An abductive logic programming architecture for negotiating agents

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

[1] Department of Computing, Imperial College
180 Queens Gate, SW7 London, UK
{fs, ft}@doc.ic.ac.uk
[2] DEIS, Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
ptorroni@deis.unibo.it

Abstract. In this paper, we present a framework for agent negotiation based on abductive logic programming. The framework is based on an existing architecture for logic-based agents, and extends it by accommodating dialogues for negotiation. As an application of negotiating agents, we propose a resource-exchanging problem. The innovative contribution of this work is in the definition of an operational model, including an agent cycle and dialogue cycle, and in the results that apply in the general case of abductive agents and in the specific case of a class of agent systems.

1 Introduction

In multi-agents systems, agents may need to interact in order to exchange resources, when the resources they own are not enough to achieve their goals prior to exchange. If each agent has global visibility of the other agents' resources and intentions, a "global" plan can be generated by a single individual. However, a global plan cannot be possibly generated if there is no such visibility or if the design of the system is based on agent autonomy. Also, the autonomous agent behaviour and the consequent dynamic evolution of the system could make any such plan obsolete before completion. In there comes negotiation.

The focus of this work is the knowledge and the reasoning that is required to build negotiation dialogues between agents, in order to exchange resources that help achieve the agents' objectives. We express such knowledge in a declarative way, as logic programs and integrity constraints, as in abductive logic programming [1], with the dialogue moves treated as abducibles/hypotheses, and dialogue/negotiation policies treated as integrity constraints.

We propose an operational model, based on the abductive reasoning mechanism and on an agent cycle, and show that it can be used to generate (negotiation) dialogues. The framework extends an agent architecture initially proposed in [2], to enable dialogue generation. The agent architecture in [2] relies upon executing the IFF abductive proof-procedure defined in [3] as the reasoning engine of the agent. We identify the inadequacy of the IFF procedure for our domain of application, and adopt the procedure in [4] instead. This paper represents in many respects a progress from previous related work [5, 6]. It provides a better formalization of the framework for negotiation, by defining an agent cycle that accommodates dialogues as a result of the agent reasoning. This improvement is necessary to prove some strong results about the general negotiation framework. The paper also defines two classes of agents (N-agents and N^+-agents), that implement the behaviour of self-interested resource-bounded agents and that are provably able to solve a resource reallocation problem.

The paper is organized as follows: in Section 2 we review and revise the dialogue framework of [6, 5], in Section 3 we augment this framework with new concepts to characterise the state of negotiating agents and specify a concrete variety of negotiating agents, referred to as self-interested agents, in Section 4 we define the abductive framework, abductive agents and concrete instances of abductive agents referred to as N-agents and N^+-agents, and in Section 5 we present the formal results. Section 6 concludes.

2 Background: Knowledge and Dialogues for Negotiation

Agents negotiate using a shared language, consisting of possible utterances, or *dialogue moves*, defined as follows:[1]

Definition 1. A *dialogue move* (between X and Y) 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, $X \neq Y$, D is the *identifier* of the dialogue to which the move belongs, and T is the *time* when the move is uttered. **Subject** is the *content* of the move, expressed in some given *content language*.

Note that the identifier of a dialogue uniquely determines the dialogue that includes the dialogue move. Note also that this component of a dialogue move is new and was not present in the definition in [5].

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.

Definition 2. 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 simple example of language for negotiation is ($\mathcal{N}$ stands for *negotiation*):

$$\begin{aligned}\mathcal{L}_{\mathcal{N}} = \{ & tell(X, Y, \mathbf{request}(\mathbf{give}(Resource)), D, T),\\ & tell(X, Y, \mathbf{accept}(\mathbf{request}(\mathbf{give}(Resource))), D, T)\\ & tell(X, Y, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(Resource))), D, T) \;\}\end{aligned}$$

The initial and final moves are:

$$\mathcal{I}(\mathcal{L}_{\mathcal{N}}) = \{tell(X, Y, \mathbf{request}(\mathbf{give}(Resource)), D, T)\}$$

$$\begin{aligned}\mathcal{F}(\mathcal{L}_{\mathcal{N}}) = \{ & [\textit{succ. moves:}]\ tell(X, Y, \mathbf{accept}(\mathbf{request}(\mathbf{give}(Resource))), D, T)\\ & [\textit{unsucc. moves:}]\ tell(X, Y, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(Resource))), D, T)\}\end{aligned}$$

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

In this example all moves are either initial or final, although this is not always the case (see [6]). We sometimes represent a dialogue move in abbreviated form as $p(D,T)$ or simply as p, if the discussion is independent of the missing parameters.

