Università degli Studi di Bologna
DEIS

A study on the termination of negotiation dialogues

Paolo Torroni

November 7, 2001

DEIS Technical Report no. DEIS-LIA-01-008 *LIA Series no. 52*

A study on the termination of negotiation dialogues

Paolo Torroni

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

November 7, 2001

Abstract. Dialogue represents a powerful means to solve problems using agents that have an explicit knowledge representation, and exhibit a goal-oriented behaviour. In recent years, computational logic gave a relevant contribution to the development of Multi-Agent Systems, showing that a logic-based formalism can be effectively used to model and implement the agent knowledge, reasoning, and interactions, and can be used to generate dialogues among agents and to prove properties such as termination and success. In this paper, we discuss the meaning of termination in agent dialogue, and identify a trade-off between ensuring dialogue termination, and therefore robustness in the agent system, and achieving completeness in problem solving. Then, building on an existing negotiation framework, where dialogues are obtained as a product of the combination of the reasoning activity of two agents on a logic program, we define a syntactic transformation of existing agent programs that aims at ensuring termination in the negotiation process. We show how such transformations can make existing agent systems more robust against possible situations of non-terminating dialogues, while reducing the class of reachable solutions in a specific application domain, that of resource reallocation.

Keywords: *Multi-Agent Systems, Negotiation, Dialogue, Termination, Computational Logic, Abduction*

1 Introduction

Dialogue is one of the most flexible interaction patterns in Multi-Agent Systems, being something between completely fixed protocols and totally free conversations

[3]. Intuitively, a dialogue is not the (purely reactive) question-answer sequence of a client server architecture framed in a rigid protocol, but is rather a kind of interaction grounded on an expressive enough knowledge representation. Dialogues start, in general, from the need to achieve an explicit goal. The goal of initiating a dialogue could be for example to persuade another party, to find an information, to verify an assumption, and so on [15]. In the case of negotiation dialogues, for instance, agents need to negotiate because they operate in environments with limited resource availability, and the goal of a dialogue is to obtain a resource. This general idea applies to different scenarios with different meanings.

In recent years, computational logic gave a relevant contribution to the development of Multi-Agent Systems [11, 4], and proved effective to model and implement the agent knowledge, reasoning, and interactions. Work on argumentation and persuasion, that under certain circumstances are considered suitable techniques and strategies to support conversation and goal achievement, led to many argumentative frameworks [1, 9, 10]. They often try and embrace very hard problems and result in very good descriptive models for them, but lack most of the times an execution model. In [13], Sadri et al. described a logic-based approach to negotiation which does not take into account persuasion and argumentation in their classic understanding, but which on the other hand allows for proving properties such as termination, correctness and completeness. The strength of such approach is in that it proposes an execution model that can be used to achieve an implementation of the system.

In the area of negotiation, there are some general results about dialogue properties, like termination and success. In [16], the authors consider the use of logic-based languages for negotiation, and identify two important computational problems in the use of logic-based languages for negotiation – the problem of determining if agreement has been reached in a negotiation, and the problem of determining if a particular negotiation protocol will lead to an agreement.

In this paper, we tackle the problem of termination of agent dialogues. We start by discussing the meaning of termination in agent dialogue, and show a trade-off between ensuring termination of dialogues, and therefore robustness in the agent system, and achieving completeness in problem solving. Then, we show how such ideas apply in practice to a concrete case of negotiation framework. Building on [12], where agent dialogue is obtained as a product of the combination of the reasoning activity of two agents performing an abductive derivation on a logic program, we show how the agent programs can be transformed in order to ensure termination in the negotiation process, and we define three different degrees of such transformation. We show how they make the agents more robust against possible situations of non-terminating dialogues, while reducing the class of solutions that can be found in a specific application domain, that of resource reallocation.

2 Agent dialogue termination: robustness vs. completeness

In [2] agent societies are categorized in terms of openness, flexibility, stability and trustfulness, and it is claimed that whereas open societies support openness and flexibility, closed societies support stability and trustfulness. The author suggests two classes of societies (semi-open and semi-closed), that balance the trade-off between these aspects, because in many situations there is a need for societies that support all of them.

In the case of dialogue, the problem of determining such trade-off still holds, because the dialogue can be used as a means to let heterogeneous agents communicate, despite the differences among them, and without necessarily sticking to a given protocol. On the other hand, if we let agents openly join societies, with no control on the individuals that access them, problems could arise from their diversity. For instance, dialogues can last forever.

This work focuses on negotiation dialogues. Let us start by describing what we intend by dialogue, and let us do it is by example, before we define it formally in the next section. In the following dialogue, inspired by [10], agent a will ask agent b for a resource (a *nail*), needed to carry out a task (i.e., to hang a picture). Once the request is refused, a asks b the reason why, with the purpose of acquiring additional information and finding an alternative solution to her goal.

Example 1
$tell(a, b, request(give(nail)), 1)$
$tell(b, a, refuse(request(give(nail))), 2)$
$tell(a, b, challenge(refuse(request(give(nail)))), 3)$
$tell(b, a, justify(refuse(request(give(nail))), not\ have(nail)\}), 4)$ $\square$

In general, a dialogue is a sequence of alternative *dialogue moves*, or *performatives*, where a performative is a message in the form *tell(Sender, Recipient, Subject, Time)*. Time, in particular, is understood as a transaction time. The concept of *termination* of a negotiation dialogue can be recovered into the idea that at a certain point an agent makes a final move [16]. Of course, the other agent is supposed to recognize that such move is intended to terminate the dialogue. If no agent makes any final move, both agents could keep exchanging messages, without getting to an end. Example 2 shows a dialogue between two (particularly overpolite) agents that keep exchanging greetings.

