Argumentation for Distributed Extended Logic Programs *

Iara de Almeida Móra†, José Júlio Alferes‡and Michael Schroeder§

Abstract

Argumentation semantics in extended logic programming has been defined in [5, 12] for a single agent which determines its believes by an internal argumentation process. In this paper we extend the argumentation framework for a multi-agent setting [13] including cooperation. We present an algorithm for inference in the framework and line out, by an example how, it is implemented using vivid agents [16, 14].

1 Introduction

In the last 5-10 years researchers have been devoting much work to the semantics of logic programs. Among the different approaches, that usually either emphasise an operational or a declarative view, argumentation semantics turned out to be a very intuitive approach. Rather than defining a semantics in technical terms, argumentation semantics uses the metaphor of argumentation as used in politics, law, discourse, etc, and formalizes a part of it sufficient to give a meaning to extended logic programs. Intuitively, argumentation semantics treats the evaluation of a logic program as an argumentation process, where a goal G holds if all arguments supporting G cannot be attacked "anymore". Thus, logic programming can be seen as a discourse involving attacking and counter-attacking arguments.

While argumentation in rhetorics comprises a variety of figures, logic programming can be described in terms of two figures: Reductio ad absurdum- and ground-attack [6] or, equivalently, rebut and undercut [12]. 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. Argumentation semantics in extended logic programming

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

†CENTRIA, U. Nova de Lisboa 2825 Monte da Caparica, Portugal

‡CENTRIA and D.M., Univ. Évora, Largo dos Colegiais, 7000 Évora, Portugal

§Institut für Rechnergestützte Wissensverarbeitung,Lange Laube 3, D-30159 Hannover, Germany

has been defined in [5, 12] for a single agent which determines its believes by an internal argumentation process. In this paper we extend the argumentation framework for a multi-agent setting [13] including cooperation. We present an algorithm for inference in the framework and line out, by an example, how it is implemented using vivid agents [16, 14].

2 Basic Definitions

Since Prolog became a standard in logic programming much research has been devoted to the semantics of logic programs. In particular, Prolog's unsatisfactory treatment of negation as finite failure led to many innovations. Well-founded semantics [8] turned out to be a promising approach to cope with negation by default. Subsequent work extended well-founded semantics with a form of explicit negation [11, 4], defining *WFSX* , and showed that the richer language is appropriate for a spate of knowledge representation and reasoning forms [9, 10, 3].

Definition 2.1 *An* extended logic program *is a (possibly infinite) set of ground rules 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 ommited. An objective literal is either an atom* A *or its explicit negation* $\neg A$*. Literals of the form not* L *are called default literals.*

WFSX extends *WFS* for normal programs, and relates both negations in extended logic program by the so called *Coherence Principle*: if $\neg L$ is true, and since it is an explicit declaration of the falsity of L, then L should be false by default, i.e. *not* L should also be true.

The following definitions for argumentation are based on [5, 12]. In contrast to the latter we do not distinguish between strict and defeasible rules. However, our results can be extended in that direction.

Definition 2.2 *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* $n \leq i \leq m$*, and 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* subargument *of the argument* A *(possibly for some other conclusion* L*) iff* A' *is a subset of* A*.*

Definition 2.3 *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 an* $r : L_0 \leftarrow L_1, \ldots, L_l, not\ L_{l+1}, \ldots, not\ L_m \in A_2$ *and a* j *such that* $l + 1 \leq j \leq m$ *and* $L = L_j$*;* 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*.*

Definition 2.4 *An argument is* coherent *if it does not contain subarguments attacking each other. A set Args of arguments is called* conflict-free *if no two arguments in Args attack each other.*

Definition 2.5 *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; A_1 is* acceptable *wrt. a set Args of arguments iff each argument defeating A_1 is strictly defeated by an argument in Args.*

Definition 2.6 *Let P be an extended logic program and S a subset of arguments of P. The* characteristic function *of P and S is $F_P(S) = \{A \in S \mid A$ is acceptable wrt. $S\}$. A is* justified *iff A is in the least fixpoint of F_P. A is* overruled *iff A is attacked by a justified argument. A is* defensible *iff A is neither justified nor overruled.*

Example 2.1 *Let P be an extended logic program $\{hasPorsche; rich \leftarrow hasPorsche; \neg rich \leftarrow not\ hasPorsche\}$. The only argument for the conclusion rich is $A_1 = [hasPorsche; rich \leftarrow hasPorsche]$, and for $\neg rich$ is $A_2 = [\neg rich \leftarrow not\ hasPorsche]$. Note that we also have $A_3 = [hasPorsche]$, a subargument of A_1 for the conclusion hasPorsche.*

A_1 rebuts A_2 and vice versa. As A_3 strictly defeats A_2 (by undercutting), A_3, and so also A_1, are acceptable arguments, and A_2 is an overruled argument.

