Argumentation and Cooperation for Distributed Extended Logic Programs*

Iara de Almeida Móra
CENTRIA
New University of Lisbon
2825 Monte de Caparica, Portugal
idm@di.fct.unl.pt

José Júlio Alferes
CENTRIA and Dept.Mathem.
University of Évora
Rua Romão Ramalho, 59
7000 Évora, Portugal
jja@dmat.uevora.pt

Michael Schroeder
Inst. for Knowledge-based Systems
University of Hannover
Lange Laude 3
30159 Hannover, Germany
schroede@kbs.uni-hannover.de

Abstract

Argumentation semantics in extended logic programming has been defined in (Dung 1993; Prakken & Sartor 1997) for a single agent, determining the agents' beliefs by an internal argumentation process. In this paper we extend the argumentation framework to a multi-agent setting including cooperation among agents. We present an inference algorithm for the framework and an example showing the process of argumentation and cooperation.

Keywords: Argumentation, Extended Logic Programming, Multi-agent systems

Introduction

The notion of autonomous agent is used to denote the fact that an agent has the ability to decide for itself which goals it should adopt and how these goals should be achieve. In most agents applications the autonomous components need to interact with one another given the inherent interdependencies which exist between then. The predominant mechanism for managing these interdependencies at run-time is negotiation.

Negotiation is a process that takes place between two or more agents which communicate to try and to come to a mutually acceptable agreement on some matter to achieve goals which they cannot, or prefer not to, achieve on their own. Their goals may conflict – in which case the agents have to bargain about which agent achieve which goal – or the agents may depend upon each other to achieve the goals.

It is our conviction that the use of a particular argumentation system provides the negotiation mechanism. Argument-based systems analyze defeasible, or non-monotonic reasoning in terms of the interactions between arguments to get an agreement for a goal. Non-monotonicity arises from the fact that arguments can be defeated by stronger (counter)arguments. The intuitive notions of argument, counterargument, attack and defeat have natural counterparts in the real world, which makes argument-based systems suitable for multi-agents applications. The ability to view extended logic programs as argumentation systems opens the way for the use of this language in formalizing communication among reasoning computing agents in a distributed framework.

Argumentation semantics in extended logic programming has been defined in (Dung 1993; 1995; Prakken & Sartor 1997; Móra & Alferes 1998b) for an agent, determining the agent's beliefs by an internal argumentation process. Intuitively, argumentation semantics treats the evaluation of a logic program as an argumentation process, where a goal G holds if all arguments supporting G can no longer be attacked. Thus, logic programming is seen as a discourse involving attacking and counter-attacking arguments.

While Dung (Dung 1993) uses argumentation to define a declarative semantics for extended logic programs, Prakken and Sartor's work (Prakken & Sartor 1997) is driven by their applications in legal reasoning. Logic programming can be described in terms of two figures: Reductio ad Absurdum- and ground-attack (Dung 1993) or, equivalently, rebut and undercut (Prakken & Sartor 1997). The former classifies an argument that leads to a contradiction under the current believes and arguments, and the latter an argument that falsifies the premise of one of the current arguments.

Next we introduce extended logic programming. After that, we present basic argumentation definitions and a proof proposal for an argument that coincides with the extended well founded semantics (WFSX), a semantics for extended logic programs(for details, see (Pereira & Alferes 1992)). Then we extend the argumentation framework for a multi-agent set-

*Thanks to Luis Moniz Pereira, Wolfgang Nejdl, Daniela Plewe, and Gerd Wagner. The work is financially supported by JNICT project ACROPOLE PBIC/TIT/2519/95, the European project BMFB/JNICT, and the Brazilian CAPES.

ting (Schroeder, Móra, & Alferes 1997) including cooperation. Finally, we present an algorithm for inference in the framework and an example showing the process of argumentation and cooperation.

Extended Logic Programming

Due to its declarative nature, as well as its procedural implementations, logic programming can be seen as a good language for knowledge representation. In fact, much work have been done concerning the use of logic programs for knowledge representation (see, e.g., (Alferes & Pereira 1996) and references therein), and concerning its relation to other well-known non-monotonic formalism for knowledge representation and reasoning, such as default logics, auto-epistemic logics, etc. In the sequel we will use a logic programming language to represent the agents' knowledge.

Normal logic programs use a non-monotonic form of "default negation"[1] whose major distinction from the classical negation is that it can be assumed in the absence of evidence to the contrary. Default negated literal are thus viewed as hypotheses which, under certain conditions, can be assumed.

Although default negation proved to be quite useful in various domains and application frameworks, it is not the only form of negation that is needed in non-monotonic formalisms. Indeed, while default negation *not* A of an atomic formula A is always assumed "by default", we often need to be more careful before jumping to negative conclusions. For example, it would make little sense to say $guilty(X) \leftarrow not\ innocent(X)$ to express the fact that being guilty is the opposite of being innocent because it would imply that people should be considered guilty "by default".