Example 2
$tell(a, b, hello, 1)$
$tell(b, a, hello, 2)$

$tell(a, b, hello, 3)$
$\ldots$ $\square$

The situation of Example 2 could be due to the fact that both agents' programs force them to reply to an incoming greeting with an equal greeting. We can imagine that loop conditions of the like could unpredictably arise each time we put together, in an open society, agents that were independently programmed. An obvious solution to this problem could be to force agents not to tell the same thing twice. But this measure does not really solve the problem, as Example 3 shows: agents could keep exchanging slightly different messages, and still get stuck in a loop condition.

Example 3
$tell(a, b, hello(1), 1)$
$tell(b, a, hello(2), 2)$
$tell(a, b, hello(3), 3)$
$\ldots$ $\square$

Then, we could think to introduce a more restrictive measure, e.g., based on message patterns, that prevents agents from telling a message whose "pattern" is the same as a previous one in the same dialogue.[1] But this could result in preventing agents from finding solutions (or *agreements*, in the case of negotiation), that could be found otherwise. Example 4 below shows a possibly successful dialogue that would solve a resource reallocation problem (agent a obtains a *screw* from b and therefore can execute a plan to achieve her goal of hanging a picture). Such dialogue would not be permitted if agent b was prevented from making the move 'propose an exchange' (such is the meaning of the *promise* performative, that we inherit from [1]) twice.

Example 4
$tell(a, b, request(give(nail)), 1)$
$tell(b, a, challenge(request(give(nail))), 2)$
$tell(a, b, justify(request(give(nail)), \ldots), 3)$
$tell(b, a, promise(give(bluetac), \ldots), 4)$
$tell(a, b, refuse(promise(give(bluetac), \ldots)), 5)$
$tell(b, a, promise(give(screw), \ldots), 6)$
$tell(a, b, accept(promise(give(screw), \ldots)), 7)$ $\square$

Still, unless we consider very generic (and therefore very restrictive) patterns, the threat of non-termination remains. Example 5 could evoke a familiar situation

[1] the pattern could be, in this case: *tell(a, b, hello(_), _)*, where the *underline* indicates whatever ground expression.

to those who have spent some moments of their life dealing with small children... In the example, the *challenge* performative, also derived from [1], has the meaning of asking for a justification of what the dialogue partner just said.

Example 5
$tell(a, b, request(give(nail)), 1)$
$tell(b, a, challenge(request(give(nail))), 2)$
$tell(a, b, justify(request(give(nail)), \ldots), 3)$
$tell(b, a, challenge(justify(request(give(nail)), \ldots)), 4)$
$\ldots \quad \square$

In the end, we realize intuitively what follows: the more we reduce the set of dialogue moves that the agents can exchange in the course of a dialogue, the more we reduce the universe of reachable solutions of a negotiation problem. We think that the choice about to which extent the dialogue should be constrained to certain patterns must be left to the system designer(s).

In the next section, we describe a concrete negotiation framework, firstly introduced in [13] and further extended in [12], that makes use of a logic formalism. In such framework, the course of agent dialogues is ruled by the knowledge expressed into the agents' abductive logic programs. Based on that, we introduce several syntactic transformations of such programs, that make them robust in various degrees against non-termination, and reflect the above discussed trade-off. Finally, in Section 4 we prove a theorem that determines a bound in the maximum length of a dialogue, measured in terms of number of exchanged messages, and we extend such result to dialogue sequences.

3 Abduction and negotiation

The dialogue framework that we are going to sketch in this section is derived from [13]. It is composed of a *knowledge representation* including an abductive logic program (ALP), a *language*, a *proof-procedure*, and a *communication layer*. Agents are provided with a suitable architecture, including in particular a *planner*. The communication layer is a shared blackboard where agents can post / retrieve messages. As far as the knowledge representation, we will only say here that agents have a (declarative) representation of goals $\mathcal{G}$, beliefs $\mathcal{B}$, and intentions $\mathcal{I}$, i.e., plans to achieve goals. Agents will access their beliefs by means of predicates such as $have(Resource)$, $need(Resource)$, and in a similar way to their intentions ($intend(Intention)$). The purpose of negotiation is for the agent to obtain the missing resources, while retaining the available ones that are necessary for the plan in its current intention. As the focus of the paper is on the termination issue, and for space limitations, we will not describe the framework in detail here, although we need to give some intuition on the abductive proof-procedure adopted by the agents, in order to prove the termination results of Section 4.

3.1 A negotiation framework

In its classical understanding, abduction is a reasoning mechanism that allows to find a suitable explanation to a certain observation or goal, based on an abductive program. In general, an abductive program is expressed in terms of a triple $\langle P, \mathcal{A}, IC \rangle$, where P is a logic program, $\mathcal{A}$ is a set of *abducible* predicates, i.e., *open* predicates which can be used to form explaining sentences, and IC is a set of integrity constraints. Given a goal g, abduction aims at finding a set $\Delta \subseteq \mathcal{A}$ of abducible predicates that can be supposed true and thus enlarge P, in order to *entail* g. The adoption of automatic proof procedures such as that of [5] or [6], supported by a suitable agent cycle such as for instance the *observe-think-act* of [8], will implement a concrete concept of entailment with respect to knowledge bases expressed in abductive logic programming terms. The execution of the proof procedure within the agent cycle allows to produce hypotheses (explanations) that are consistent with the agent constraints, IC, when certain phases of the agent cycle are reached. Constraints play a major role in abduction, since they are used to drive the formulation of hypotheses and prevent the procedure from generating wrong explanations to goals. For this reason, abduction has been originally used for diagnosis and expert systems.

