An Approach for Argumentation-based Reasoning Using Defeasible Logic in Multi-Agent Programming Languages

Alison R. Panisson, Felipe Meneguzzi, Renata Vieira, and Rafael H. Bordini

Pontifical Catholic University of Rio Grande do Sul – PUCRS
Postgraduate Programme in Computer Science – School of Informatics (FACIN)
Porto Alegre – RS – Brazil
alison.panisson@acad.pucrs.br,
{felipe.meneguzzi,renata.vieira,rafael.bordini}@pucrs.br

Abstract. Argumentation systems are intimately related to nonmonotonic reasoning, of which defeasible reasoning is one widely-known approach. For example, the literature points out defeasible logic (a particular formalisation of defeasible reasoning) as a practical platform upon which to develop an argumentation system. In this paper, we develop an approach to endow AgentSpeak agents with argumentative capabilities using defeasible reasoning, aiming in future work the development of multi-agent applications where agents interact through argumentation-based dialogues. We demonstrate the viability of our approach through an implementation in the Jason platform.

Keywords: Argumentation-based reasoning, Agent-oriented programming language, Defeasible reasoning

1 Introduction

Argumentation has received significant interest in the multi-agent system community in recent years because it give us means for allowing an agent to reconcile: (i) conflicting information within itself; (ii) its informational state with new perception of the environment; and (iii) conflicting information from multiple agents through communication [1].

Argumentation can be divided into two main lines of research in the multi-agent community [1] (i) argumentation focused on reasoning (nonmonotonic reasoning) over incomplete, conflicting, or uncertain information, where arguments for and against certain conclusions (beliefs, goals, etc.) are constructed and compared; and (ii) argumentation focused on communication/interaction between agents that allows the exchange of arguments to justify a stance and to provide reasons that defend claims.

In this work we use defeasible reasoning as a platform for the development of argumentation systems, as argued in [2], where we develop argumentation-based reasoning using an adaptation of defeasible-prolog (d-Prolog for short), which is

a practical implementation of defeasible logic formalism. Our work differs of the literature in the sense that the argumentation-based reasoning is part of agent's reasoning. We argue that this approach allows more flexibility and transparency in the development of such agents with argumentative capabilities.

The remainder of this paper is organized as follows. In the next section, we present a background of the concepts and technologies used throughout the paper. Section 3 then describes the relation between defeasible reasoning and argumentation, where we describe some work which are related to our and that make the link between this concepts. Section 4 describes our approach of argumentation-based reasoning based in defeasible reasoning and the adaptation of defeasible logic formalist and its implementation in an agent-oriented programming language. In the final section of this paper, we make some final remarks and discuss possible directions for future work.

2 Background

In this section we describe the main concepts and technologies involved in our work. We begin with agent-oriented programming language giving focus to AgentSpeak language extension found in Jason platform, which is the language used to make our approach within the practice. Then we present defeasible reasoning concepts using the defeasible logic formalism and d-Prolog (logical implementation of defeasible logic). Then we describe argumentation, including concepts and definitions necessary for understanding the remainder of this paper.

2.1 Agent-oriented Programming Languages

In the agent-oriented programming paradigm, the agents are computational entities with autonomous behaviour (*i.e.*, able to make decisions and act without direct human intervention on unexpected circumstances). These computational entities are situated in an environment that they are able to sense (through sensors), act upon it (through effectors), and communicate through message passing.

One of the most studied architectures for cognitive agents is the BDI (*Beliefs-Desires-Intentions*) architecture which provides a particular structure for agent internal states based on "mental attitudes". The internal state of a BDI agent is formed by: (i) *Beliefs* that represent the information about the world (including itself and other agents) available to that agent; (ii) *Desires* representing the motivations of the agent, *i.e.*, the states of the environment that the agent would like to reach; and (iii) *Intentions* which are desires that the agent is committed to achieve by following particular *plans* of action.

There exist many agent-oriented programming languages and platforms, such as Jason, Jadex, Jack, AgentFactory, 2APL, GOAL, Golog, and MetateM, as pointed out in [3]. Those languages differ in the agent architecture used, in the

form of communication/interaction between them, and also on the programming paradigms that inspired or underlie each language.

Among the languages mentioned above, AgentSpeak(L), the language on which Jason [4] is based, is one of the best-known languages inspired by the BDI architecture. AgentSpeak(L) is an abstract logic-based agent-oriented programming language introduced by Rao [5], and subsequently extended and formalised in a series of papers by Bordini, Hübner, and various colleagues.

