x, whose r.r.p. has not been solved, i.e., $\exists x \in A | \mathcal{K}_{x,t'} \vdash \mathbf{m}(R)$, and the sequence of dialogues associated with x is terminated. This could be due to either of the following reasons: (i) $\nexists y \in A | \mathcal{K}_{y,t'} \vdash \mathbf{a}(R)$, or ($ii$) $\forall y \in A | \mathcal{K}_{y,t'} \vdash \mathbf{a}(R)$, $\mathcal{K}_{y,t} \nvdash \mathbf{a}(R)$. If ($i$), then the problem is unsolvable without changing some agent's intentions, which is in contradiction with the premises. If (ii), then it means that such agents y had a dialogue, starting at t_1 and terminating at t_2, such that $\mathcal{K}_{y,t_1} \nvdash \mathbf{a}(R)$ and $\mathcal{K}_{x,t_2} \vdash \mathbf{a}(R)$, which means that x is not an $\mathcal{N}$-agent, which is again in contrast with the premises.