Definition 3. An *agent system* consists of: a language for negotiation $\mathcal{L}$; and a finite set A, with at least two elements, where each $X \in A$ is a ground term, representing the name of an agent, and each agent is equipped at each time T with a *knowledge base* $\mathcal{K}_{X,T}$.

In the sequel, we will sometimes drop X and/or T, if clear from the context.

We assume that the knowledge base of each agent is represented in some logic-based language equipped with a notion of entailment $\vdash$. For simplicity, we assume that agents share the logic-based language and $\vdash$ (but not the knowledge bases necessarily). In Section 4, we will adopt abductive logic programming [1] as the logic-based language used to model the knowledge base of agents.

The knowledge base $\mathcal{K}_{X,T}$ of agent X at time T consists of *(1) domain-dependent beliefs* about the world, such as the information about the agent system, e.g., $agents(\{a,b,c\})$; *(2) domain-independent beliefs*, used to regulate the negotiation dialogues, e.g. see $IC.1 - IC.3$ later in this section, and to represent changes in the agent's own state, such as its ownership of resources, e.g. see $D.1 - D.4$ later in this section; *(3)* information about the *resources* the agent owns before the dialogues start, e.g., $have(picture, 0)$, with 0 the initial time; *(4)* the agent's *goal*, e.g., $goal(hung(picture))$; *(5)* the agent's *intention*, represented as $plan(P, Req)$, i.e., the *plan* P that the agent intends to carry out in order to achieve its goal, associated with the set of resources Req that are required to carry it out, e.g., $plan(\langle hit(nail), hang(picture)\rangle, \{picture, nail, hammer\})$; and finally, *(6)* the past (time-stamped) *utterances*, e.g. $tell(a, b, \mathbf{request}(\mathbf{give}(nail)), d, 1)$, $tell(b, a, \mathbf{accept}(\mathbf{request}(\mathbf{give}(nail))), d, 3)$.

It is worth noting that the goal and the intention are not time-stamped. In fact, we assume that plans in intentions are given initially and do not change. The only part of $\mathcal{K}_{X,T}$ that changes over time is the the set of past utterances *(6)*, as this part grows monotonically during the course of dialogues.

The following definitions of *have* and *need* for agent $a \in A$ (similarly for the other agents in A) may be part of the domain-independent beliefs *(2)* in $\mathcal{K}_a$:

$$
\begin{array}{ll}
(D.1) & have(R,T) \leftarrow have(R,0) \ \wedge 0 < T \wedge \neg[gave\_away(R,T1), 0 < T1 \leq T] \\
 & have(R,T) \leftarrow obtained(R,T1) \ \wedge T1 < T \\
 & \qquad\qquad\qquad\qquad \wedge\ \neg[gave\_away(R,T2), T1 < T2 \leq T] \\
(D.2) & obtained(R,T) \leftarrow tell(X,a,accept(request(give(R))),D,T) \\
(D.3) & gave\_away(R,T) \leftarrow tell(a,X,accept(request(give(R))),D,T) \\
(D.4) & need(R,T) \leftarrow have(R,T) \ \wedge plan(P,Req) \ \wedge R \in Req
\end{array}
$$

Note that we assume 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.

For a given agent $X \in A$, where A is an agent system 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, and $\mathcal{L}_X^{out}$, of all dialogue move schemata of which X is the utterer. Then, negotiation policies can be specified by sets of dialogue constraints, defined as follows:

Definition 4. 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 $p(D,T) \wedge C \Rightarrow [\exists\, T'(\hat{p}(D,T') \wedge T < T')]$, 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 the receiver of $p(D,T)$ is the utterer of $\hat{p}(D,T')$,
- C is a conjunction of literals in the language of the knowledge base of X, [2]
- any variables not explicitly quantified are implicitly universally quantified over the constraint.

The move $p(D,T)$ is referred to as the *trigger*, $\hat{p}(D,T')$ 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 $p(D,T) \wedge C \Rightarrow [\exists\, T'(\hat{p}(D,T') \wedge T < T')]$ of agent X is as follows: if at time T in a dialogue D some other agent Y utters $p(D,T) = tell(Y,X,Subject,D,T)$, then the corresponding instance of the dialogue constraint *is triggered* and, if the condition C is entailed by $\mathcal{K}_{X,T'}$, then the constraint *fires* and X will utter $\hat{p}(D,T') = tell(X,Y,Subject',D,T')$, at a later time T' that is available for utterances. The instantiation of T' is left to the agent cycle, and can be performed as explained in section 4.