AgentSpeak(L) is based in the Procedural Reasoning System (PRS) (Figure 1) where the agents are equipped with a library of pre-compiled plans. Plans in PRS have the following components: (i) a *goal* – the post-condition of the plan (the things that it achieves); (ii) a *context* – the pre-condition for the plan, defining what must be true of the environment in order for the plan to be successful; and (iii) a *body* – the 'recipe' part of the plan – it may contain a list of actions and sub-goals in order to achieve the main goal.

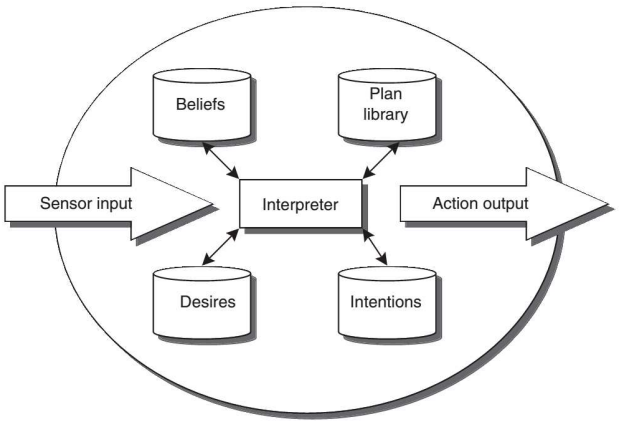

Fig. 1. The Procedural Reasoning System (PRS) ([4]).

Plans in AgentSpeak(L) have the following format:

```
triggering_event : context <- body.
```

where the *triggering_event* represents a new agent goal (or belief), which is to be pursued and has the format $!goal(Parameter)$, the *context* has preconditions needed to perform that plan to achieve that goal, and the *body* is a sequence of actions and sub-goals (which trigger others events and the use of other plans) to achieve the goal.

In particular, the main AgentSpeak(L) extensions available in Jason (a Java-based platform for the development of multi-agent systems), according to [4], and relevant to our current and future work are:

- **Strong negation:** Strong negation helps the modeling of systems where uncertainty cannot be avoided, allowing the representation of things that the agent believes to be true, believes to be false, and things that the agent is ignorant about;
- **Belief annotations:** One interesting characteristic present in Jason is that it automatically generates annotations for all beliefs in the belief base about the source from where the belief was obtained (sensing the environment, communication with other agents, or a mental note created by the agent itself). The annotation has the following format: $likes(john, music)[source(john)]$, stating that the source of the belief that john likes music is agent john itself.
- **Speech-act based communication:** Jason uses performatives based on speech acts in its communication language, which goes well with the availability of formal semantics of mental attitudes for the Jason extension of AgentSpeak;

Strong negation is useful in this work for writing defeasible rules and contradictory information. Belief annotations and speech-act based communication are useful in our future work, as we intend to use the reasoning mechanism presented in this paper in agent communication/interaction.

As mentioned above, the communication between agents is via message passing, and Jason's communication is based on speech acts and is performed by the pre-defined internal action '*.send*' that has the following format:

$$.send(receiver, illocutionary\_force, propositional\_content)$$

where *receiver* is the name of an agent (each agent has a unique individual name in the multi-agent system) or a list of agent names, for whom the message is being sent. The *propositional_content* is a term in AgentSpeak (a literal, triggering event, plan, or a list of literals or plans). The *illocutionary_force* denotes the intention of the sender (often called *performative*), as in speech-act theory. The formal semantics of receiving such messages is given in [6], and a complete list of all the illocutionary forces available can be found in [4].

New illocutionary forces can be easily added, as well as the effects that each will have on the agent's mental state. In Jason, agent plans can be written in AgentSpeak to give semantics to new performatives, hence providing an elegant and transparent way for programming agents that are capable of argumentation-based communication.

2.2 Defeasible Reasoning

Defeasible reasoning is a simple and efficient approach to nonmonotonic reasoning where the objective is to formalise nonmonotonic inferences of the type "birds generally fly". Such inferences hold only if a defeasible theory contains no rule inferring contrary information, which will be explained throughout this section using a specific formalisation of defeasible reasoning called defeasible logic [7,8].