Moreover, in normal logic programs the negative information is implicit, i.e. it is not possible to explicitly state falsity – propositions are assumed false if there is no reason to believe they are true. Though this is what is wanted in some cases, having this single form of (implicit) negation is a serious limitation in other cases. In fact, in various situations one may want to explicitly declare that something is false.

These were the main motivations for the generalization of the language of logic programs to include an explicit form of negation (Gelfond & Lifschitz 1990). The so generalized language was called *Extended Logic Programming*.

Definition 1 (Extended Logic Program) *An* extended logic program *(ELP) is a (possibly infinite) set of ground rules*[2] *of the form*

$$L_0 \leftarrow L_1, \ldots, L_l, not\ L_{l+1}, \ldots, not\ L_m\ (0 \leq l \leq m)$$

where each L_i is an objective literal ($0 \leq i \leq m$). If $n = 0$ then the rule is called a fact *and the arrow symbol is omitted. An objective literal is either an atom A or its explicit negation $\neg A$. Literals of the form not L are called default literals. As usual L_0 is called the* head, *and $L_1, \ldots, not\ L_n$ the* body *of the rule.*

For this language, various declarative semantics have been defined, e.g. the answer-sets semantics (Gelfond & Lifschitz 1990) (which is a generalization of the stable models semantics of normal logic programs), the well-founded semantics with explicit negation (WFSX) (Pereira & Alferes 1992), and the well-founded semantics with "classical" negation (Przymusinski 1990). Both of the last two are generalizations of the well-founded semantics of normal programs. However, the former, unlike the latter, considers the so-called coherence requirement relating the two form of negation: *"if L is explicitly false then L must be assumed false by default"*. For further details on the subject of extended logic programs, their semantics, applications, and implementation see (Alferes & Pereira 1996).

Argumentation

The following definitions for argumentation are based on (Prakken & Sartor 1997). They propose an argument as a sequence of ground rules [3]:

Definition 2 (Argument) *Let P be an extended logic program. An* argument *for a conclusion L is a finite sequence $A = [r_n; \ldots; r_m]$ of rules $r_i \in P$ such that*

1. *for every i ($n \leq i \leq m$), and for every objective literal L_j in the body of r_i there is a $k < i$ such that L_j is the head of r_k.*
2. *L is the head of some rule of A.*
3. *No two distinct rules in the sequence have the same head.*

An argument A' (for some conclusion L') is a sub-argument *of the argument A (possibly for some other conclusion L) iff A' is a prefix of A.*

[1] Or "negation as failure" as it is also usually dubbed in the literature of logic programming.

[2] For simplicity, we use non-grounded rules in some examples. These rules simply stand for its ground version, i.e. for the ground rules obtained by substituting in all possible ways each of the variables by elements of the Herbrand Universe.

[3] In contrast to the latter we do not distinguish between strict and defensible rules. However, our results can be extended in this direction.

An argument attacks another argument via rebut or undercut. The difference depends on whether the attacking argument contradicts a conclusion, in the former, or an assumption of another argument, in the latter:

Definition 3 (Undercut, Rebut, Attack) *Let A_1 and A_2 be two arguments, then*

- A_1 undercuts A_2 *iff A_1 is an argument for L and A_2 is an argument with assumption not L, i.e. there is*

$$L_0 \leftarrow L_1, \ldots, L_l, not\ L_{l+1}, \ldots, not\ L_m \in A_2$$

  and there exists j $(l+1 \leq j \leq m)$ such that $L = L_j$.

- A_1 rebuts A_2 *iff A_1 is an argument for L and A_2 is an argument for $\neg L$.*

- A_1 attacks A_2 *iff A_1 undercuts or rebuts A_2.*

Not all arguments make sense. For example, it does not make sense to have an argument which assumes some hypothesis and, at the same time, concludes its negation. The notions of coherent argument and conflict-free set of arguments formalize what we mean by arguments and sets of arguments that "make sense":

Definition 4 (Coherent, Conflict-free)

- *An argument is* coherent *if it does not contain sub-arguments attacking each other.*

- *A set of arguments $Args$ is called* conflict-free *if no two arguments in $Args$ attack each other.*

Defeat of an argument can be direct, or indirect, by defeating one of its sub-arguments. In particular, any incoherent argument is defeated by the empty argument.

Definition 5 (Defeat, Strictly Defeat) *Let A_1 and A_2 be two arguments, then*

- A_1 defeats A_2 *iff A_1 is empty and A_2 incoherent; or A_1 undercuts A_2 or A_1 rebuts A_2 and A_2 does not undercut A_1.*

- A_1 strictly defeats A_2 *iff A_1 defeats A_2 but not vice versa.*

An argument A is acceptable if it can be defended against all attacks, i.e. all arguments that attack A can be counter attacked by *undercuting*.