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 them, whenever they fire. In this sense, constraints behave like active rules in databases and integrity constraints in abductive logic programming. In Section 4, we will adopt the negotiation policy $\mathcal{N}$ consisting of the following three dialogue constraints (a is the name of the agent which has $\mathcal{N}$ in its knowledge base), which are part of the domain-independent beliefs *(2)* in $\mathcal{K}_a$:

$$
\begin{array}{ll}
(IC.1) & tell(X,a,\mathbf{request}(\mathbf{give}(R)),D,T) \wedge\ have(R,T)\ \wedge\ \neg need(R,T) \\
 & \quad \Rightarrow \exists\, T'\,(tell(a,X,\mathbf{accept}(\mathbf{request}(\mathbf{give}(R))),D,T')\ \wedge T < T') \\
(IC.2) & tell(X,a,\mathbf{request}(\mathbf{give}(R)),D,T) \wedge\ have(R,T)\ \wedge\ need(R,T) \\
 & \quad \Rightarrow \exists\, T'\,(tell(a,X,\mathbf{refuse}(\mathbf{request}(\mathbf{give}(R))),D,T')\ \wedge T < T') \\
(IC.3) & tell(X,a,\mathbf{request}(\mathbf{give}(R)),D,T) \wedge\ \neg have(R,T) \\
 & \quad \Rightarrow \exists\, T'\,(tell(a,X,\mathbf{refuse}(\mathbf{request}(\mathbf{give}(R))),D,T')\ \wedge T < T')
\end{array}
$$

Definition 5. Given an agent system A equipped with $\mathcal{L}$, a *dialogue* between two agents X and Y in A is a set of ground dialogue moves in $\mathcal{L}$, $\{p_0, p_1, p_2, \ldots\}$, 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] 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.

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 $U \in \{X, Y\}$ only if there exists a (grounded) dialogue constraint $p_{i-1} \wedge C \Rightarrow [p_i \wedge t_{i-1} < t_i] \in \mathcal{K}_U$ such that $\mathcal{K}_{U,t_i} \wedge p_{i-1} \vdash C$;
4. there is an identifier D such that, $\forall\ i \geq 0$, the dialogue identifier of p_i is D;
5. $\forall t$, $t_{i-1} < t < t_i$, $\forall i > 0$ s.t. p_i and p_{i-1} belong to the dialogue, there exist no utterances with either X or Y being either the receiver or the utterer.

A dialogue $\{p_0, p_1, \dots p_m\}$, $m \geq 0$, 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 constraints fire. In condition 3, t represents the time at which the incoming utterance is recorded by the receiving agent. By condition 4, the dialogue moves of a dialogue share the same dialogue identifier. By condition 5, dialogues are *atomic* and *interleaved*, where by *atomicity* we mean that each agent is involved in at most one dialogue at each time and by *interleaving* we mean that dialogue moves must alternate between the two agents within the dialogue. Conditions 4 and 5 are new and were not present in the definition of dialogues in [5]. The purpose of these new conditions is to avoid having to deal with multiple negotiation dialogues involving the same agent at the same time. To accommodate such concurrent dialogues we need to extend our set of subjects in the dialogue moves and to provide some form of concurrency control mechanisms, both of which are beyond the scope of this paper.

In Section 4 we propose a concrete agent framework that can produce dialogues, according to the above definition. In this paper we are interested in dialogues starting with the request of a resource R, defined as follows.

Definition 6. Given an agent system A equipped with $\mathcal{L}$, a *request dialogue* with respect to a resource R of an agent $X \in A$ is a dialogue $\{p_0, p_1, p_2, \dots\}$ between X and some other agent $Y \in A$ such that, for some $T \geq 0$,

- $p_0 = \mathit{tell}(X, Y, \mathit{request}(\mathit{give}(R)), D, T)$, and
- $\mathcal{K}_{X,T} \vdash \mathit{plan}(P, \mathit{Req}) \wedge R \in \mathit{Req} \wedge \neg \mathit{have}(R, T)$.