Knowledge in a defeasible theory is organised as facts, rules, and a "superiority" relation. Rules are separated into strict rules, defeasible rules, and defeaters:

- **Facts**: facts are indisputable statements (*e.g.*, "Alison is a graduate student");
- **Strict rules**: strict rules are rules in the classical logic sense, where if the premises are indisputable (*i.e.*, facts) then so is the conclusion (*e.g.*, "graduate students are students").
- **Defeasible rules**: defeasible rules are rules that can be defeated by contrary evidence (*e.g.*, "graduate students usually study hard").
- **Defeaters or undercutting defeaters**: defeaters are rules that are used to prevent some conclusions from being derived rather than to draw particular conclusions;
- **Superiority relation**: superiority relation is a binary relation between rules which defines whether a rule is superior to another, and is used in case applying the rules would lead to contradicting conclusions.

Conclusions can be derived *strictly* or *defeasibly*. A conclusion is *strictly* derived if it is derived using only strict rules and facts contained in the knowledge base. A conclusion is *defeasibly* derived if it is derived using any clauses of the knowledge base including defeasible rules [9].

As defeasible rules represent disputable knowledge, it can be defeated by contrary evidence (provided by other rules). The two types of defeat are: (i) *rebut*, where the conclusion of the rule is defeated because another rule derives the negation of that conclusion (*i.e.*, a contrary conclusion can be obtained through another rule); and (ii) *undercut*, where the conclusion of the rule cannot be derived because an applicable defeater rule concludes the contrary (recall that the defeater cannot be used to conclude anything, it just prevents the conclusion of the contrary).

An answer to a query in a defeasible knowledge base can be of five types [9]:

- **definitely yes:** meaning that a conclusion is proved using only facts and strict rules, and therefore cannot be withdrawn when new knowledge is added to the available theory;
- **definitely no:** meaning that the negation of the queried conclusion can be proved using facts and strict rules;
- **presumably yes:** meaning that the conclusions can be defeasibly proved, so it might need to be withdrawn when new knowledge becomes available;
- **presumably no:** meaning that the negation of the query can be defeasibly proved, that is, although the query cannot be presently concluded, it might be concluded if new knowledge becomes available.
- **cannot tell:** it is not possible to answer the query either affirmatively or negatively (because both the queried formula and its conclusion can be defeasibly derived and the superiority relation does not favour one or the other).

Defeasible logic is an nonmonotonic logic introduced by Nute [7,8] as a way to formalise defeasible reasoning. Defeasible logic was made practical in the d-Prolog programming language [9] (an extension of Prolog based on defeasible logic). Defeasible logic and defeasible Prolog (d-Prolog for short) have all

types of knowledge representation mechanisms defined in the theory of defeasible reasoning, including facts, strict rules, defeasible rules, undercutting rules or defeaters, and superiority relation, as described above.

The representation in d-Prolog of defeasible rules and defeaters is possible through the introduction of the new binary infix functors `:=` and `:^` , respectively. It also introduces strong negation with the functor `neg`, which differs from the negation-as-failure operator `not`. D-Prolog also introduces a type of defeat by specificity, where more specific conclusions defeat more general ones. This is exemplified by the well-known Tweety triangle:

```
flies(X)     := bird(X).
neg flies(X) := penguin(X).
bird(X)      := penguin(X).
penguin(tweety).
```

All clauses in the example are defeasible rules. If we make a query for whether "tweety flies" using `?- flies(tweety)` in d-Prolog, the answer will be "presumably no" because the rule for *penguin* is more specific than the rule for *bird*. The specificity is defined by two inferences where it is tested for the two rules in conflict whether one of them can be derived from the other, and if that is the case, that which is derived from the other is defeated for being less specific. In this example, a rule with a `penguin` premise is more specific than one requiring `bird` to be inferred because the rule `bird(X) := penguin(X)` says that normally penguins are birds, hence the class of penguins is more specific than that of birds (membership to the latter can be inferred from membership to the former).

Another characteristic of defeasible logic is having the so called "preempting defeaters" [9] or "ambiguity blocking" [2], where defeasible rules that are rebutted by a superior rule are no longer available to rebut other rules.