Definition 6 (Acceptable) *An argument A is* acceptable *wrt. a set of arguments $Args$ iff each argument undercuting A is strictly defeated by an argument in $Args$.*

Note that our notion of acceptability differs from Prakken and Sartor's definition. In theirs, an argument A is acceptable wrt. $Args$ iff each *defeating* argument is strictly defeated by $Args$. In general, their proposal is more sceptical than ours:

Example 1 *Consider the program P :*

$$\begin{array}{rcl} \neg a & & \\ a & \leftarrow & not\ b \\ b & \leftarrow & not\ a \end{array}$$

and the set of arguments of P:

$$Args = \{[\neg a], [a \leftarrow not\ b], [b \leftarrow not\ a]\}$$

Note that arguments $[\neg a]$ and $[a \leftarrow not\ b]$ defeat each other. So, for Prakken and Sartor's definition there is no acceptable argument wrt. $Args$: e.g, $[\neg a]$ is not acceptable because $[a \leftarrow not\ b]$ defeats it and is not strictly defeated by $Args$.

This conclusion contradicts our intuition: since $\neg a$ is a fact, it should have some kind of "priority" over the rule $a \leftarrow not\ b$ whose premise is a default *assumption. Based on our proposal, $[\neg a]$ and $[b \leftarrow not\ a]$ are acceptable wrt. $Args$ whereas $[a \leftarrow not\ b]$ is not:*

- *$[\neg a]$ is acceptable because no argument undercuts it*

- *$[b \leftarrow not\ a]$ is acceptable because the only argument undercutting it ($[a \leftarrow not\ b]$) is defeated by $[\neg a]$*

- *$[a \leftarrow not\ b]$ is not acceptable because the only argument undercutting it ($[b \leftarrow not\ a]$) is not strictly defeated by any other argument in $Args$*

The fixpoint operator below captures the set of acceptable arguments:

Definition 7 (Characteristic Function) *Let P be an extended logic program and S be a subset of arguments of P. The* characteristic function *of P and S is*

$$F_P(S) = \{A \in S \mid A \text{ is acceptable wrt. } S\}$$

Proposition 1 *F_P is monotonic.*

The status of an argument is determined based on all ways in which arguments interact. It takes as input the set of all possible arguments and their mutual defeat relations, and produces as output a split of arguments into three classes: arguments with which a dispute can be 'won', 'lost', and arguments which leave the dispute undecided. The "weakest link" principle defines that an argument cannot be justified unless all of its sub-arguments are justified.

The intuitive idea is that the set of *justified* (those which "win") arguments is constructed step-by-step. We first collect into $JustArgs_1$ all arguments which are directly justified, by their own strength: those are the ones which are not defeated by any other counterargument. Then we add all arguments that are justified with the help of arguments in $JustArgs_1$. More exactly, (1) each argument that has all its subarguments justified in the previous step is justified, and (2) each argument that has attacking counterarguments, is added if all those counterarguments are defeated by an argument already in $JustArgs_1$. The resulting set is $JustArgs_2$. We repeat this step until we obtain a fixpoint, which is then the set of all justified arguments.

We define the losing, or "overruled" arguments as those that are attacked by a justified argument and, finally, the undeciding, or "defeasible" arguments as those that are neither justified nor overruled:

Definition 8 (Justified, Overruled, Defeasible) *Let P be an extended logic program, F_P be the* characteristic function *of P, and A be an argument then*

- *A is* justified *iff A is in the least fixpoint of F_P (called $JustArgs$)*
- *A is* overruled *iff A is attacked by a justified argument*
- *A is* defeasible *iff A is neither justified nor overruled*

Proposition 2 *F_P is finitary iff each argument in $JustArgs$ is attacked by at most a finite numbers of arguments in $JustArgs$.*

Definition 9 (Conclusion for a literal) *Let A be an argument for a literal L*

- *L is* justified *conclusion iff A is a justified argument.*
- *L is* overruled *conclusion iff L is not justified and A is an overruled argument.*
- *L is* defeasible *conclusion iff L is not justified nor overruled and A is a defeasible argument.*

Example 2 *Continuing the example 1, $\neg a$ and b are justified conclusions, and a is an overruled one.*

The result on this example coincides with the one obtained by WFSX with the Coherence Principle. And this is also the case in general:

Proposition 3 *Let P be a non-contradictory extended logic program and L be an objective literal then*

- *$L \in WFSX(P)$ (L is true in P) iff an argument for L is justified;*
- *not $L \in WFSX(P)$ (L is false in P) iff all arguments for L are overruled;*
- *$\{L, not\ L\} \cap WFSX(P) = \emptyset$ (L is undefined in P) iff all arguments for L are defensible*

Prakken and Sartor propose a proof for a justified argument for L as a dialogue tree where the root of the tree is an argument for a literal L, and each branch of the tree is a dialogue between a proponent P and an opponent O. A *move* in a dialogue consists of an argument attacking the last move of the other player. The required force of a move depends on who states it. Since the proponent wants a conclusion to be justified, a proponent's argument has to be strictly defeating while, since the opponents only want to prevent the conclusion from being justified, an opponent move may be just defeating.

The dialogue definition below is a modification of the original one (Prakken & Sartor 1997), so that it can capture our modified notion of acceptability, where attacks to an argument cannot be rebuts.