In recent times, many different understandings of abductive reasoning have been conceived. Abduction has been used, e.g., for planning and scheduling, where the 'hypotheses' that can be made refer to task scheduling, and the constraints can be used, e.g., to prevent task overlapping and resource conflict. In an argumentation framework, abduction has been proposed to build arguments out of a knowledge base [7]. In [13], abduction has been used to model agent dialogue, following an argumentative approach. In particular, the abducible hypotheses are dialogue performatives. The abductive agent program is provided with *dialogue constraints* that are fired each time the agent is expected to produce a dialogue move, e.g., each time another agent sends him a request for a resource. Such move is then produced as a hypothesis that must be assumed true in order to keep the knowledge base consistent. The agent knowledge is considered consistent if the agent replies to a partner's moves, according to the current status of her knowledge base. The use of abduction in the agent dialogue context, as opposed to other (less formal) approaches, has several advantages, among which the possibility to determine properties of the dialogue itself, and the 1-1 relationship holding between specification and implementation, due to the operational semantics of the adopted abductive proof-procedure.

In the following we will show a dialogue constraint, taken from a very simple agent program:

Example 6
$tell(X, a, \mathbf{request}(\mathbf{give}(R)), T) \ \wedge \ have(R, T)$
$\quad \Rightarrow \qquad tell(a, X, \mathbf{accept}(\mathbf{request}(\mathbf{give}(R))), T+1) \quad \Box$

Constraints here are expressed in terms of condition-action rules, leading in this particular case from the perception of another agent's dialogue move (*observation* phase) to the expression of a new dialogue move (*action* phase). For instance, the first constraint of the example reads: 'if agent a receives a request from another agent, X, about a resource R that she has, then a tells X that she will accept the request'. Such rules are interpreted (*think* phase) by the IFF abductive proof-procedure, framed in an *observe-think-act* agent cycle [8].

3.2 IFF-terminating programs

In this paper, we will not make any concrete assumptions on the syntax of the language of the knowledge base of agents, except for assuming that it contains notions of *literal*, *complement* of sentences, *true* and *false*, and that such language is equipped with a notion of *entailment*, such that, for every ground literal in the language, either the literal or its complement is entailed, and such that no literal and its complement are entailed at the same time.

The IFF [5] is a rewriting abductive proof-procedure consisting of a number of inference rules. Two basic inference rules are *unfolding* (backward reasoning), and *propagation* (forward reasoning). Implications are obtained by repeatedly applying the inference rules of the proof procedure to either an integrity constraint in the given program (ALP), or to the result of rewriting negative literals *not* A as $false \ \rightarrow \ A$. We will not describe the proof-procedure in detail here, but we will focus on the issue of proof termination, and give a characterization of the class of *IFF-terminating* programs, i.e., of those abductive logic programs for which all IFF-trees for grounded queries are finite.

Intuitively, the reason why a program is not IFF-terminating can be recovered into the presence in the program of rules / constraints whose combination leads to infinite propagation or unfolding. It is possible to identify three cases, that can be generalized.[2] In the following, `p` and `q` represent literals.

1) *unfolding + unfolding*
   `p` $\leftrightarrow$ `q`
   `q` $\leftrightarrow$ `p`

[2]Here, for the sake of simplicity, we consider only ground programs, thus assuming that P and IC have already been instantiated. The results could be generalized to non-ground programs.

2) *unfolding + propagation*
   p $\leftrightarrow$ q
   q $\Rightarrow$ p

3) *propagation + propagation*
   p $\Rightarrow$ q
   q $\Rightarrow$ p

In all cases, p unfolds / propagates to q and vice versa, ad infinitum. In order to characterize a class of programs that terminate, we define the property of *aciclicity*, tailored to the case of ALP in relationship to the IFF proof procedure (IFF-aciclicity). In fact, if an ALP is IFF-acyclic, then it is IFF-terminating. An ALP is IFF-acyclic if we can found a *level mapping* $||$ such that:

1. for every ground instance of every clause (if-definition) in P, say $A \leftarrow L_1, \ldots, L_i, \ldots, L_n$, we have $|A| > |L_i|$ (i.e., P is acyclic)

2. for every ground instance of every integrity constraint in IC, say $L_1, \ldots, L_j, \ldots, L_m \Rightarrow A$, we have:

   - $|L_j| > |A|$
   - if K is a negative literal, say *not* M in the body of the integrity constraint, then $|L_j| > |M|$

The presence of such level mapping ensures that the situations above do not occur in an agent program. We will call IFF-acyclic programs *acceptable*: in fact, we cannot guarantee termination if an agent gets stuck in an infinite branch of the derivation tree, producing therefore no dialogue move at all. In the following, we will always require that the agent programs are acceptable. This work builds on these results and identifies a class of agent programs that ensure the termination of negotiation dialogues and sequences of dialogues.