For any literal L, L is a *justified conclusion* iff it is a conclusion of a justified argument; a *defensible conclusion* iff it is not justified and it is a conclusion of some defensible argument; and an *overruled conclusion* iff it is not justified or defensible, and a conclusion of an overruled argument.

3 Argumentation and WFSX inference

To relate argumentation and extended logic programming inferences, we review *WFSX* [4], a semantics for extended logic programs. For similarity with Prakkens's argumentation, instead of the declarative (bottom-up) original definition of *WFSX* , we present an equivalent top-down inference operator for *WFSX* . The equivalence between the definitions is shown in [3].

The inference operator has three parameters M, LA, and GA, where M is either t or tu indicating that we want to prove verity (t) and non-falsity (tu), and LA and GA are lists of local and global ancestors that allow to detect negative and positive loops which lead to inference of non-falsity and failure, respectively (for details see[1, 2, 4]):

Definition 3.1 *Let P be an extended logic program and Body a sequence of*

literals $L_1, \ldots, L_l, not\ L_{l+1}, \ldots, not\ L_m$, *then*

$$
\begin{array}{rcl}
P \vdash L & \textit{iff} & P, \emptyset, \emptyset, t \vdash L \\
P, M \vdash L & \textit{iff} & P, \emptyset, \emptyset, M \vdash L \\
P, LA, GA, M \vdash \textit{true} & & \\
P, LA, GA, M \vdash (L_1, L_2) & \textit{iff} & P, LA, GA, M \vdash L_1 \ \&\ P, LA, GA, M \vdash L_2 \\
P, LA, GA, M \vdash not\ L & \textit{iff} & M = t \ \&\ P, \emptyset, GA, tu \not\vdash L \ or \\
 & & M = tu \ \&\ P, GA, GA, t \not\vdash L \\
P, LA, GA, t \vdash L & \textit{iff} & L \notin LA \ \&\ L \leftarrow Body \in P \ \& \\
 & & P, LA \cup \{L\}, GA \cup \{L\}, t \vdash Body \\
P, LA, GA, tu \vdash L & \textit{iff} & L \notin LA \ \&\ L \leftarrow Body \in P \ \& \\
 & & P, LA \cup \{L\}, GA \cup \{L\}, tu \vdash Body, not\ \neg L
\end{array}
$$

This inference operator is similar to the inference operator for *WFS* of normal programs of [3], the only difference being in the last line above. There, to prove non-falsity of a literal L, the non-falsity of $not\ \neg L$ must be proven (i.e. verity of $\neg L$ must not be proven). This implements the Coherence principle: if $\neg L$ is true then non-falsity of $not \neg L$ is not proven and, by the last requirement, non-falsity of L is not proven also. Thus L is false, i.e. $not\ L$ is true, as desired by the Coherence Principle.

Prakken and Sartor propose a proof for a justified argument as a dialogue tree where each branch of the tree is a dialogue between a proponent P and an opponent O, and the root of the tree is an argument for L. A *move* in a dialogue consists of an argument attacking the last move of the other player. The required strength 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. The opponent simply wants to prevent the conclusion from being justified. Thus there is no need for its move to be strictly defeating: it is enough for it to be defeating.

As shown in [13] this proposal is more skeptical than *WFSX* inference because it does not assume a fact L as a justified argument even when $[\neg L \leftarrow Body]$ is overruled by some other argument. The dialogue definition below is a modification of Prakken and Sartor's original one, for it to concide with *WFSX* (for details see [13]).

Definition 3.2 *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}.

Definition 3.3 *A* dialogue tree *is a finite tree of moves such that*

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

Example 3.1 *Let $P = \{\neg a; a \leftarrow not\ b; b \leftarrow not\ a\}$ be an extended logic program. The arguments for this program are $A_1 = [\neg a]$, $A_2 = [b \leftarrow not\ a]$ and $A_3 = [a \leftarrow not\ b]$. Assuming the proponent starts a dialogue by proposing A_2 to prove b, the opponent proposes A_3 to undercut A_2. Then the proponent's argument A_1 strictly defeat A_3 and b is concluded as justified. As we see on the dialogue above, a is an overruled conclusion because A_3 is strictly defeated by A_2. $\neg a$ is a justified conclusion because the opponent cannot counter-argue the argument A_1.*

Proposition 3.1 *Relation of Argumentation and $\vdash$*
Let P be an extended logic program and L an objective literal then

- *$L \in WFSX(P)$ (L is true in P), i.e. $P, t \vdash L$, iff L is a conclusion from a justified argument;*
- *not $L \in WFSX(P)$ (L is false in P), i.e. $P, t \vdash not\ L$, iff L is a conclusion from an overruled argument; and*
- *$\{L, not\ L\} \cap WFSX$ (L is undefined in P), i.e. $P, t \nvdash L$ and $P, tu \vdash L$, iff L is a conclusion from a defensible argument.*

4 Multi-Agent Argumentation

Given several extended logic programs, each corresponding to an agent with an individual view of the world, we may want to obtain agents global believes. In this case we want to exchange parts of several independent or overlapping programs and to obtain a consensus among the programs concerning inference of literals. This requires multi-agents argumentation, and also cooperation among agents. In this section we extend both the *WFSX* inference operator and dialogue trees to cope with multi-agent system.