In the sequel, unless otherwise stated, by *dialogue* we mean *request dialogue.*

In order to obtain all the resources missing in a plan, a single dialogue might not be enough, in general, and a *sequence of dialogues* might be needed for an agent to obtain all the required resources. In order to produce sequences of dialogues, agents may run a *dialogue cycle*, having the following properties $\mathcal{P}$:

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

The idea behind 3 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 be executable. After the cycle, if successful in obtaining all the resources, the agent can execute the plan in its intention. One dialogue cycle with these properties is defined in [5]. In Section 4 we give a concrete implementation of an agent dialogue cycle with these properties.

Within our dialogue framework we can specify properties of the knowledge of agents and, as we will see in Section 4, of the behaviours of agents.

The following definitions give two useful properties of the knowledge of agents (KB stands for "knowledge base") paving the way towards building agents producing one and only one move in response to any non-final move of other agents.

Definition 7. An agent KB $\mathcal{K}_X$ is *deterministic* iff for each incoming dialogue move $p(D,T)$ which is a ground instance of a schema in $\mathcal{L}_X^{in}$, there exists at most one dialogue constraint in $\mathcal{K}_X$ which is triggered by $p(D,T)$ and which fires.

Definition 8. An agent KB $\mathcal{K}_X$ is *exhaustive* iff for each dialogue move $p(D,T)$ which is a ground instance of a schema in $\mathcal{L}_X^{in} \setminus \mathcal{F}(\mathcal{L})$, there exists at least one dialogue constraint that is triggered by $p(D,T)$ and which fires.

3 Specification of Agents' States

We can characterise the state of the agents in terms of their need and ownership of resources. To this end, 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 $\mathcal{I}_X$ (namely, if $\mathcal{I}_X{=}plan(P,Req)$, then $\mathcal{R}(\mathcal{I}_X){=}Req$). Also, let $\mathcal{R}_{X,T}$ be the set of resources X owns at time T. Then, for any resource R, we define the following predicates **a**, **m**, **n**, **i**, standing for *available*, *missing*, *needed*, and *indifferent*, respectively:

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

Assuming a formal definition of "have" and "need" in $\mathcal{K}_{X,T}$, e.g. $D.1 - D.4$ in section 2, the above **a**, **m**, **n**, **i** can be formally defined as follows:

- $\mathcal{K}_{X,T} \vdash \mathbf{a}(R,T)$ *iff* $\mathcal{K}_{X,T} \vdash have(R,T) \wedge \mathcal{K}_{X,T} \vdash \neg need(R,T)$
- $\mathcal{K}_{X,T} \vdash \mathbf{m}(R,T)$ *iff* $\mathcal{K}_{X,T} \vdash \neg have(R,T) \wedge \mathcal{K}_{X,T} \vdash plan(P,Req) \wedge R \in Req$
- $\mathcal{K}_{X,T} \vdash \mathbf{n}(R,T)$ *iff* $\mathcal{K}_{X,T} \vdash need(R,T) \wedge \mathcal{K}_{X,T} \vdash have(R,T)$
- $\mathcal{K}_{X,T} \vdash \mathbf{i}(R,T)$ *iff* $\mathcal{K}_{X,T} \vdash \neg have(R,T) \wedge \mathcal{K}_{X,T} \vdash \neg need(R,T)$

In the sequel, unless otherwise specified, we will not assume any specific definition of "have" and "need", but we will assume that, for any agent X, time T and resource R, either $\mathcal{K}_{X,T} \vdash have(R,T)$ or $\mathcal{K}_{X,T} \vdash \neg have(R,T)$ and either $\mathcal{K}_{X,T} \vdash need(R,T)$ or $\mathcal{K}_{X,T} \vdash \neg need(R,T)$.

State transitions, as a result of dialogues, can be characterised in terms of **a**, **m**, **n**, **i**.

In general, after a terminated request dialogue D between X and Y, initiated by X with respect to a resource R and an intention $\mathcal{I}_{\mathcal{X}}$, X might have been successful or not in obtaining the resource R from Y: Let us suppose that D started at time T and terminated at time $T1$, and let $T' > T1$. If D is successful, the knowledge bases of X and Y change as follows:

$\mathcal{K}_{X,T} \vdash \mathbf{m}(R,T)$ and $\mathcal{K}_{X,T'} \vdash \mathbf{n}(R,T')$

$\mathcal{K}_{Y,T} \vdash \mathbf{a}(R,T)$ and $\mathcal{K}_{Y,T'} \vdash \mathbf{i}(R,T')$.

If D has been unsuccessful, then neither agents' knowledge about resources changes, and X might decide to engage in a new dialogue to obtain R from a different agent.