3.3 Dialogues

Let us now formally define what we intend by dialogue. In the sequel, capital letters stand for variables and lower-case letters stand for ground terms.

Definition 1 (*performative or dialogue move*)
A *performative* or *dialogue move* is an instance of a schema of the form *tell*(X, Y, **Subject**, T), where X is the *utterer* and Y is the *receiver* of the performative, and T is the *time* when the performative is uttered. **Subject** is the *content* of the performative, expressed in some given *content language*.

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

An example of language for negotiation is the following, taken from [13]:

$$\begin{aligned}\mathcal{L}_1 = \{\ & tell(X, Y, \mathbf{request}(\mathbf{give}(Resource)), T),\\ & tell(X, Y, \mathbf{accept}(Move), T),\\ & tell(X, Y, \mathbf{refuse}(Move), T),\\ & tell(X, Y, \mathbf{challenge}(Move), T),\\ & tell(X, Y, \mathbf{justify}(Move, Support), T),\\ & tell(X, Y, \mathbf{promise}(\mathbf{give}(Resource),\\ & \mathbf{give}(Resource')), T)\ \}\end{aligned}$$

The initial and final moves of $\mathcal{L}_1$ are:

$$\begin{aligned}\mathcal{I}(\mathcal{L}_1) = & \{tell(X, Y, \mathbf{request}(\mathbf{give}(Resource)), T)\}\\ \mathcal{F}(\mathcal{L}_1) = & \{\ tell(X, Y, \mathbf{accept}(\mathbf{request}(\mathbf{give}(Resource))), T),\\ & \quad tell(X, Y, \mathbf{accept}(\mathbf{promise}(\mathbf{give}(Resource),\\ & \mathbf{give}(Resource'))), T),\\ & \quad tell(X, Y, \mathbf{refuse}(\mathbf{request}(\mathbf{give}(Resource))), T),\\ & \quad tell(X, Y, \mathbf{refuse}(\mathbf{promise}(\mathbf{give}(Resource),\\ & \mathbf{give}(Resource'))), T)\ \}.\end{aligned}$$

As in this paper we are interested in negotiation for the exchange of resources, we will assume that there always exists a **request** move in the initial moves of any language for negotiation.

Definition 3 (*agent system*)
An *agent system* is a finite set A, where each $x \in A$ is a ground term, representing the name of an agent, equipped with a *knowledge base* $\mathcal{K}(x)$.

We will assume that in an agent system, the agents share a common language for negotiation as well as a common content language. For a given agent $x \in A$, where A is equipped with $\mathcal{L}$, we define the sets $\mathcal{L}^{in}(x)$, of all performative schemata of which x is the receiver, but not the utterer; and $\mathcal{L}^{out}(x)$, of all performative schemata of which x is the utterer, but not the receiver. Note that we do not allow for agents to utter performatives to themselves. In the sequel, we will often omit x, if clear from the context, and simply write $\mathcal{L}^{in}$ and $\mathcal{L}^{out}$. In our ALP framework, outgoing performatives are *abducibles*, which implies that no definition for them is allowed. In other words, there does not exist a rule in the agent program that contains an outgoing performative in the head.

Negotiation protocols can be specified by sets of 'dialogue constraints', defined as follows:

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: $p(T) \wedge C \Rightarrow \hat{p}(T+1)$, where

- $p(T) \in \mathcal{L}^{in}(x)$ and $\hat{p}(T+1) \in \mathcal{L}^{out}(x)$,
- the utterer of $p(T)$ is the receiver of $\hat{p}(T+1)$, and
- C is a conjunction of literals in the language of the knowledge base of x.[3]

Any variables in a dialogue constraint are implicitly universally quantified from the outside. The performative $p(T)$ is referred to as the *trigger*, $\hat{p}(T+1)$ as the *next move* and C as the *condition* of the dialogue constraint.

The intuitive meaning of a dialogue constraint $p(T) \wedge C \Rightarrow \hat{p}(T+1)$ of agent x is as follows: if at a certain time T in a dialogue some other agent y utters a performative $p(T)$, 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 $\hat{p}(T+1)$, with y as receiver, at the next time $T+1$. This behaviour of dialogue constraints can be achieved by employing an automatic proof procedure such as that of [5] within an *observe-think-act* agent cycle [8], as we said before. The execution of the proof procedure within the agent cycle allows to produce dialogue moves immediately after a dialogue constraint is fired. A concrete example of a dialogue constraint allowing an agent x to accept a request is that of Example 6, where the trigger is *tell(Y, a, request(give(R)), T)*, the condition is *have(R, T)*, and the next move is *tell(a, Y, accept(request(give(R))), T + 1)*.

We will refer to the set of dialogue constraints associated with an agent $x \in A$ as $\mathcal{S}(x)$, and we will call it the *agent program* of x. We will often omit x if clear from the context or unimportant. In order to be able to generate a dialogue, two agent programs must be properly combined, that exhibit two important properties: *determinism* and *exhaustiveness*. We say that an agent program is deterministic and exhaustive if it generates exactly one *next move* $\hat{p}(T+1)$ for a given trigger $p(T)$, and a condition C, except when $p(T)$ is a final move. We call $\mathcal{P}$ the space of

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

acceptable, exhaustive, and deterministic agent programs. Some examples of such programs can be found in [12].