Definition 4.1 *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).*

Definition 4.2 *Let $\mathcal{A} = \{Ag_1, \ldots, Ag_n\}$ be a MAS and Body a sequence of literals $L_1, \ldots, L_l, not\ L_{l+1}, \ldots, not\ L_m$ then*

$$
\begin{array}{rcll}
\mathcal{A} \vdash_i L & \textit{iff} & \mathcal{A}, \emptyset, \emptyset, t \vdash_i L & (1) \\
\mathcal{A}, LA, GA, M \vdash_i true & & & (2) \\
\mathcal{A}, LA, GA, M \vdash_i (L_1, L_2) & \textit{iff} & \mathcal{A}, LA, GA, M \vdash_i L_1\ \&\ \mathcal{A}, LA, GA, M \vdash_i L_2 & (3) \\
\mathcal{A}, LA, GA, M \vdash_i not\ L & \textit{iff} & M = t\ \&\ \mathcal{A}, \emptyset, GA, tu \nvdash_i L\ or & (4) \\
 & & M = tu\ \&\ \mathcal{A}, GA, GA, t \nvdash_i L & \\
\mathcal{A}, LA, GA, t \vdash_i L & \textit{iff} & L \notin LA\ \&\ \exists\ L \leftarrow Body \in \mathcal{A}\ \& & (5) \\
 & & \forall\ 1 < k < l,\ \exists\ 1 < j < n\ \textit{such that} & (5.1) \\
 & & \quad \mathcal{A}, LA \cup \{L\}, GA \cup \{L\}, t \vdash_j L_k\ \& & \\
 & & \forall\ l+1 \leq k \leq m,\ \forall\ 1 \leq j \leq n: & (5.2) \\
 & & \quad \mathcal{A}, LA \cup \{L\}, GA \cup \{L\}, t \vdash_j not\ L_k & \\
\mathcal{A}, LA, GA, tu \vdash_i L & \textit{iff} & L \notin LA\ \&\ \exists\ L \leftarrow Body \in \mathcal{A}\ \& & (6) \\
 & & \mathcal{A}, LA \cup \{L\}, GA \cup \{L\}, tu \vdash_i Body, not \neg L & (6.1)
\end{array}
$$

$\mathcal{A} \models L$ *iff* $\{\bigcup_{i=1}^{n} \mathcal{A}\} \models L$

While 1. to 4. and 6. of $\vdash_i$ are identical to $\vdash$, item 5 differs. Item 5 states that an agent Ag_i can infer L if it exist a rule for L whose body can be proven with the help of some other agent. Item 5.1 states that, for all objective literals $L_k \in Body$, at least one agent Ag_j should be able to prove L_k. If an agent Ag_i has a rule $L_k \leftarrow Body$ then it tries to prove L_k itself; otherwise L_k can be proven by another agent Ag_j $(j \neq i)$. In other words, a literal L_k is true if at least one agent can built an argument. itself or by using cooperation. Item 5.2 states that all agents must agree about all default assumptions *not* $L_k \in Body$. In other words, no agent can *undercut* none of the objective literals *not* $L_k \in Body$. Note that, to prove the non-falsity of L (item 6) all agents must agree on the non-falsity of *not* $\neg L$. In other words, L cannot be *rebuted* by other agents (including itself).

Multi-agent inference is closely related to the single agent semantics but, in general, they do not coincide. The main difference is that for multi-agent inference default assumptions are treated locally, whereas conclusions via arguments are subject to global consensus.

Proposition 4.1 $\vdash$ *is not equivalent to* $\vdash_i$ $(i > 1)$.

Example 4.1 *For* $Ag_1 = \{a \leftarrow not\ b\}$, $Ag_2 = \{b\}$ *and* $\mathcal{A} = \{Ag_1, Ag_2\}$ *we have* $\mathcal{A} \vdash_1 not\ b, not\ a$, $\mathcal{A} \vdash_2 b, not\ a$, *and* $\mathcal{A} \vdash b, not\ a$. *At first sight it seems puzzling that* Ag_1 *does not infer* a. *The reason is that concluding* a *requires* Ag_2 *to agree on not* b *which is obviously not the case.*

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 one in order to counter-argue.