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

Definition 9. An agent X is called *self-interested* if, for all request dialogues D, with respect to a resource R, between agents X and Y starting at time T and terminating at time $T1$, and for any time T' such that $T < T1 < T'$:

- if $\mathcal{K}_{X,T} \vdash \mathbf{a}(R,T)$ then either $\mathcal{K}_{X,T'} \vdash \mathbf{a}(R,T')$ or $\mathcal{K}_{X,T'} \vdash \mathbf{i}(R,T')$;
- if $\mathcal{K}_{X,T} \vdash \mathbf{m}(R,T)$ then either $\mathcal{K}_{X,T'} \vdash \mathbf{m}(R,T')$ or $\mathcal{K}_{X,T'} \vdash \mathbf{n}(R,T')$;
- if $\mathcal{K}_{X,T} \vdash \mathbf{n}(R,T)$ then $\mathcal{K}_{X,T'} \vdash \mathbf{n}(R,T')$;
- if $\mathcal{K}_{X,T} \vdash \mathbf{i}(R,T)$ then either $\mathcal{K}_{X,T'} \vdash \mathbf{i}(R,T')$ or $\mathcal{K}_{X,T'} \vdash \mathbf{a}(R,T')$;
- for all $\hat{R} \in \mathcal{R}(\mathcal{I}_X)$, $\hat{R} \neq R$, then
  if $\mathcal{K}_{X,T} \vdash \mathbf{a}(\hat{R},T)$ then either $\mathcal{K}_{X,T'} \vdash \mathbf{a}(\hat{R},T')$ or $\mathcal{K}_{X,T'} \vdash \mathbf{i}(\hat{R},T')$,
  if $\mathcal{K}_{X,T} \vdash \mathbf{m}(\hat{R},T)$ then either $\mathcal{K}_{X,T'} \vdash \mathbf{m}(\hat{R},T')$ or $\mathcal{K}_{X,T'} \vdash \mathbf{n}(\hat{R},T')$,
  if $\mathcal{K}_{X,T} \vdash \mathbf{n}(\hat{R},T)$ then $\mathcal{K}_{X,T'} \vdash \mathbf{n}(\hat{R},T')$;
  if $\mathcal{K}_{X,T} \vdash \mathbf{i}(\hat{R},T)$ then either $\mathcal{K}_{X,T'} \vdash \mathbf{i}(\hat{R},T')$ or $\mathcal{K}_{X,T'} \vdash \mathbf{a}(\hat{R},T')$.

Note that the main characteristic of self-interested agents is that they never give away resources they need.

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 [1] for $\mathcal{K}$, an abductive proof-procedure for agent reasoning, and an agent cycle for interleaving reasoning with observing and acting.

An abductive logic program is a triple $\langle T, Ab, IC \rangle$, where T is a *logic program*, namely a set of if-rules of the form $H \leftarrow C$, where H is an atom and C is a conjunction of literals, and every variable is universally quantified from the outside; IC is a set of *integrity constraints*, namely if-then-rules of the form $C \Rightarrow H$, where C is a conjunction of literals and H is an atom, and every variable is universally quantified from the outside; Ab is a set of ground atoms, whose predicates, that we call "abducibles", occur in T or IC, but not in the head H of any if-rule. The knowledge $\mathcal{K}$ of an agent can be represented as an abductive logic program as follows. Ground dialogue moves are represented as abducibles. Dialogue constraints are represented as integrity constraints. [3] The rest of the knowledge $\mathcal{K}$ of agents is split between the logic program and the integrity constraints.

[3] Note that dialogue constraints do not conform to the syntax of integrity constraints, but they can be written so that they do. A dialogue constraint $p(D,T) \wedge C \Rightarrow [\exists\, T'(\hat{p}(D,T') \wedge T < T')]$ can be rewritten as $p(D,T) \wedge C \Rightarrow q(D,T)$ together with an if-rule $q(D,T) \leftarrow \hat{p}(D,T') \wedge T < T'$ in the logic program component of the abductive logic program representing $\mathcal{K}$.