Definition 10 (Dialogue)
A dialogue *is a finite nonempty sequence of moves $move_i = (Player_i, A_i)(i > 0)$, such that*

1. *$Player_i = P$ iff i is* odd*; and $Player_i = O$ iff i is* even
2. *If $Player_i = Player_j = P$ and $i \neq j$, then $A_i \neq A_j$*
3. *If $Player_i = P (i > 1)$, then A_i is a minimal (wrt. set inclusion) argument strictly defeating A_{i-1}*
4. *If $Player_i = O$, then A_i undercuts A_{i-1}*

The first condition says that P begins and the players take turns. The second condition prevents the proponent from repeating its attacks. The remaining two conditions form the heart of the definition: they state the burdens of proof for P and O. The minimally condition on P's move makes it impossible to make arguments trivially different by combining them with some other, irrelevant argument.

Finally, a dialogue tree considers all possible ways in which an opponent can attack an argument:

Definition 11 (Dialogue Tree) *A* dialogue tree *is a finite tree of moves $move_i = (Player_i, A_i)$, $Player_i \in \{P, O\}$ such that*

1. *Each branch is a dialogue*
2. *If $Player_i = P$ then the children of $move_i$ are all defeaters of A_i*

A player wins a dialogue tree iff it wins all branches (i.e. dialogues) of the tree and a player wins a dialogue if the other player cannot move (i.e. counter-argue).

Proposition 4 *Let A be an argument for a literal L.*

- *L is justified iff the proponent wins the dialogue tree starting with A.*
- *L is overruled iff at least one dialogue, in the dialogue tree for A, is defeated by an opponent's justified argument A'.*
- *L is defeasible iff A is not overruled neither justified.*

Multi-Agent Argumentation

To extend the single-agent to a multi-agent approach we start by defining multi-agent systems as sets of several extended logic programs, each one corresponding to an individual agent with an individual view of the world.

Definition 12 (Multi-agent System) *Let Ag_i be extended logic programs, where $1 \leq i \leq n$. Then the set $\mathcal{A} = \{Ag_1, \ldots, Ag_n\}$ is called a* multi-agent system *(MAS).*

We want to exchange parts of several independent or overlapping programs and to obtain a consensus among the programs concerning inference of literals. Our multi-agent semantics is closely related to the single agent one. In particular we want the behavior of a MAS with only one agent to coincide with the single agent approach. But the semantics of the MAS, when more than one agent is involved, might not necessary coincide with the semantics of the union of all agents:

Example 3 *Let $\mathcal{A} = \{Ag_1, Ag_2\}$ be a MAS such that*

$$Ag_1 = \{a \leftarrow not\ b\} \text{ and } Ag_2 = \{b\}$$

and the set of arguments of Ag_1 and Ag_2 are:

$$Args_1 = \{[a \leftarrow not\ b]\} \text{ and } Args_2 = \{[b]\}$$

Then Ag_1 concludes $\{not\ b\}$ wrt. $Args_1$ and does not conclude $\{a\}$ wrt. $Args$; Ag_2 concludes $\{not\ a\}$ wrt. $Args_2$ and $\{b\}$ wrt. $Args$; and $\mathcal{A}$ concludes $\{b,\ not\ a\}$ wrt. $Args$. At first sight it seems puzzling that Ag_1 does not assume a as true. The reason is that concluding a requires Ag_2 to agree on not b which is obviously not the case.

Definition 13 (Successfully Argumentation) *Let $\mathcal{A}$ be a MAS, Arg_i be a set of arguments of $Ag_i \in \mathcal{A}$, $A \in Arg_i$ is an argument for a literal L, and*

$$Args = \bigcup_i (Arg_i).$$

- *Agent Ag_i* successfully argues *wrt. L iff A is* justified *wrt. $Args$.*
- *L is* overruled *iff A is overruled wrt. $Args$;*
- *otherwise, L is concluded as* defeasible *wrt. $Args$.*

Since previous approaches did not tackle multi-agent argumentation there was no need for cooperation. With two or more agents involved in the process it may be the case that an agent needs the support of another in order to counter-argue.

Example 4 *Consider a trial with a judge, a defender, a prosecutor and a witness. The judge has no knowledge, while the defender knows that the accused is not guilty by default:*

$$\neg guilty \leftarrow not\ guilty$$

The prosecutor knows that the accused is guilty if there is evidence:

$$guilty \leftarrow seen$$

Finally the witness saw the accused committing the crime and therefore knows

$$seen \leftarrow$$

Then $\mathcal{A}$ is a MAS that models the trial knowledge, the agents being judge, defender, prosecutor and witness:

$$\{\emptyset, \{\neg guilty \leftarrow not\ guilty\}, \{guilty \leftarrow seen\}, \{seen\}\}$$

The defender has an argument for $\neg guilty$:

$$A_1 = [\neg guilty \leftarrow not\ guilty]$$

and the witness has

$$A_2 = [seen]$$

for seen. The prosecutor cannot support guilty because it has no knowledge about seen. If the judge asks the defender to defend the accused, the prosecutor needs the help of a witness to defeat the arguments of the defender.