Example 4.2 *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 by default not guilty:* $\neg guilty \leftarrow not\ guilty$, *i.e. explicit negation is derived by default. 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.*

$\mathcal{A} = \{\{\}, \{\neg guilty \leftarrow not\ guilty\}, \{guilty \leftarrow seen\}, \{seen\}\}$ *is a MAS that models the trial knowledge, the agents being judge, defender, prosecutor and witness, respectively. 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, a multi-agent cooperation is necessary during an argumentation dialogue when an agent has just part of the argument:

Definition 4.3 *Let Ag be an extended logic 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 $n < i \leq m$, for every objective literal L_j in the antecedent of r_i there is a $k < i$ such that L_j is the consequent of r_k. For r_n, for some objective literal L_j in the antecedent of r_n there is no $k < n$ such that L_j is the consequent of r_k;*
2. *L is the consequent of some rule of A;*
3. *No two distinct rules in the sequence have the same consequent.*

Example 4.3 *Continuing the example 4.2, the prosecutor has a partial argument $PA_1 = [guilty \leftarrow seen]$ for guilty.*

Definition 4.4 *Let A_i be an argument or a partial argument. A* cooperative dialogue *is a finite nonempty sequence of moves, $move_i = (Player_i, A_i)$, $(i > 0)$ such that*

1. *if $i \neq j$ then $A_i \neq A_j$;*
2. *Let A_i be an partial argument $[r_n; \ldots; r_m]$. $move_{i+1}$ is a move iff $A_{i+1} = [r_l; \ldots; r_k]$ is a (partial) argument for an objective literal L_i in the antecedent of the rule r_n.*

Definition 4.5 *Let $\mathcal{A}$ be a MAS. A finite sequence $A = [r_l; \ldots; r_k]$ of rules $r_i \in \mathcal{A}$ $(l \leq i \leq k)$ is a* cooperation result *for a partial argument $PA = [r_n; \ldots; r_m]$ if A is an argument for an objective literal L_j in the antecedent of the rule r_n.*

Definition 4.6 *A* cooperative dialogue tree *is a finite tree of moves such that each branch is a cooperative dialogue.*

Definition 4.7 *A cooperative dialogue is successful when the sequence of partial arguments is a cooperation result. A cooperative dialogue tree is successful if at least one branch is successful.*

We make use of a dialogue tree where each agent stores the dialogues it has been involved in. To fully implement the algorithm sketched in section 6, the dialogue tree of Def. 3.3 has to be extended to include cooperation in dialogues.

Definition 4.8 *A* dialogue tree *is a finite tree of moves such that*

1. *Each branch is a dialogue;*
2. *Each $move_i$ is an argument $A_i = [r_1, \ldots, r_n] \cup A_j$. If $[r_1, \ldots, r_n]$ is an argument (resp. a partial argument) then $A_j = \emptyset$ (resp. A_j is generated by a cooperative dialogue tree);*
3. *If $Player_i = P$ then the children of $move_i$ are all the defeaters of an argument A_i.*

To implement an argumentation algorithm we need agents with an expressive knowledge system and reaction rules such as vivid agents [16, 14].

5 Vivid Agents

A *vivid agent* is a software-controlled system, defined in [15], whose state is represented by a knowledge base, and whose behavior is represented by means of *action* and *reaction rules*. In this section we recap some basic issues of vivid agents needed for the underestanding of the algorithm and examples in the next section. For detail see [13].

The basic functionality of a vivid agent comprises a knowledge system (including an update and an inference operation), and the capability to represent and perform actions in order to be able to generate and execute plans. Since a vivid agent is 'situated' in an environment with which it has to be able to communicate, it also needs the ability to react in response to perception events, and in response to communication events created by the communication acts of other agents. Simple vivid agents whose mental state comprises only beliefs, and whose behavior is purely reactive, i.e. not based on any form of planning and plan execution, are called *reagents*. A reagent $R = \langle X, EQ, RR \rangle$, on the basis of a knowledge system $\boldsymbol{K}$ consists of:

1. a knowledge base $X \in L_{\mathrm{KB}}$,
2. an event queue EQ being a list of instantiated event expressions, and
3. a set RR of *reaction rules*.

A multi-reagent system is a tuple of reagents $\mathcal{S} = \langle R_1, \ldots, R_n \rangle$.

5.1 Operational Semantics of Reaction Rules

Reaction rules encode the behavior of vivid agents in response to perception events created by the agent's perception subsystems, and to communication events created by communication acts of other agents. [15] distinguishes between epistemic, physical and communicative reaction rules, and uses L_{PEvt} and L_{CEvt} to denote the perception and communication event languages, and $L_{\mathrm{Evt}} = L_{\mathrm{PEvt}} \cup L_{\mathrm{CEvt}}$. The following table describes the different formats of epistemic, physical and communicative reaction rules:

$$
\begin{array}{rcl}
\mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon(U), S],\ \mathit{Cond} \\
\mathsf{do}(\alpha(V)),\ \mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon(U), S],\ \mathit{Cond} \\
\mathsf{sendMsg}[\eta(V), R],\ \mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon(U), S],\ \mathit{Cond}
\end{array}
$$

The event condition $\mathsf{recvMsg}[\varepsilon(U), S]$ is a test whether the event queue of the agent contains a message of the form $\varepsilon(U)$ sent by some perception subsystem of the agent or by another agent identified by S, where $\varepsilon \in L_{\mathrm{Evt}}$ represents a perception or a communication event type, and U is a suitable list of parameters. The epistemic condition $\mathit{Cond} \in L_{\mathrm{Query}}$ refers to the current knowledge state, and the epistemic effect $\mathit{Eff} \in L_{\mathrm{Input}}$ specifies an update of the current knowledge state.