An example of *preempting defeaters* is the knowledge base represented by Π below (where we use $\Rightarrow$ to refer to defeasible inferences):

$$\Pi = \left\{ \begin{array}{l} a \Rightarrow b \; x \Rightarrow e \\ b \Rightarrow c \; e \Rightarrow \neg c \\ c \Rightarrow d \; y \Rightarrow \neg e \\ a \\ x \\ y \end{array} \right\}$$

In this example, we may conclude d using the inferences $\{a,\ a \Rightarrow b,\ b \Rightarrow c,\ c \Rightarrow d\}$, although there is a derivation $\{x,\ x \Rightarrow e,\ e \Rightarrow \neg c\}$ which rebuts the rule that concludes c; this rule (the rule that derives $\neg c$) is defeated by rule $\{y,\ y \Rightarrow \neg e\}$ which prevents the use of that rule to rebut the inference of d.

2.3 Argumentation

Argumentation can be seen as the principled interaction of different, potentially conflicting arguments, for the sake of arriving at a consistent conclusion [1].

The survey presented in [1] states that argumentation in multi-agent systems has two main lines of research: (i) *autonomous agent reasoning*, such as belief revision and decision-making under uncertainty; and (ii) as a vehicle for facilitating *multi-agent interaction*, because argumentation naturally provides tools to designing, implementing and analysing sophisticated forms of interaction among rational agents.

According to [1], argumentation lends itself naturally to two main sorts of problems encountered in multi-agent systems:

- **Forming and revising beliefs and decisions:** Argumentation provides means for forming beliefs and decisions on the basis of incomplete, conflicting, or uncertain information. This is because argumentation provides a systematic means for resolving conflicts among different arguments and arriving at consistent, well-supported standpoints;
- **Rational interaction:** Argumentation provides means for structuring dialogues between participants that have potentially conflicting viewpoints. In particular, argumentation provides a framework for ensuring that interaction among agents respects certain principles (*e.g.*, consistency of each participant's statements).

As a reasoning mechanism, argumentation provides an alternative way to mechanise nonmonotonic reasoning. Argument-based frameworks view the problem of nonmonotonic reasoning as a process in which arguments for and against certain conclusions are constructed and compared. Nonmonotonicity arises from the fact that new premises may enable the construction of new arguments to support new beliefs, or stronger counterarguments against existing beliefs. As the number of premises grows, the set of arguments that can be constructed from those premises grows monotonically. However, because new arguments may overturn existing beliefs, the sets of beliefs may grow nonmonotonically [1].

Generally, argumentation is treated abstractly, where the content of individual arguments is not relevant and an overall structure of the relations between arguments is used instead. Abstract argumentation frameworks have their origins in [10], which studies the acceptability of arguments. In [10], the focus is on the attack relation between arguments, and the sets of arguments that defend its members, representing the ones that, given a set of arguments, are acceptable.

These systems are not concerned with the content of the arguments, nor where the relation of attack comes from (based on that structure). They concern in describing, given the set of arguments and the attack relation, which arguments are acceptable and under which conditions. Abstract argumentation systems also focus in the set of arguments that are mutually defensive, and thus cannot be attacked. Such set of arguments is referred to as being *admissible*.

In argumentation framework, an argument is a pair (S, C), where S denotes the *Support* and C denotes the *Conclusion*, meaning that C is supported by S. Arguments can be *defeated* (a.k.a. *attacked*) by other arguments. Defeat between arguments can be of two types: *rebut* and *undercut*, as defined below.

Definition 1 (Rebut). *Let* (S_1, C_1) *and* (S_2, C_2) *be two arguments. Argument* (S_1, C_1) *rebuts argument* (S_2, C_2) *iff* C_1 *is equivalent to the negation of* C_2 *(i.e.,* $C_1 \equiv \neg C_2$*).*

Definition 2 (Undercut). *Let* (S_1, C_1) *and* (S_2, C_2) *be two arguments. Argument* (S_1, C_1) *undercuts argument* (S_2, C_2) *iff* C_1 *is equivalent to the negation of a formula contained in* S_2 *(i.e.,* $\exists \varphi . \varphi \in S_2$ *and* $C_1 \equiv \neg \varphi$*).*

An abstract argumentation framework is defined as a tuple $\langle A, R \rangle$ with a set of arguments (A) and a binary relation (R) that defines an attack relation between the arguments.

The status of an argument depends on its *justification state*, where an argument is *justified* if it survives the attacks (or has no argument attacking it, or the set of argument defends it from the attacks) and it is *rejected* otherwise. The formal methods that describe this evolution (whether an argument turns out as justified or not) are called *argumentation semantics*.