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 $ongoing{=}D, D \neq nil$ and either $\nexists\, p \in \texttt{input}$ such that $p = tell(X, a, Subject, D, T)$ or not *waiting*, then go to (iv);

(ii) [**record input**] if *ongoing=D* then **record an input** $b \in input$ such that $b = tell(X, a, Subject, D, T)$; otherwise, if *ongoing=nil* (reply to a new dialogue D'): **record an input** $p \in input$ (say, $p = tell(X, a, Subject, D', T)$), and set *ongoing=D'* to true;

$(ii.1)$ [**update dialogue status**] if p is a final move then set *ongoing=nil*, otherwise set *waiting* to false;

(iii) [**think**] **resume** ST by propagating the inputs, if any;

(iv) [**think**] **continue applying ST**, 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*, then go to step (vii);

$(iv.2)$ [**filter actions**] if *ongoing=D* and $D \neq nil$ and there does not exist an atomic action $tell(a, X, Subject, D, T)$ which can be executed at time $T' = T + r$, then go to step (vii);

(v) [**select action**] if *ongoing=D* and $D \neq nil$ then **select an atomic action** $tell(a, X, Subject, D, T'')$ that is an abducible computed by ST which can be executed at time $T' = T + r$ and instantiate T'' to T'; otherwise, if *ongoing=nil* (start a new dialogue D'): **select an atomic action** $tell(a, X, Subject, D', T'')$ that is an abducible computed by ST which can be executed at time $T' = T + r$, instantiate T'' to T' and set *ongoing=D'*;

$(v.1)$ [**update dialogue status**] if the selected action is a final move then set *ongoing=nil*, otherwise set *waiting* to true;

(vi) [**act**] **execute the selected action** at time T' and record the result;

(vii) [**cycle**] **cycle** at time $T' + 1$.

Fig. 1. Extended agent cycle for an agent a

We use the proof-procedure of [4], henceforth called ST, which is a modification of the IFF proof-procedure [3]. The IFF procedure forms the reasoning engine of the agents of the architecture of [2] that we adopt and modify, as we will explain later. IFF and ST share inference rules that allow *backward reasoning* with the logic programs and *forward reasoning*, modus ponens-like, with the integrity constraints. IFF proved inadequate for the purpose of producing dialogues as defined in section 2, in that it fails to produce the triggering and firing (production/active rule) behaviour of dialogue constraints that we describe in Section 2. This is due to the definition of *negation rewriting* in [3], that replaces an if-then-rule $[\neg A \Rightarrow B]$ with the disjunction $A \vee B$. To illustrate the problem, consider a dialogue (integrity) constraint $requested\_resource \wedge have\_resource \Rightarrow accept\_request$, where $have\_resource$ is defined as $have\_initially \wedge \neg gave\_away$. Given $requested\_resource$ and $have\_initially$, the constraint would be rewritten by IFF as $gave\_away \vee accept\_request$. The second disjunct could be selected, and the request accepted, even in the case the resource has already been given away.

ST modifies IFF in the handling of negation in if-then-rules. The modification involves replacing an if-then-rule $\neg A \Rightarrow B$ with the disjunction $[provable(A)] \vee [(A \Rightarrow false) \wedge B]$, which results in achieving a "negation as failure" behaviour for the negations in if-then-rules, more appropriate for the production/active rule behaviour required. For a detailed description of ST see [4].

In the architecture of [2], each agent, henceforth referred to as KS-agent, is an abductive logic program. The abducibles are *actions* to be executed as well as *observations* to be performed. In our case, actions are dialogue moves uttered by the agent, and observations are dialogue moves uttered by other agents. The behaviour of KS-agents is regulated by an observe-think-act cycle. The cycle starts at time T by observing and recording any observations from the environment. Then, the proof procedure is applied for r units of time. The amount of resources r available in the "think" phase is predefined. Forward reasoning is applied first, in order to allow for an appropriate reaction to the observations. Then, an action is selected and executed, in the "act" phase, taking care of recording the result.

We modify the agent cycle of [2] to enforce atomicity and interleaving of dialogues, as discusses in Section 2. The modified cycle for an agent a is in Figure 1. This is a modification of the original KS agent cycle in that filtering/updating steps are added, 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 are independent of the thinking part of the agent, namely the application of the proof-procedure. Also, in our 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 observations and whether or not to execute an action.

In order to enforce the properties of atomicity and interleaving, we use the following data structures:

– an input buffer, *input*, that contains all the observed predicates (namely the incoming dialogue moves) that have not yet been recorded by the agent;
– a flag *ongoing*, that either contains the identifier of the ongoing dialogue, or the constant nil, i.e., a new identifier that is not any dialogue identifier. Initially, *ongoing* is set to *nil*;
– a flag *waiting/0*, initially false, that is true if the agent is expecting a dialogue move from another agent within an ongoing dialogue.

Note that a move p with dialogue identifier D, observed during 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 p is a move of D. Also, the agent can utter a move p only if there is no ongoing dialogue (p is the first utterance of a new dialogue), or there is an ongoing dialogue D, and p is part of it.