Physical Reaction: $\mathsf{do}(\alpha(V))$ calls a procedure realizing the action α with parameters V.

Communicative Reaction: $\mathsf{sendMsg}[\eta(V), R]$ sends the message $\eta \in L_{\mathrm{CEvt}}$ with parameters V to the receiver R.

Reaction rules are triggered by events. The agent interpreter continually checks the event queue of the agent. If there is a new event message, it is matched with the event condition of all reaction rules, and the epistemic conditions of those rules matching the event are evaluated. If they are satisfiable in the current knowledge base, all free variables in the rules are instantiated accordingly resulting in a set of triggered actions with associated epistemic effects. All these actions are then executed, leading to physical actions and to sending messages to other agents, and their epistemic effects are assimilated into the current knowledge base.

6 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 [7] 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 $ask(L)$.
2. and replies to a request by $reply(L, A)$ if the agent won the argumentation; otherwise, it replies $reply(not\ L, none)$.
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 $oppose(L', A', L. A)$ 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. In case 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 in terms of reaction rules. 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 (as defined in section 4). Figure 1 shows how to use vivid reaction rules to implement this algorithm[1].

Algorithm:

Proponent, Argumentation:

1. On receipt of a *query*, from an external agent[2], to infer L then try to built an argument itself (see rule 1 on figure 1) or by asking for cooperation to complete a partial argument PA (see rules 2,4): if there is one argument A then create a dialogue tree for A and propose it to all involved agents, i.e. agents that are able to argue; otherwise, answer that L is a false conclusion (6,7).
2. On receipt of an *agreement* (13), check the states of all dialogues: if some dialogue 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 (14); otherwise, reply negatively (15).
3. On receipt of an *oppose*, put the counter-argument CA on the dialogue tree. Try to built an counter-argument CCA as a proponnent, itself (16) or by asking for cooperation to complete a partial argument PA (17): if there is such CCA then put it on the dialogue tree and propose CCA to all involved agents (18). If there is no such CCA then answer the initial request negatively (19,21).

Opponent, Argumentation:

1. On receipt of a *proposal*, try to built a counter-argument CA as an opponent, itself (8) or by asking for cooperation to complete a partial argument PA (9,10), and then oppose. If there is no such CA then *agree* with proposal (11,12).

Proponent and Opponent, Cooperation for L:

1. If some objective literal L cannot be infered because there is no rule $L \leftarrow Body$ then query all involved agents, i.e. agents that are able to cooperate to infer L.
2. On receipt of a *query*, from an agent that is not external, to infer L then try to built an argument A for L, itself (3) or by cooperation (2,5) and reply it. If there is no such A then reply negatively (6,7).

The example below shows how dialogues can be expressed by reaction rules which are executable. Vivid agents have a further advantage as tested for implemented legal reasoning. They are formally underpinned, which allows to verify protocols. In the proof theory lined out in [16, 14] we can prove the following proposition statingfootnotesize that inference by $\vdash_i$ is equivalent to a reply transition eventually reached by the vivid agent:

[1] Due to lack of space we do not describe the following meta predicates but their meaning is straightforward: $argument/2$, $partialArgument/3$, $possible\_attack/3$, $built\_argument/3$, $create\_dialogueTree/2$, $put\_dialogueTree/4$, $removeAgent/3$ and $verify\_dialogueTree/2$.