Definition 5 (*dialogue*)
A *dialogue* between two agents x and y is a set of ground performatives, $\{p_0, p_1, \ldots\}$, such that, for some given $t \geq 0$:

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 $p_{i-1} \wedge C \Rightarrow p_i \in \mathcal{S}(\alpha)$ such that $\mathcal{K}(\alpha) \wedge p_{i-1}$ entails C.

By condition 1, a dialogue is in fact a *sequence* of performatives. By condition 2, agents utter alternatively 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. 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 a performative in $\mathcal{F}(\mathcal{L})$.

Intuitively, a dialogue should begin with an initial move, according to the given language for negotiation. The kind of dialogue that is relevant to our purposes is that started with a request of a resource R. In the knowledge representation that we chose in the reference framework, we call *missing(Rs)* the set of resource that an agent is missing before she can start executing an intention $\mathcal{I}$. A *request* dialogue will be initiated by an agent x whose intention $\mathcal{I}$ contains R in its set of *missing* resources.

Definition 6 (*request dialogue*)
A *request dialogue* with respect to a resource R and an intention $\mathcal{I}$ of agent x is a dialogue $\{p_0, p_1, p_2, \ldots\}$ between x and some agent $y \in A$ such that, for some $t \geq 0$,

- $p_0 = tell(x, y, request(give(R)), t)$,
- $missing(Rs) \in \mathcal{I}$ and
- $R \in Rs$.

As a consequence of a dialogue, the agent's intentions might change. According to the way intentions are modified, a classification of types of terminated request dialogues is given in [12]. In the sequel, we will assume that a terminated request dialogue, for a given resource R and intention $\mathcal{I}$, returns an intention $\mathcal{I}'$.

4 Ensuring termination

We can consider the dialogue as a particular IFF derivation, obtained by interleaving resolution steps made by the two agents. Therefore, we can extend the termination results of 3.2 to dialogue programs.

4.1 Terminating dialogue programs

We argue that the possible reasons for an infinite derivation tree are still, *mutatis mutandis*, those of Section 3.2. Let us consider again the three cases, and see if they can occur when the knowledge is distributed between two agents, a and b. We put on the left side of each rule / constraint the name of the agent whose program contains it. We assume that the agents' programs are *acceptable.*

1) *unfolding + unfolding*
   [a] $\mathtt{p} \leftrightarrow \mathtt{q}$
   [a] $\mathtt{q} \leftrightarrow \mathtt{p}$

2a) *unfolding + propagation*
   [a] $\mathtt{p} \leftrightarrow \mathtt{q}$
   [a] $\mathtt{q} \Rightarrow \mathtt{p}$

2b) *unfolding + propagation*
   [a] $\mathtt{p} \leftrightarrow \mathtt{q}$
   [b] $\mathtt{q} \Rightarrow \mathtt{p}$

3) *propagation + propagation*
   [a] $\mathtt{p(T)} \wedge \mathtt{C_p[T]} \Rightarrow \mathtt{q(T+1)}$
   [b] $\mathtt{q(T)} \wedge \mathtt{C_q[T]} \Rightarrow \mathtt{p(T+1)}$

(1) and (2a) are both forbidden by the program *acceptability* requirement. (2b) does not represent a possible cause of non-termination:[4] since `q` is not abducible, therefore it is not possible to communicate it to b. As for the third case, let us consider, as `p(T)`, *tell(b, a, Subject, T)*, and as `q(T)`, *tell(a, b, Subject, T)*. This is apparently a threat for termination. For instance, this is the case in Example 2, where *Subject* is *hello*. We could therefore introduce some restrictions to the dialogue protocols, in order to prevent such situation, at the cost of a reduction of the space of reachable solutions, as explained before by examples.

[4]The only way to pass the computation thread from an agent a to her dialogue partner b is through an abducible representing a dialogue move in the head of one of a's diaolgue constraints.

4.2 Three degrees of restrictions

In order to try and prevent *propagation* from causing infinite dialogue move generation (case 3), we should make sure that the same integrity constraint of an agent program is not triggered infinitely many times. To this purpose, we define a transformation $\mathcal{T}$ that maps an element $\mathcal{S} \in \mathcal{P}$ of the domain of (acceptable, exhaustive and deterministic) agent programs into another element, $\mathcal{T}(\mathcal{S})$ in the same domain. $\mathcal{T}$ is defined as follows:

Definition 7 (*Agent program transformation*)
Given a language $\mathcal{L}$, an agent x and an agent program $\mathcal{S}(x) \in \mathcal{P}$, the transformation $\mathcal{T}$ with respect to a given set of literals (*restriction*) $p_{check}(T)$, $\mathcal{T}(p_{check}(T)) : \mathcal{P} \rightarrow \mathcal{P}$ is defined in the following way:

$$\begin{array}{l}
\forall ic \in \mathcal{S}(x), ic = p(T) \ \wedge \ C[T] \ \ \Rightarrow \ \ \hat{p}(T+1), \mathcal{T}(ic) \stackrel{def}{=} \\
\quad \left\{ \begin{array}{lll} p(T) \ \wedge \ C[T] \ \wedge \ not \ p_{check}(T) & \Rightarrow & \hat{p}(T+1) \\ p(T) \ \wedge \ C[T] \ \wedge \ \qquad p_{check}(T) & \Rightarrow & f(T+1) \end{array} \right. \\
where \ f(T+1) \in \mathcal{F}(\mathcal{L}), \qquad\qquad if \ \hat{p}(T+1) \in \mathcal{F}(\mathcal{L}); \\
\quad ic, \qquad\qquad otherwise.
\end{array}$$