As shown by the example above, the prosecutor cannot continue to argue because it has just a part of an argument, i.e. a partial argument:

Definition 14 (Partial Argument) *Let Ag be an extended logical program. A* partial argument *for a conclusion L is a finite sequence $PA = [r_n; \ldots; r_m]$ of rules $r_i \in Ag$ such that*

1. *for every i $(n < i \leq m)$, and for every objective literal L_j in the body of r_i there is a $k < i$ such that L_j is the head of r_k.*

2. *L is the head of some rule of A*

3. *No two distinct rules in the sequence have the same head.*

Note that, contrary to the definition of argument, an objective literal L_j in the body of r_n need not necessarily have a rule with head L in the argument. Intuitively this simply means that the argument is incomplete with respect to that L_j.

Example 5 *Continuing the example 4, the prosecutor has a partial argument for guilty:*

$$PA = [guilty \leftarrow seen]$$

This partial argument is incomplete wrt. seen.

To continue an argumentation process, a partial argument is not enough. If an agent has only a partial argument then it should be completed by a multi-agent cooperation:

Definition 15 (Successfully cooperation) *Let $\mathcal{A}$ be a MAS. Agent $Ag_i \in \mathcal{A}$ successfully cooperates with a set of agents $Ags \subseteq \mathcal{A}$ wrt. a partial argument PA (for some literal L) of Ag_i iff for all objective literals $L_k \in PA$ there is an agent $Ag_j \in Ags$ such that Ag_j has an argument for L_k.*

To obtain the results of multi-agent cooperation we define cooperative dialogue. Intuitively, in cooperative dialogues other agents provide arguments for the still incomplete objective literals. The process stop either if no agents can provide the needed arguments or if there are no more incomplete literals:

Definition 16 (Cooperative Dialogue) *Let $\mathcal{A}$ be a MAS, Ag be an agent in $\mathcal{A}$, $A = [r_n; \ldots; r_m]$ be a partial argument (for some literal L) of Ag, and Inc be the set of objective literals L_j in the body of r_n.*

A cooperative dialogue *for A in Ag is a finite nonempty sequence of moves, $move_i^c = (Ag_k, A_i)$ $(i > 0)$ such that*

1. *$move_1^c = (Ag, A)$.*

2. *if $i \neq j$ then $A_i \neq A_j$.*

3. *Let $move_i^c = (Ag_i, [r_n; \ldots; r_m])$ then if there exists an $Ag_k \in \mathcal{A}$ with an argument for some $L \in Inc$, $move_{i+1}^c = (Ag_k, [r_{n_k}; \ldots; r_{m_k}; r_n; \ldots; r_m])$; otherwise, $move_i^c$ is the last move in the dialogue.*

A cooperative dialogue of an agent Ag for a partial argument A is successful if its last move (Ag_f, A_f) is a (complete) argument. Then we call A_f its result.

The first condition says that an agent Ag starts a cooperation process for a partial argument A. The second prevents agents from repeating arguments. The third defines that the next movement is an argument for some incomplete literal L in the previous partial argument. The process stops either if there is no argument for such L, meaning that the cooperation failed, or if there is no more incomplete literal, meaning that the partial argument is completed.

Proposition 5 *Let $\mathcal{A}$ be a MAS, and Ag_k be an agent in $\mathcal{A}$. A cooperation for a partial argument PA (for some L) of Ag_k is successful if a cooperative dialogue for PA is successful.*

Finally, we extended the dialogue process of Def. 10 to include cooperation and argumentation between various agents.

Definition 17 (Extended Dialogue) *Let $\mathcal{A}$ be a MAS and Ag be an agent in $\mathcal{A}$. An* extended dialogue *for an argument A of an agent Ag is a finite nonempty sequence of moves $move_i^a = (Role_i, Ag_i, A_i)$ $(i > 0)$, where $Role_i \in \{P, O\}$ and $Ag_i \in \mathcal{A}$, such that*

1. *$move_1^a = (P, Ag, A)$.*

2. *$Role_i = P$ iff i is* odd*; and $Role_i = O$ iff i is* even

3. *If $Role_i = Role_j = P$ and $i \neq j$, then $A_i \neq A_j$*

4. *A_i is either an argument of Ag_i or the* result *of a successful cooperation dialogue for a partial argument of Ag_i.*

5. *If $Role_i = P$ $(i > 1)$ then A_i is a minimal (wrt. set inclusion) argument strictly defeating A_{i-1}*

6. *If $Role_i = O$ then A_i undercuts A_{i-1}*

The first condition says that an agent Ag starts an argumentation process for an argument A as a proponent. The second one says that the agent's argument can either be used to attack the argument in the first move (in which case $Role = O$), i.e. the agent is an opponent; or to counterattack (in which case $Role = P$), i.e. the agent is a proponent. The third prevents the proponent agents from repeating arguments. The fourth states that a dialogue just continues with arguments of individual agents, or the results of successful cooperations for a partial arguments of individual agents. The remaining two conditions state the burdens of proof for agents having the role P and O. The minimally condition on P's move makes it impossible to make arguments trivially different by combining them with some other, irrelevant, argument.