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

1. $send(propose(L, A), LAg_a)$,
   $asked(Ag, L)$,
   $proposed(L, A, LAg_a) \longleftarrow$
   $recv(ask(L), Ag)$,
   $externalUser(Ag)$,
   $argument(L, A)$,
   $built\_dialogueTree(L, A)$,
   $listAgentArgue(LAg_a)$.
2. $send(ask(L_n), LAg_c)$,
   $waitReply(L, PA, LAg_c)$,
   $asked(Ag, L) \longleftarrow$
   $recv(ask(L), Ag)$,
   $partialArgument(L, PA, L_n)$,
   $listAgentCooperate(LAg_c)$.
3. $send(reply(L, A), Ag) \longleftarrow$
   $recv(ask(L), Ag)$,
   $not\ externalUser(Ag)$,
   $argument(L, A)$.
4. $send(propose(L, A), LAg_a)$,
   $proposed(L, A, LAg_a)$,
   $not\ waitReply(L, PA, LAg_c) \longleftarrow$
   $recv(reply(L_n, A_n), Ag_c)$,
   $asked(Ag, L)$,
   $external\,Agent(Ag)$,
   $waitReply(L, PA, LAg_c)$,
   $built\_argument(PA, A_n, A)$,
   $built\_dialogueTree(L, A)$,
   $listAgentArgue(LAg_a)$.
5. $send(reply(L, A), Ag)$,
   $not\ waitReply(L, PA, LAg_c)$,
   $not\ asked(L, A) \longleftarrow$
   $recv(reply(L_n, A_n), Ag_c)$,
   $asked(Ag, L)$,
   $not\ externalAgent(Ag)$,
   $waitReply(L, PA, LAg_c)$,
   $built\_argument(PA, A_n, A)$.
6. $not\ waitReply(L, PA, LAg_c)$,
   $waitReply(L, PA, NLAg)$,
   $send(verifyReply, Id) \longleftarrow$
   $recv(reply(not\ L, none), Ag_c)$,
   $waitReply(L, PA, LAg_c)$,
   $removeAgent(Ag_c, LAg_c, NLAg_c)$,
   $myId(Id)$.
7. $send(reply(not\ L, none), Ag)$,
   $not\ waitReply(L, PA, [])$,
   $not\ asked(Ag, L) \longleftarrow$
   $recv(verifyReply, Id)$,
   $myId(Id)$,
   $waitReply(L, PA, [])$,
   $asked(Ag, L)$.
8. $send(oppose(CL, CA, L, A), Ag_a) \longleftarrow$
   $recv(propose(L, A), Ag_a)$,
   $possibleAttack(o, A, CL)$,
   $argument(CL, CA)$.
9. $send(ask(L_n), LAg_c)$,
   $waitOppose(CL, PA, L, A, Ag_a, LAg_c) \longleftarrow$
   $recv(propose(L, A), Ag_a)$,
   $possibleAttack(o, A, CL)$,
   $partialArgument(CL, PA, L_n)$,
   $listAgentCooperate(LAg_c)$.
10. $send(oppose(CL, CA, L, A), Ag_a)$,
    $not\ waitOppose(CL, PA, L, A, Ag_a, LAg_c) \longleftarrow$
    $recv(reply(L_n, A_n), Ag_c)$,
    $waitOppose(CL, PA, L, A, Ag_a, LAg_c)$,
    $built\_argument(PA, A_n, CA)$.
11. $send(verifyOppose, Id)$,
    $not\ waitOppose(CL, PA, L, A, Ag_a, LAg_c)$,
    $waitOppose(CL, PA, L, A, Ag_a, NLAg_c) \longleftarrow$
    $recv(reply(not\ CL, none), Ag_c)$,
    $waitOppose(CL, PA, L, A, Ag_a, LAg_c)$,
    $removeAgent(Ag_c, LAg_c, NLAg_c)$,
    $myId(Id)$.
12. $send(agree(L, A), Ag_a)$
    $not\ waitOppose(CL, PA, L, A, Ag_a, []), \longleftarrow$
    $recv(verifyOppose, Id)$,
    $myId(Id)$,
    $waitOppose(CL, PA, L, A, Ag_a, [])$.

Figure 1: Reaction rules - 1 to 12.

13. $send(verifyDialogue, Id)$
$not\ proposed(L, A, LAg_a)$,
$proposed(L, A, NLAg_a) \longleftarrow$
$recv(agree(L, A), Id)$,
$proposed(L, A, LAg_a)$,
$removeAgent(Ag, LAg_a, NLAg_a)$,
$myId(Id)$.

14. $send(reply(L, A), Ag)$
$not\ proposed(L, A, [])$,
$not\ asked(Ag, L) \longleftarrow$
$recv(verifyDialogue, Id)$,
$proposed(L, A, [])$,
$not\ proposed(L_n, A_n, [Ag_1|LAg])$,
$verifyDialogueTree$,
$asked(Ag, L)$.

15. $send(reply(not\ L, A), Ag)$
$not\ proposed(L, A, [])$,
$not\ asked(Ag, L) \longleftarrow$
$recv(verifyDialogue, Id)$,
$proposed(L, A, [])$,
$not\ proposed(L_n, A_n, [Ag_1|LAg])$,
$not\ verifyDialogueTree$,
$asked(Ag, L)$.

16. $send(propose(CCL, CCA), LAg_a) \longleftarrow$
$recv(oppose(CL, CA, L, A), Ag_a)$,
$put\_dialogueTree(CL, CA, L, A)$,
$possibleAttack(p, CA, CCL)$,
$argument(CCL, CCA)$,
$put\_dialogueTree(CCL, CCA, CL, CA)$,
$listAgentArgue(LAg_a)$.