Dung, in [10], proposed four well-established semantics for abstract argumentation systems:

- *complete extension/semantics*: E is a complete extension iff E is admissible and every arguments in S (the set of arguments) is acceptable;
- *grounded extension/semantics*: grounded semantics is best explained by the process of building it incrementally from unattacked arguments. The attacked arguments are suppressed, and the process is repeated until no new unattacked arguments arise after a deletion step. The set of all unattacked arguments identified up to that point is the grounded extension;
- *stable extension/semantics*: a stable extension attacks all arguments not included in it;
- *preferred extension*: is the maximum set of arguments in that argumentation framework that defends itself from attacks.

Other definitions found in [10] are:

- A set of arguments S is *conflict free* if there are no arguments A and B in S such that A attacks B.
- An argument A is an *acceptable argument* if, for all arguments that attack A, the set of arguments S defends A.
- A conflict-free set of arguments S is *admissible* iff each argument in S is acceptable with respect to S.

3 Defeasible Logic and Argumentation

Several works considering reasoning mechanisms based on argumentation can be found in the literature ([11,12,13,14,15]), most of them based on abstract argumentation systems at the theoretical level. Recently, Berariu [16] presents an approach for defeasible reasoning to implement argumentation-based reasoning in BDI agent. The author argues that, given the scarcity of practical work in

the area of argumentation-based reasoning, it is now time to address the challenge of putting such well-structured abstract theory into practice, proving its usefulness in real applications. Thus, the system developed in [16] extends the Jason platform with a module for argumentation, which is decoupled from the BDI reasoning cycle, operating in a customised belief base of the agent.

There are three key aspects of the system developed by Berariu [16]. First, the approach does not interfere in the execution of plans, creation of goals, or agent's commitments. Second, the argumentation module provides general argumentation and not just for a specific need of a particular application. With it, the agent is capable of nonmonotonic argumentation-based reasoning and can participate in dialogues and negotiation if the strategies and protocol which the agent must follow are programmed in it (but no example is given). Finally, as an instantiation of the latest instantiation of Dung's abstract formalism presented by Prakken's in [17], the work has two kinds of inferences: strict and defeasible rules (as defined in Section 2.2) and it uses the extension-based semantics, where the arguments' justification is defined:

justified: corresponds to skeptical justification (the argument belongs to all extensions);
defensible: corresponds to credulous justification (the argument belongs to at least one extension);
overruled: arguments that cannot be justified and are rejected.

Also, in [16], a number of special beliefs are used in the belief base representing specific elements of the argumentation formalism.

- defeasible and strict rules are represented using the predicates $defeasible\_rule(RuleName, RuleText)$ and $strict\_rule(RuleName, RuleText)$ respectively.
- contradictory and contrary information are represented with the predicates $contradictory(Literal1, Literal2)$ and $contrary(Literal1, Literal2)$ respectively.

These rules are then used in conjunction with a series of queries in the agent belief based, as follows. An agent queries the argumentation module with the belief $why(Proposition)$ and the response is in format of a belief $because(X, Y)$ with X being either *in* or *out*. The possible responses are:

- $because(out, unknown)$: the proposition is not in KB (Knowledge Base);
- $because(in, Premise(premise\_type))$: the proposition is a premise, and is not the result of any form of reasoning;
- $because(out, \neg Proposition)$: the proposition is not in the KB, but its negation is an acceptable argument;
- $because(in, \neg Preposition)$: the proposition is not in the KB, but its negation is an overruled argument;
- $because(in, Rule)$: the proposition is accepted and is the result of applying the rule $Rule$;

– $because(out, ListOfDefeats)$: returns a list the conclusions of the arguments that defeated the proposition;

The resulting system is quite significant, but it suffers from a key limitation. The fact that the reasoning mechanism to be a decoupled module and it to work with addition and reaction to beliefs limits its potential. It makes the programming agents more difficult, where it is necessary predict reactions of adding beliefs (response of module).

We found in the literature reasons for using defeasible logic [7,8] and its practical implementation of defeasible Prolog [9] (d-Prolog for short) as argumentation systems. We demonstrate that an adaptation of d-Prolog allows the implementation of argumentation-based reasoning in agent-oriented programming language, with all characteristics of [16] and, further, the approach allows the agents reasoning about such rules (defeasible and strict rules) during the executions of plans to achieve their goals as well as to query if an argument is acceptable or not at runtime.