In order for agents to generate sequences of dialogues with the properties $\mathcal{P}$ given in Section 2, the dialogue cycle of the agents may be specified by the following abductive logic program:

$$
\begin{array}{rl}
\textbf{(D.5)} & get\_all(M) \leftarrow M = \emptyset \\
& get\_all(M) \leftarrow R \in M \wedge get(R) \ \wedge R' = M \setminus \{R\} \ \wedge get\_all(R') \\
\textbf{(D.6)} & get(R) \leftarrow agents(A) \wedge ask\_agents(A, R) \\
\textbf{(D.7)} & ask\_agents(A, R) \leftarrow X \in A \wedge tell(a, X, request(give(R)), d(a, X, R, T), T) \wedge \\
& \qquad\qquad\qquad\qquad\qquad A' = A \setminus \{X\} \ \wedge process(X, A', R, d(a, X, R, T), T) \\
\textbf{(D.8)} & process(X, A, R, D, T) \leftarrow tell(X, a, accept(request(give(R))), D, T') \\
& process(X, A, R, D, T) \ \leftarrow \ tell(X, a, refuse(request(give(R))), D, T') \ \wedge \\
& ask\_agents(A, R) \\
\textbf{(D.9)} & to\_be\_asked(M) \leftarrow plan(\_, Req) \ \wedge missing(Req, \emptyset, M) \\
\textbf{(D.10)} & missing(Set, Acc, Out) \leftarrow Set = \emptyset \wedge Out = Acc \\
& missing(R \cup Set, Acc, Out) \leftarrow have(R, 0) \wedge missing(Set, Acc, Out) \\
& missing(R \cup Set, Acc, Out) \leftarrow \neg have(R, 0) \wedge missing(Set, Acc \cup \{R\}, Out) \\
\textbf{(D.11)} & X \in Y \leftarrow Y = [X|X'] \\
& \\
\textbf{(IC.4)} & to\_be\_asked(M) \Rightarrow get\_all(M) \\
\textbf{(IC.5)} & ask\_for(A, R) \wedge A = \emptyset \Rightarrow \bot
\end{array}
$$

The two integrity constraints are used to start the negotiation process (IC.4) from the missing resources of the agent's plan, and to determine failure (IC.5) once the agent has failed to get a specific resource after asking all the agents in the system (but itself). Note that definition D.7 performs the allocation of a unique dialogue identifier to a newly started dialogue. This identifier is a function(d) of the agent starting the dialogue, the receiver of the request, the requested resource and the time of the utterance. Note also that definition D.11 assumes a list-like implementation of sets.

5 Formal Results

We define the following agent types and give some formal results.

Definition 10. An abductive agent X is exhaustive and deterministic if X produces one and only one ground move, in response to any non-final move of other agents of which X is the receiver.

Definition 11. An *abductive agent* is an agent whose *knowledge base* $\mathcal{K}$ is an abductive logic program, with, in particular, the dialogue moves in the negotiation language $\mathcal{L}$ of the agent represented as abducibles in the abductive logic program, the dialogue constraints in $\mathcal{K}$ represented as integrity constraints in the abductive logic program, equipped with the *ST abductive proof-procedure* as its reasoning engine with the entailment $\mathcal{K}$ is equipped with provided by provability in ST, and with the *extended agent cycle*.

An $\mathcal{N}$*-agent* is a particular instance of an abductive agent whose logic program consists of definitions D.1-D.4, appropriate definition for **a**, **m**, **n**, **i**, and integrity constraints IC.1-IC.3.

An $\mathcal{N}^+$*-agent* is an $\mathcal{N}$-agent whose knowledge is extended by the dialogue cycle given by D.5-D.11 and IC.4-IC.5.

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

Theorem 2 $\mathcal{N}$-agents are self-interested agents.

Theorem 3 If X is an abductive agent and $\mathcal{K}_X$ is exhaustive and deterministic then X is exhaustive and deterministic.
Corollary 1. $\mathcal{N}$-agents are exhaustive and deterministic.
Theorem 4 $\mathcal{N}^+$-agents are self-interested agents.
Theorem 5 If X is an $\mathcal{N}^+$-agent, then $\mathcal{K}_X$ is exhaustive and deterministic.
Corollary 2. $\mathcal{N}^+$-agents are exhaustive and deterministic.
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 X is exhaustive and deterministic, then X 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 exactly one agent is guaranteed to produce only one dialogue move, after a dialogue is initiated by either agent, until a final move is made, thus producing a dialogue that conforms to the definition given in Section 2.
Theorem 6 (*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 the sequence of all 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 5.