17. $send(ask(L_n)), LAg_c)$,
$waitPropose(CCL, PA, LAg_c) \longleftarrow$
$recv(oppose(CL, CA, L, A), Ag_a)$,
$put\_dialogueTree(CL, CA, L, A)$,
$possibleAttack(p, CA, CCL)$,
$partialArgument(CCL, PA, L_n)$,
$listAgentCooperate(LAg_c)$.

18. $send(propose(CCL, CCA), LAg_a)$,
$not\ waitPropose(CCL, PA, LAg_c) \longleftarrow$
$recv(reply(L_n, A_n), Ag_c)$,
$waitPropose(CCL, PA, LAg_c)$,
$built\_argument(PA, A_n, CCA)$,
$put\_dialogueTree(CCL, CCA, CL, CA)$,
$listAgentArgue(LAg_a)$.

19. $send(verifyPropose, Id)$,
$not\ waitPropose(CCL, PA, LAg_c)$,
$waitOppose(CCL, PA, NLAg_c) \longleftarrow$
$recv(reply(not\ CL, none), Ag_c)$,
$waitPropose(CCL, PA, LAg_c)$,
$removeAgent(Ag_c, LAg_c, NLAg_c)$,
$myId(Id)$.

20. $send(reply(not\ L, none), Ag)$,
$not\ waitPropose(CCL, PA, [])$,
$not\ asked(Ag, L) \longleftarrow$
$recv(verifyOppose, Id)$,
$waitPropose(CCL, PA, LAg_c)$,
$asked(Ag, L)$.

Figure 2: Reaction rules - 13 to 20.

Proposition 6.1 *Let* $\mathcal{A} = \{Ag_1, \ldots, Ag_n\}$ *be a MAS. Then* $\mathcal{A} \vdash_i L$ *iff* $(Ag_i, RR_i, [ask(L)|EQ]) \rightarrow^{*} \stackrel{reply(L)}{\longrightarrow} (Ag_i, RR_i, EQ')$

Finally, we present an example to show the process of cooperation and argumentation. 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 4.2. 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 commiting the crime and therefore knows seen. As the defensor cannot rebut[3] this testimony, then it agrees and the prosecutor wins the argumentation dialogue. On the other side, 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. The trial knowledge is modelled by the MAS:

$$
\begin{array}{rcl}
\mathcal{A} & = & \{defender, prosecutor, witness_d, witness_p\} \\
defender & = & \{listAgentArgue([prosecutor]); listAgentCooperate([witness_d]); \\
 & & \quad \neg guilty \leftarrow not\ guilty\} \\
prosecutor & = & \{listAgentArgue([defender]); listAgentCooperate([witness_p]); \\
 & & \quad guilty \leftarrow seen\} \\
witness_p & = & \{listAgentCooperate([]); using\_glasses; seen\} \\
witness_d & = & \{listAgentCooperate([]); \\
 & & \quad need\_glasses; \neg seen \leftarrow need\_glasses, not\ using\_glasses\}
\end{array}
$$

The argumentation dialogue is shown in the figure 3. Note that the judge can conclude the accused is guilty in both cases. This is so because the testominies of the prosecution witness cannot be defeated. Nevertheless if the prosecutor witness could not remember that he was using glasses, the testemonies 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 obtain a consensual result.

7 Comparison and Conclusion

The work presented in this paper is based on work by Dung [5, 6] and Prakken and Sartor [12] 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. Prakken and Sartor are motivated by legal reasoning and define a rigorous and concise framework similar to Dung's. Prakken and Sartor also only deal with a single agent and therefore do not tackle the issue of cooperation. Defeasible priorities as used in [12] can be easily added to the implementation by accordingly redefining the meta predicate *(counter)argument*.

[3] This happens because the defensor is the oponnent, see Def.3.2

1.	$judge$	$\longrightarrow$	$prosecutor$	$ask(guilty)$
2.	$prosecutor$			*cooperation for a partial argument* PA_1
3.	$prosecutor$	$\longrightarrow$	$witness_p$	$ask(seen)$
4.	$prosecutor$	$\longleftarrow$	$witness_p$	$reply(seen, A_3)$
5.	$prosecutor$			*assimilates* $A_5 = A_3 \cup PA_1$
6.	$prosecutor$	$\longrightarrow$	$defender$	$propose(guilty, A_5)$
7.	$prosecutor$	$\longleftarrow$	$defender$	$agree(guilty, A_5)$
8.	$prosecutor$			*checks the dialogue*
9.	$judge$	$\longleftarrow$	$prosecutor$	$reply(guilty, A_5)$