Finally, an extended dialogue tree considers all possible ways in which an agent can attack an argument:

Definition 18 (Extended Dialogue Tree) *Let $\mathcal{A}$ be a MAS, Ag be an agent in $\mathcal{A}$ and A be an argument of Ag. An* extended dialogue tree *for A is a finite tree of moves $move_i^a = (Role, Ag_k, A_i)$, such that*

1. *Each branch is an* extended dialogue *for A*
2. *If $Role = P$ then the children of $move_i^a$ are all the defeaters of an argument A_i*

An agent wins an extended dialogue tree iff it wins all branches (i.e. dialogues) of the tree and an agent wins a dialogue if another agent cannot move (i.e. counter-argue).

Proposition 6 *Let Ag and Ag_k be agents in $\mathcal{A}$, and A be an argument for a literal L of Ag.*

- *L is justified iff the Ag wins the dialogue tree starting with A.*
- *L is overruled iff at least one dialogue, in the dialogue tree for A, is defeated by an opponent Ag_k's justified argument.*
- *L is defeasible iff A is not overruled neither justified.*

In the remainder of the paper we only use *extended dialogue tree* (resp. *extended dialogue*). So, for simplicity, we refer to them simply as *dialogue tree* (resp. *dialogue*).

Argumentation Protocol

To implement the MAS's inference operation, the agents evolve an argumentation process. In the light of speech act theory and subsequent agent communication languages such as ACL, KIF, or KQML (Finin *et al.* 1993) we can identify five relevant speech acts of the dialogues. By assuming that L (resp. L') is an objective literal and A (resp. A') is an argument that supports L (resp. L') then

1. An agent requests inference of a literal L by sending a token $ask(L)$ and
2. If the agent wins the argumentation with argument A, it'll reply to a request by $reply(L, A)$; otherwise, it'll reply $reply(not\ L, none)$.
3. An agent proposes a conclusion L to other involved agents by sending the token $propose(L, A)$.
4. An agent opposes an argument by sending an $oppose(L', A', L, A)$, where $L' \leftarrow A'$ is an argument attacking the previously received $L \leftarrow A$. If the attack is by rebut then $L' = \neg L$; if it is by undercut then $L' = L''$ and $A = [\ldots, not\ L'', \ldots]$.
5. When the received argument A is acceptable the agent acknowledges this by sending $agree(L, A)$.

Using the above speech acts we define an argumentation protocol. We assume that a proposing agent sends its conclusion to all involved agents, but the opposition against a conclusion occurs only between the proponent and the opposer. In the following we sketch an algorithm for multi-agents argumentation and cooperation:

Algorithm:

- Proponent and Opponent, Cooperation:
  1. If the agent has just a partial argument for L (cf.Def.14) then *ask* all involved agents, i.e. agents that are able to cooperate (cf. Def.15) to complete the partial argument.
  2. On receipt of a *query*, from an agent that is not external[4], to infer L try to built an argument A for L, by itself or by cooperation, and reply it. If there is no such A then reply negatively.
  3. On receipt of a *reply*. If it is a positive answer then update the cooperation process as a cooperative dialogue (cf.Def.16) and continue to argue; otherwise, give up to built the previous counter-argument and try to built other argument (see argumentation).
- Proponent, Argumentation:
  1. On receipt of a *query* from an external agent to infer L try to built an argument, by itself or by asking other agents to complete a partial argument PA (see cooperation). If there is one argument A then create a dialogue tree (cf.Def.18) for A and propose it to all involved agents, i.e. agents that are able to argue; otherwise, reply that L is a false conclusion.
  2. On receipt of an *agreement*, check the state of the dialogue tree: if some dialogue (cf.Def.10) has not finished yet then wait for other answers, else check dialogue tree's leaves: if all leaves received are wining leaves then reply the agreement positively; otherwise, reply negatively.
  3. On receipt of an *oppose*, put the counter-argument CA on the involved dialogue (cf.Def.10). Try to built a counter-argument CCA as a proponent, itself or by asking other agents to complete a partial argument PA (see cooperation): if there is such CCA then put it on the involved dialogue (cf.Def.10) and propose CCA to all involved

[4] An external agent is an agent not belonging to the MAS. In our trial example, the judge is an external agent.

agents. If there is no such CCA then reply the initial request negatively.

- Opponent, Argumentation:

  1. On receipt of a *proposal*, try to built a counter-argument CA as an opponent, itself or by asking for cooperation to complete a partial argument PA, and then oppose. If there is no such CA then agree with proposal.

Finally, we present an example to show the process of argumentation and cooperation. We assume a trial with a judge, a defensor, a prosecutor, a defense witness and a prosecution witness. The judge, the prosecutor and the defensor have the same knowledge as shown in example 4.