6 Conclusions

We have presented a framework for agent dialogue and negotiation based on abductive reasoning. In order to provide an operational model for the framework, we introduced a modified version of the agent cycle of [2], with respect to the proof-procedure introduced in [4], rather than the IFF that [2] chooses to adopt. Our model is able to produce sequences of negotiation dialogues that conform to the given definition.

Following other related work on agent dialogue, we defined dialogue in a two-agents setting. It is possible to extend it to accommodate multi-part negotiation schemes, e.g. by means of auction-like interaction patterns, but this goes beyond the scope of this paper. Some work done in this direction is that of [7]. Among the current approaches to negotiation via dialogue, our work is to the best of our knowledge the only one that is based on abductive logic programming. An approach to agent communication and negotiation that makes use of abduction is proposed in [8], where the authors use deduction to derive information from a received message, and abduction to obtain proposals in reply to requests. In our framework, abduction is used not only to formulate replies to requests, but it is the main form of agent reasoning. This allows us to prove Theorem 6.

Kraus et al. [9], Amgoud et al. [10] and Parsons et al. [11] are some of the argumentation-based approaches proposed to agent negotiation. In general, such approaches, focused on argumentation and persuasion, lead to different results from ours, being somewhat more descriptive, and not aiming to determine specific properties of the policy regulations.

The innovative contribution of our work is in the definition of the operational model, including the proposed agent cycle and dialogue cycle, and in the results that apply in the general case of abductive agents and in the specific case of $\mathcal{N}$-systems. We consider this to be an important achievement, because it provides a high-level specification of a system that is directly implementable, and that guarantees certain properties to hold. On the other hand, since we prose a complete approach to a complex problem such as resource reallocation, a weak point of this work could be its scalability, if we want to apply it to the case, for instance, of agents that can can dynamically modify their plans. To cope with huge search spaces, we believe that we could profitably build on the formal results of our work and apply incomplete search methods like those based on metaheuristics. Some work has already been done towards the combination of the two approaches [12]. In the future, we aim at extending our framework to cope with exchange of uncountable resources, or of resources that may have a different "meaning" for autonomous reasoning agents, such as information.

Acknowledgements

We would like to thank Bob Kowalski for his interest in this work and for helpful discussions and suggestions about the dialogue cycle, and the anonymous referees for their helpful comments. This work has been 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. Kakas, A.C., Kowalski, R.A., Toni, F.: The role of abduction in logic programming. Handbook of Logic in AI and Logic Programming **5** (1998) 235–324
2. Kowalski, R.A., Sadri, F.: From logic programming to multi-agent systems. Annals of Mathematics and AI (1999)
3. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming (1997)
4. Sadri, F., Toni, F.: Abduction with negation as failure for active and reactive rules. In Proc. AI*IA'99. LNAI 1792, Springer-Verlag (2000) 49–60
5. Sadri, F., Toni, F., Torroni, P.: Dialogues for negotiation: agent varieties and dialogue sequences. In: Intelligent Agents VIII. LNAI 2333, Springer-Verlag (2002)
6. Sadri, F., Toni, F., Torroni, P.: Logic agents, dialogues and negotiation: an abductive approach. In: Proc. AISB'01 Convention, York, UK. (2001)
7. Torroni, P., Toni, F.: Extending a logic based one-to-one negotiation framework to one-to-many negotiation. In: ESAW II. LNAI 2203, Springer-Verlag (2001)
8. Hindriks, K., de Boer, F., van der Hoek, W., Meyer, J.: Semantics of communicating agents based on deduction and abduction. In: Proc. CABS. (1999)
9. Kraus, S., Sycara, K., Evenchik, A.: Reaching agreements through argumentation; a logical model and implementation. Artificial Intelligence **104** (1998) 1–69
10. Amgoud, L., Parsons, S., Maudet, N.: Arguments, dialogue and negotiation. In: Proc. 14th ECAI, Berlin, Germany, IOS Press (2000)
11. Parsons, S., Sierra, C., Jennings, N.R.: Agents that reason and negotiate by arguing. Journal of Logic and Computation **8** (1998) 261–292
12. Roli, A., Torroni, P.: Logics, local search and resource allocation (short paper). In: Proc. STAIRS, Lyon, France. (2002)