Therefore, there is a 1-1 correspondence between a *restriction* $p_{check}(T)$ and a transformation function $\mathcal{T}$. We will called *restricted according to* $\mathcal{T}$ the programs that are elements of the co-domain of $\mathcal{T}$. If an acceptable program is restricted according to $\mathcal{T}$, when the partner agent produces a move triggering a *dialogue constraint* whose condition and restriction are both verified, the agent will jump to a final state, thus interrupting the dialogue. It is easy to prove that, given a transformation function $\mathcal{T}$ associated with a restriction $p_{check}(T)$, if the $p_{check}(T)$ is ground for all possible instantiations of $p(T) \ \wedge \ C[T]$, $\mathcal{T}$ transforms exhaustive and deterministic programs into exhaustive and deterministic programs, i.e., $\mathcal{T}$ maps from $\mathcal{P}$ to $\mathcal{P}$.

The choice of the restriction can be made in several ways; we will define here three different kinds of restrictions, that formally reflect the considerations of Section 2.

(i) The check is made on ground instances of predicates,

(ii) The check is made on predicate patterns,

(iii) The check is made against an ordering.

Let us consider them one by one. Case (*i*), *check made on ground instances of predicates*, restricts the applicability of a dialogue constraint by preventing it from being triggered twice *by the same instance of dialogue move* at different times. This is in line with the characterization of dialogue given in [13], and prevents situations of infinite 'pure' loop, such as the one in Example 2, generated by the constraint $tell(X, a, hello, T) \Rightarrow tell(a, X, hello, T + 1)$ of agent a. The restriction in this case could be:

$$p^{(i)}_{check}(T) = tell(X, a, hello, T1) \ \wedge \ T1 < T$$

This does not guarantee termination in a finite number of steps, though, as shown by Example 3. The *check on predicate patterns* could be implemented in that case by the restriction

$$p^{(ii)}_{check}(T) = tell(X, Y, hello(\_), T1) \ \wedge \ T1 < T$$

The case of check made on predicate patterns is a more restrictive policy, that has the already mentioned drawbacks and limitations. In particular, the situation of Example 5 could be caused by an agent that contains in her program the following dialogue constraint: $tell(b, a, Anything, T) \Rightarrow$
$tell(a, b, challenge(Anything), T + 1)$.

A solution to this could be to establish an *ordering* among the dialogue constraints of an agent. Before going on, let us point out that the problem of non-termination is of the agent that starts the negotiation dialogue, although our results are independent on this. In fact, if it is true that the dialogue can be terminated by either agent, on the other hand, broadly speaking, the one that started it is the one that we expect to be waiting for a reply, and not *vice versa*.

Now, the intuition is that the agent that started the negotiation process, let us say a, can ideally draw a tree of possible dialogues, that has as a root his initial move, let us say, $tell(a, b, request(\ldots), 0)$. A correct tree could be drawn if a knew b's program exactly, which is not an assumption we want to make. However, in first approximation, a can assume that b has the same constraints as she has. Then, a can generate a tree that has as nodes the possible dialogue moves, and as branches the integrity constraints that lead from one move to another one. An example of such tree, for the language $\mathcal{L}_1$ and the negotiation program defined in [13], is that of Figure 1.

The purpose of drawing a tree, that goes from an initial move to some possible final moves, is to have an ordering function, that allows to order the dialogue moves, and consider an ongoing dialogue *valid* as far as the tree or graph is explored in one direction (the one that leads to final moves). It is important to notice that not all agent programs in $\mathcal{P}$ allow us to draw a tree: in order to do that, such ordering function must exist. We call the existence of such function *acyclicity*, as we did in Section 3.2 in the case of IFF-terminating programs. If such function does not exist, it is not possible to adopt policy *(iii)*, and it is easy to imagine that a dialogue between two agents having both a non-acyclic program will not likely terminate. More formally, an instance of ordering function, that we call (*rank* function) is defined as follows:

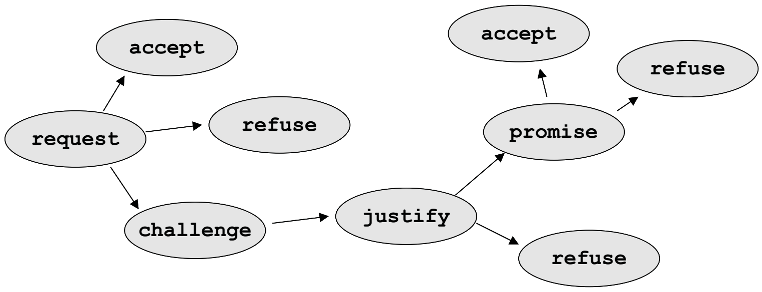

Figure 1. *Dialogue tree.*

Definition 8 (*Rank function*)
A *rank* function, mapping from a language $\mathcal{L}$ to the set of natural numbers $\mathbb{N}$, $rank : \mathcal{L} \to \mathbb{N}$, is *procedurally* defined for a given agent program $\mathcal{S} \in \mathcal{P}$ as follows in two steps. First we *label* all the performatives of $\mathcal{L}$, by applying one of the following rules, until no rule produces any change in the labeling:

- for all $p \in \mathcal{L}$ that have not been labeled yet, if $p \in initial(\mathcal{L})$, then $label(p) = 0$;
- for all $p \in \mathcal{L}$ that have not been labeled yet, if $p \notin initial(\mathcal{L})$ and $\exists ic \in \mathcal{S} : ic = p \wedge C \Rightarrow (\bar{p})$, and $label(\bar{p}) = r$, then $label(p) = r + 1$
- for all $p \in \mathcal{L}$ that have already been labeled, let $label(p)$ be r. Then, if $\exists ic \in \mathcal{S} : ic = (\bar{p}) \wedge C \Rightarrow p$ such that $label(\bar{p}) \leq r$, then $label(\bar{p}) = r + 1$

If it is not possible to apply such labeling to the language (i.e., if it is not possible to complete this procedure in finitely many steps), it means that the program is not acyclic. Otherwise, after labeling the language, let R be $\max_{p \in \mathcal{L}} label(p)$ (R is finite). Then, *rank* is defined as $rank(p) = R - label(p)$.

Once a rank function $rank(p, n)$, $n \in \mathbb{N}$ is defined for all $p \in \mathcal{L}$, the restriction turns out:

$$p^{(iii)}_{check}(T) = rank(p(T), R) \ \wedge \ q(T1) \ \wedge \ T1 < T \ \wedge \\ rank(q(T1), R1) \ \wedge \ R1 \leq R$$

That is, a move of higher rank has been made after a move of lower rank. It is worth to notice that the introduction of the restrictions does not modify the existing language ranking.

We will call this policy *check against an ordering*. It is more restrictive than the previous ones, since it does not allow jumping backwards from one branch to another one, and, again, some more possibly successful dialogues could be rejected. On the other hand, if applied to acceptable, exhaustive, and deterministic programs, it is enough to ensure termination, as stated by the following theorem:

Theorem 1 (*Finite termination of a dialogue with check against an ordering*) Let a and b be two agents provided with *acceptable*, *exhaustive*, and *deterministic* programs. Let a's program be *restricted* according to a *check against an ordering* policy. Therefore, a dialogue d between a and b *will terminate* before finitely many moves. In particular, if R is the maximum rank of a dialogue move, d will have *at most* $R+2$ moves.

Proof The proof for such theorem is given inductively. Given a dialogue $d = \{p_1, p_2, \ldots, p_j, \ldots\}$, between agents a and b, and started by a, let R be the maximum rank of a dialogue move in a's program. By definition of $rank$, all nodes ranked 0 must be leaves, and therefore final moves, while all the leaves ranked R are initial moves. Then we have, $\forall p_j$:

- if p_j is final, the dialogue is terminated;
- if p_j is not final, with $rank(p_j) = r_j > 0$, and it is uttered by b (i.e., j is even), then p_{j+1} must be computed by a, and will be either a final move, or will be ranked $rank(p_{j+1}) = r_j - 1$;
- if p_j is not final, with $rank(p_j) = r_j > 0$, and it is uttered by a (i.e., j is odd), then p_{j+1} must be computed by b, and it will be either a final move, or a move p_{j+1} ranked $rank(p_{j+1})$. If $rank(p_{j+1}) \geq rank(p_j)$, then the next move p_{j+2} will be final and the dialogue will terminate; otherwise p_{j+2} will be ranked $rank(p_{j+2}) = rank(p_{j+1}) - 1 \leq rank(p_j) - 2$;
- $rank(p_1) \leq R$ by definition.

Therefore, each move p_j is ranked r, with $r \leq R+1-j$. Now, if in the dialogue there are two moves with the same rank, the next move will be the last one. Since there cannot be more than two moves with the same rank, and the maximum rank is R, the maximum number of moves computed in a dialogue is $R+2$. For all non-final move p_j uttered by either agent, there exists a (unique) next move p_{j+1}, since both agent programs are exhaustive and deterministic. Moreover, the reasoning required by either party to compute a dialogue move terminates in a finite number of steps, since both programs are acceptable. Therefore, the dialogue terminates in at most $R+2$ moves. ∎

4.3 Termination of dialogue sequences

We can easily extend this termination result to the case of dialogue sequences, formally defined in [12], aimed at collecting all the (finitely many) missing resources, *missing($\mathcal{I}$)*, with respect to an intention $\mathcal{I}$, whose *cost* is defined as the cardinality of *missing($\mathcal{I}$)*. We do not have the space here to formally describe the dialogue sequence, as we did with dialogues; still we would like to give the intuition, and a sketchy proof of the second theorem that we are going to enunciate in the following. Dialogue sequences are defined such that an agent cannot ask the same resource twice to the same agent, within the same sequence. Since dialogues can modify the agents' intentions, a dialogue sequence $\{d_1, \ldots, d_n\}$ is associated with a series of intentions $\{\mathcal{I}_0, \ldots, \mathcal{I}_n\}$, where $\forall i, 1 \leq i \leq n$, $\mathcal{I}_i$ is the agent intention resulting from d_i, and $\mathcal{I}_0 = \mathcal{I}$ is the 'initial' intention.

Definition 9 (*Termination of a sequence of dialogues*)
A sequence of dialogues $s(\mathcal{I})$ with respect to an initial intention $\mathcal{I}_a$ of an agent a is *terminated* when, given that $\mathcal{I}_n$ is the intention after d_n, there exists no possible request dialogue with respect to $\mathcal{I}_n$ that a can start.