If the judge asks the prosecutor to prove that the accused is guilty, the prosecutor needs the help of a witness. The prosecution witness saw the accused committing the crime and therefore knows seen. As the defensor cannot rebut this testimony, then it agrees and the prosecutor wins the argumentation dialogue.

On the other hand, if the judge asks the defender to prove that the accused is not guilty, the defender needs the help of a witness to destroy the prosecutor testimony. The defense witness knows that the prosecution witness needs to use glasses, and that if he is not using them then he cannot see. The accusation wins because the prosecution witness affirms he was using the glasses and the defense cannot rebut this argument.

As shown in the example, it is not necessary for an agent to cooperate and argue with all other agents in the multi-agent system. Accordingly, in the implementation, we restrict argumentation and cooperation of an agent to a predefined (provided by the user) set of other agents. These predefined set are declared in the program of an agent Ag_i with predicates:

$$argue\_with(ListOfAgentsA)$$

$$cooperate\_with(ListOfAgentsC)$$

meaning that Ag_i can only argue (resp. ask by cooperation) with the agents in $ListOfAgentsA$ (resp. $ListOfAgentsC$). Then

$$\mathcal{A} = \{defender, prosecutor, witness_d, witness_p\}$$

is a MAS that models the trial, such that each agent has the following knowledge:

$$\begin{aligned} defender \;=\; & \{argue\_with([prosecuter]);\\ & cooperate\_with([witness_d]);\\ & \neg guilty \leftarrow not\ guilty\} \end{aligned}$$

$judge$	$\longrightarrow$	$prosecutor$	$ask(g)$
$prosecutor$			$cooperation\ for\ PA_1$
$prosecutor$	$\longrightarrow$	$witness_p$	$ask(s)$
$prosecutor$	$\longleftarrow$	$witness_p$	$reply(s, A_3)$
$prosecutor$			$A_5 = A_3 + PA_1$
$prosecutor$	$\longrightarrow$	$defender$	$propose(g, A_5)$
$prosecutor$	$\longleftarrow$	$defender$	$agree(g, A_5)$
$prosecutor$			$checks\ dialogue$
$judge$	$\longleftarrow$	$prosecutor$	$reply(g, A_5)$

Figure 1: Trace of the consistent trial example.

$$\begin{aligned} prosecutor \;=\; & \{argue\_with([defender]);\\ & cooperate\_with([witness_p]);\\ & guilty \leftarrow seen\} \end{aligned}$$

$$\begin{aligned} witness_p \;=\; & \{argue\_with([]);\\ & cooperate\_wiyth([]);\\ & using\_glasses;\ seen\} \end{aligned}$$

$$\begin{aligned} witness_d \;=\; & \{argue\_with([]);\\ & cooperate\_with([]);\\ & need\_glasses;\\ & \neg seen \leftarrow need\_glasses,\\ & not\ using\_glasses\} \end{aligned}$$

and the arguments, with obvious abbreviations, are:

$$\begin{aligned} A_1 &= [\neg g \leftarrow not\ g]\\ A_2 &= [n\_g; \neg s \leftarrow n\_g, not\ u\_g]\\ A_3 &= [s]\\ A_4 &= [u\_g]\\ PA_1 &= [g \leftarrow s] \end{aligned}$$

Note that the judge can conclude the accused is guilty in both cases, the argumentation dialogues is shown in the figures 1 and 2. This is so because the testimonies of the prosecution witness cannot be defeated. Nevertheless if the prosecutor witness could not remember that he was using glasses and the testimonies of both witnesses could not be defeated and the judge would then be able to conclude both *guilty* and $\neg guilty$ (i.e. a consensus could not be reached). To solve this problem, priorities should be used in order to always obtain a consensual result.

Conclusion and Further work

The work presented in this paper is based on work by Dung (Dung 1993; 1995) and Prakken and Sartor (Prakken & Sartor 1997) on argumentation. Dung defines a declarative semantics for extended logic programs using the metaphor of argumentation. Our work continues this line of research in that we extend the single-agent approach to a multi-agent one.

$judge$	$\longrightarrow$	$defender$	$ask(\neg g)$
$defender$	$\longrightarrow$	$prosecutor$	$propose(\neg g, A_1)$
$prosecutor$			$cooperation\ for\ PA_1$
$prosecutor$	$\longrightarrow$	$witness_p$	$ask(s)$
$prosecutor$	$\longleftarrow$	$witness_p$	$reply(s, A_3)$
$prosecutor$			$A_5 = A_3 + PA_1$
$defender$	$\longleftarrow$	$prosecutor$	$oppose(g, A_5, \neg g, A_1)$
$defender$			$cooperation\ to prove \neg s$
$defender$	$\longrightarrow$	$witness_d$	$ask(\neg s)$
$defender$	$\longleftarrow$	$witness_d$	$reply(\neg s, A_2)$
$defender$			$A_6 = A_2 + \emptyset$
$defender$	$\longrightarrow$	$prosecutor$	$propose(\neg s, A_6)$
$prosecutor$			$cooperation\ to\ prove\ u\_g$
$prosecutor$	$\longrightarrow$	$witness_p$	$ask(u\_g)$
$prosecutor$	$\longleftarrow$	$witness_p$	$reply(u\_g, A_4)$
$prosecutor$			$A_7 = A_4 + \emptyset$
$defender$	$\longleftarrow$	$prosecutor$	$oppose(u\_g, A_7, \neg s, A_6)$
$defender$			$cooperation\ to\ prove \neg u\_g$
$defender$	$\longrightarrow$	$witness_d$	$ask(\neg u\_g)$
$witness_d$			$cooperation\ to\ prove \neg u\_g$
$defender$	$\longleftarrow$	$witness_d$	$reply(not \neg u\_g, none)$
$judge$	$\longleftarrow$	$defender$	$reply(not \neg g, none)$