1.	$judge$	$\longrightarrow$	$defender$	$ask(\neg guilty)$
2.	$defender$	$\longrightarrow$	$prosecutor$	$propose(\neg guilty, A_1)$
3.	$prosecutor$			*cooperation for a partial argument* PA_1
4.	$prosecutor$	$\longrightarrow$	$witness_p$	$ask(seen)$
5.	$prosecutor$	$\longleftarrow$	$witness_p$	$reply(seen, A_3)$
6.	$prosecutor$			*assimilates* $A_5 = A_3 \cup PA_1$
7.	$defender$	$\longleftarrow$	$prosecutor$	$oppose(guilty, A_5, \neg guilty, A_1)$
8.	$defender$			*cooperation to prove the golo* $\neg seen$
9.	$defender$	$\longrightarrow$	$witness_d$	$ask(\neg seen)$
10.	$defender$	$\longleftarrow$	$witness_d$	$reply(\neg seen, A_2)$
11.	$defender$			*assimilates* $A_6 = \emptyset \cup A_2$
11.	$defender$	$\longrightarrow$	$prosecutor$	$propose(\neg seen, A_6)$
12.	$prosecutor$			*cooperation to prove the golo* $using\_glasses$
13.	$prosecutor$	$\longrightarrow$	$witness_p$	$ask(using\_glasses)$
14.	$prosecutor$	$\longleftarrow$	$witness_p$	$reply(using\_glasses, A_4)$
15.	$prosecutor$			*assimilates* $A_7 = \emptyset \cup A_4$
14.	$defender$	$\longleftarrow$	$prosecutor$	$oppose(using\_glasses, A_7, \neg seen, A_6)$
15.	$defender$			*cooperation to prove the golo* $\neg using\_glasses$
16.	$defender$	$\longrightarrow$	$witness_d$	$ask(\neg using\_glasses)$
17.	$witness_d$			*cooperation to prove the golo* $\neg using\_glasses$
18.	$defender$	$\longleftarrow$	$witness_d$	$reply(not\ \neg using\_glasses, none)$
19.	$judge$	$\longleftarrow$	$defender$	$reply(not\ \neg guilty)$

$$
\begin{aligned}
A_1 &= [\neg guilty \leftarrow not\ guilty] \\
A_2 = A_5 &= [need\_glasses; \neg seen \leftarrow need\_glasses, not\ using\_glasses] \\
A_3 &= [seen] \\
A_4 &= [using\_glasses] \\
PA_1 &= [guilty \leftarrow seen]
\end{aligned}
$$

Figure 3: Trace of the consistent trial example.

In this article we showed the connection between top-down inference of WFSX and argumentation; extended top-down inference from single to multi-agent and discussed the relation; designed an argumentation language to specify argumentation protocols for multi-agent systems and implemented multi-agents inference using vivid agents.

References

[1] J. J. Alferes, C. V. Damásio, and L. M. Pereira. Top–down query evaluation for well–founded semantics with explicit negation. In A. Cohn, editor, *European Conference on Artificial Intelligence'94*, pages 140–144. John Wiley & Sons, 1994.

[2] J. J. Alferes, C. V. Damásio, and L. M. Pereira. A top-down derivation procedure for programs with explicit negation. In M. Bruynooghe, editor, *Proc. of the International Logic Programming Symposium'94*, pages 424–438. MIT Press, November 1994.

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

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

[5] P. M. Dung. An argumentation semantics for logic programming with explicit negation. In *10th Int. Conf. on Logic Programming*, pages 616–630. MIT Press, 1993.

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

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

[8] Allen Van Gelder, Kenneth Ross, and John S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceeding of the 7th ACM Symposium on Principles of Databse Systems*, pages 221–230. Austin, Texas, 1988.

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

[10] L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *2nd Int. Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 334–348, Lisboa, Portugal, June 1993. MIT Press.

[11] Luís Moniz Pereira and José Júlio Alferes. Well founded semantics for logic programs with explicit negation. In *B. Neumann (Ed.), European Conference on Artificial Intelligence*, pages 102–106. John Wiley & Sons, 1992.

[12] Henry Prakken and Giovanni Sartor. Argument-based extended logig prpgramming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 1997.

[13] I. A. Schroeder, M.; Móra and J. J. Alferes. Vivid agents arguing about distributed extended logic programs. In *Progress in Artificial Intelligence, 8th Portuguese Conference on Artificial Intelligence EPIA97*. Springer–Verlag, Coimbra, Portugal, 1997. to appear.

[14] Michael Schroeder, Rui Marques, Gerd Wagner, and José Cunha. CAP - Concurrent Action and Planning: Using PVM-Prolog to implement vivid agents. In *Proceedings of the fifth Conference on Practical Applications of Prolog*, 1997. to be published.

[15] Gerd Wagner. *Vivid Logic – Knowledge-Based Reasoning with Two Kinds of Negation*, volume LNAI 764. Springer–Verlag, 1994.

[16] Gerd Wagner. A logical and operational model of scalable knowledge-and perception-based agents. In *Proceedings of MAAMAW96, LNAI 1038*. Springer-Verlag, 1996.