This could be due to two reasons: either after s there are no more missing resources in the intention, i.e. $cost(\mathcal{I}_n) = 0$, or $cost(\mathcal{I}_n) > 0$ and in a's program there is no constraint that can start a request.

In order to ensure termination, we could program the agent such that after every single dialogue the set of missing resources rather shrinks than grows in size: $\forall i > 0, cost(\mathcal{I}_i) > (\mathcal{I}_{i-1})$. If an agent program is such that a dialogue can only decrease the cost of an intention, we call such agent *self-interested rational.* In that case, given a system of $n+1$ agents and an intention $\mathcal{I}$, the length $length(s(\mathcal{I}))$ of a sequence $s(\mathcal{I})$ of dialogue with respect to an intention $\mathcal{I}$, i.e., the number of dialogues in $s(\mathcal{I})$, is bounded by the product $n \cdot cost(\mathcal{I})$. It is possible to prove that termination is a property that holds for *self-interested rational* agents whose programs are *restricted* according to a *check against an ordering* policy:
Theorem 2 (*Termination of a sequence of dialogues for a restricted agent program*)
Let A be a system of $n+1$ agents, and $s(\mathcal{I})$ be a sequence of dialogues with respect to an initial intention $\mathcal{I}$ of a *self-interested rational* agent $a \in A$. Let the agent programs be acceptable, exhaustive, deterministic, and in particular let a's program be *restricted* according to a *check against an ordering* policy. Then $s(\mathcal{I})$ *will terminate* before finitely many dialogue moves. In particular, if R is the maximum rank of a dialogue move, according to a's ranking function, $s(\mathcal{I})$ will terminate after at most $n \cdot cost(\mathcal{I}) \cdot (R+2)$ dialogue moves.

Proof (sketch) As the number of dialogues in $s(\mathcal{I})$ is bounded by the product $n \cdot cost(\mathcal{I})$, and each dialogue is terminated in at most $R+2$ moves, the whole sequence of dialogues will terminate, and will terminate after at most $n \cdot cost(\mathcal{I}) \cdot (R+2)$

dialogue moves.[5]

■

In the end, we would like to make a parallel between the concept of restriction introduced to ensure the termination of a (negotiation) dialogue and the self-interested rationality assumption of Theorem 2. Indeed, self-interested rationality could be considered a limitation, in that it reduces the space of agent programs. It reflects into a reduction of the space of the achievable solutions of a resource reallocation problem (it is easy to imagine situations where two negotiating agents get stuck in a 'local maximum' because none of them wants to give away a resource). There are some results in this respect in [14]. Due to such reduction, a *weak* notion of completeness has been introduced in [12].

5 Discussion

In this work, we have dealt with the problem of termination in dialogue-based agent negotiation. Building on an existing dialogue framework, where the course of dialogue is determined by rules and constraints embodied in the agents' programs, we introduced several syntactic transformation rules that can modify those programs towards a better robustness. Two results have been proven, determining an upper limit to the maximum length of a dialogue and of a sequence of dialogues, measured in terms of number of exchanged messages. Such results can be generalized, and reflect an existing trade-off between the need to ensure termination in the negotiation process and the loss in terms of reachable states in the universe of possible solutions to the problems addressed by negotiation.

References

[1] L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue and negotiation. In W. Horn, editor, *Proceedings 14th European Conference on Artificial Intelligence, Berlin, Germany.* IOS Press, August 2000.

[2] P. Davidsson. Categories of artificial societies. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Proc. ESAW'01, Praha, Czech Republic*, number 2203 in Lecture Notes in Computer Science. Springer-Verlag, December 2001.

[3] F. Dignum, B. Dunin-Keplicz, and R. Verbrugge. Dialogue in team formation.

[5] It could easily be proven that the bound is less than that, but it does not seem to make a significant difference, therefore we will consider as a reference this overestimating, still easy to understand and to use, upper bound.

In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, number 1916 in LNCS, pages 264–280. Springer-Verlag, 2000.

[4] Jürgen Dix, Fariba Sadri, and Ken Satoh, editors. *Computational Logic and Multi Agency*, volume to appear of *Special Issue of the Annals in Mathematics and Artificial Intelligence*. Baltzer Science Publishers, December 2001.

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

[6] A. Kakas and P. Mancarella. On the relation between Truth Maintenance and Abduction. In T. Fukumura, editor, *Proceedings of the first Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan*, pages 438–443, 1990.

[7] 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.

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

[9] S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation; a logical model and implementation. *Artificial Intelligence*, 104:1–69, 1998.

[10] S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

[11] S. Rochefort, F. Sadri, and F. Toni, editors. *Proc. International Workshop on Multi-Agent Systems in Logic Programming, in conjunction with ICLP'99, Las Cruces, New Mexico.* November 1999.

[12] 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 VII (best paper award)*. Springer Verlag, 2001.

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

[14] T. Sandholm. *Negotiation among Self-Interested Computationally Limited Agents.* Computer science, University of Massachusetts at Amherst, September 1996.

[15] D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning.* State University of New York Press, Albany, NY, 1995.

[16] M. J. Wooldridge and S. Parsons. Languages for negotiation. In W. Horn, editor, *Proceedings 14th European Conference on Artificial Intelligence, Berlin, Germany.* IOS Press, August 2000.