Figure 2: Trace of the consistent trial example.

Prakken and Sartor are motivated by legal reasoning and define a rigorous and concise framework similar to Dung's, and they also deal only with a single agent and therefore do not tackle the issue of cooperation. The Prakken and Sartor's argumentation semantics defines a set of priorities to solve conflicts on the argumentation dialogue (for details see (Prakken & Sartor 1997)) and it can be easily added to our argumentation proposal. When there is no determined priority, literals involved in some conflict are simple considered as defeasible.

Since our argumentation semantic incorporates the Coherence principle it is more credulous and leads to more intuitive results than their proposal.

Our previous work on argumentation (Schroeder, Móra, & Alferes 1997) has defined multi-agent argumentation. However it proposed a simplified cooperation process: a partial argument is defined as a rule. We extend the definition of partial argument for also incomplete sequence of rules. We improve the previous algorithm to have cooperation and simulate it for a multi-agent inference. We show the connection between *WFSX* semantics and argumentation.

Finally, Prakken and Sartor (Prakken & Sartor 1997) and Dung (Dung 1995) propose different methods to get sceptical or credulous argumentation. As further work, we intend to define just one (flexible) argumentation framework to obtain conclusions in more credulous or sceptical way[5], depending on the application's goal.

References

Alferes, J. J., and Pereira, L. M. 1996. *Reasoning with Logic Programming.* (LNAI 1111), Springer-Verlag.

Alferes, J. J.; Damásio, C. V.; and Pereira, L. M. 1995. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning* 14(1):93–147.

Dung, P. M. 1993. An argumentation semantics for logic programming with explicit negation. In *Proc. of the 10th International Conference on Logic Programming*, 616–630. MIT Press.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77(2):321–357.

Finin, T.; Weber, J.; Wiederhold, G.; Genesereth, M.; Fritzson, R.; McKay, D.; McGuire, J.; Pelavin, R.; Shapiro, S.; and Beck, C. 1993. Specification of the KQML agent communication language. Technical report, The DARPA Knowledge Sharing Initiative, External Interfaces Working Group, Baltimore, USA.

Gelder, A. V.; Ross, K.; and Schlipf, J. S. 1988. Unfounded sets and well-founded semantics for general logic programs. In *Proceeding of the 7th ACM Symposium on Principles of Databse Systems.* Austin, Texas. 221–230.

Gelfond, M., and Lifschitz, V. 1990. Logic programs with classical negation. In Warren, and Szeredi., eds., *7th Int. Conf. on LP*, 579–597. MIT Press.

Móra, I. A., and Alferes, J. J. 1998a. Argumentative and cooperative multi-agent system for extended logic programs. In *XIVth Brazilian Symposium on Artificial Intelligence.* Springer-Verlag. Submitted.

Móra, I. A., and Alferes, J. J. 1998b. Credulus and sceptical argumentation for extended logic programs. In *Internal Report.*

Pereira, L. M., and Alferes, J. J. 1992. Well founded semantics for logic programs with explicit negation. In *B. Neumann (Ed.), European Conference on Artificial Intelligence.* John Wiley & Sons. 102–106.

Pereira, L. M.; Aparício, J. N.; and Alferes, J. J. 1993. Non–monotonic reasoning with logic programming. *Journal of Logic Programming. Special issue on Nonmonotonic reasoning* 17(2, 3 & 4).

[5]Some results can be found in (Móra & Alferes 1998b; 1998a).

Prakken, H., and Sartor, G. 1997. Argument-based extended logig programming with defeasible priorities. *Journal of Applied Non-Classical Logics*.

Przymusinski, T. 1990. Extended stable semantics for normal and disjunctive programs. In Warren, and Szeredi., eds., *7th Int. Conf. on LP*, 459–477. MIT Press.

Schroeder, M.; Móra, I. A.; and Alferes, J. J. 1997. Vivid agents arguing about distributed extended logic programs. In Costa, E., and Cardoso, A., eds., *Progress in Artificial Intelligence, 8th Portuguese Conference on Artificial Intelligence EPIA97*, volume LNAI1323. Coimbra, Portugal: Springer–Verlag. 217–228.