In [2], a significant link between defeasible reasoning and argumentation is established, where Dung-like argumentation semantics is given for defeasible logic, providing a Dung-like argumentation system. The author argues that defeasible reasoning provides an efficient implementation platform for argumentation systems. The argumentation semantics proposed is for classical defeasible logic (as in [7,8]) and provides an ambiguity blocking argumentation system. The paper presents the usual pieces of an argumentation system: *logical language* and definitions of *argument*, *conflict between arguments*, and the *status of arguments*. The language is the same as defeasible logic (presented in Section 2.2). Arguments of the type (R, h) are formed by a set of rules R that have as consequent the literal h.

The types of arguments depend on the rules used:

– A *supportive argument* is a finite argument where no defeaters are used;
– A *strict argument* is an argument in which only strict rules are used;
– An argument that is not strict is called *defeasible*.

In [2] the characterisation of conclusions of defeasible logic in argumentation terms also is defined (being p a literal):

– if p is *strictly proved* there is a strict supportive argument for p;
– if p *is not strictly proved* there is no strict argument for p;
– if p is *defeasibly derived* there is a supportive argument for p;
– if p *is not defeasible derived* there is no supportive argument for p.

The definition of attack is usual, where defeasible arguments can attack or undercut other defeasible arguments and can be attacked or undercut by strict arguments (strict argument cannot be attacked). The paper defines the so called *defeasible semantics* which determines whether an argument is accepted or rejected in order to capture defeasible provability in defeasible logic [7,8] with ambiguity blocking (in original defeasible logic). This is formally defined below.

Definition 3. *An argument A for p is acceptable in a set of arguments S if A is finite, and: (a) A is strict, or (b) every argument attacking A is undercut by S (i.e., it is proved that the premises of all the arguments that attack A cannot be proved, the so called ambiguity blocking).*

This definition is achieved with ambiguity blocking, also called *preempting defeaters* defined in Section 2.2.

Definition 4. *An argument A is rejected by the sets of arguments S and T when A is not strict and: (a) a proper subargument of A is in S, or (b) it is attacked by an argument supported by T (i.e., the attacking argument must be supported by the set of justified arguments).*

The authors in [2], also, compare defeasible semantics with the grounded semantic defined by Dung in [10] and they define that:

- if an argument A is justified under grounded semantics, then A is justified under defeasible semantics;
- if an argument A is rejected under grounded semantics, then A is rejected under defeasible semantics;
- if a literal p is justified under grounded semantics, then p is justified under defeasible semantics;
- if a literal p is rejected under grounded semantics, then p is rejected under defeasible semantics.

4 The approach for argumentation-based reasoning using defeasible logic

We have implemented defeasible reasoning (defeasible logic more specifically) in Jason through a set of Prolog-like rules that had to be modified in order to be processed in Jason (*e.g.*, the cut operator is not available). We have adapted the rule presentation representing defeasible and strict knowledge to a form similar to the approach in [16], as follows:

Facts: facts are represented as in the d-Prolog implementation of defeasible logic, where "Alison is a graduate student" is represented by a simple predicate such as `grad_student(alison)`;

Strict Rules: the strict rules are represented as a special predicate `strict_rule(Head,Body)`, where for example "graduate students are students" is represented as `strict_rule(student(X),grad_student(X))`; the body can also be a list of predicates;

Defeasible Rules: the defeasible rules are represented as a special predicate `defeasible_rule(Head,Body)`, where "graduate student usually studies hard" is represented as `defeasible_rule(studies_hard(X), grad_student(X))`; as above, the body can also be a list rather than a single predicate.

Defeater: the defeaters are represented with the predicate `undercut_rule(Head,Body)`;

Superiority relation: the superiority relation is represented as `sup(`$Rule_1$`,`$Rule_2$`)` where $Rule_1$ is superior who $Rule_2$.

Another predicate is used to declare the complement of a proposition, for example, *good* is the complement of *bad*, in our representation we use the predicate $comp(good, bad)$ to define the complement. The adaptation of d-Prolog follows the example presented below. The rules are based in logic and the formal semantic and syntax of AgentSpeak language extension used can be found in [4].

```
strict_der(Content) :- Content.
strict_der([Content]):- strict_der(Content).
strict_der([First|Rest]):- strict_der(First) & strict_der(Rest).
strict_der(Content) :- strict_rule(Content,Condition) &
                       strict_der(Condition).
```

In this example we demonstrate the derivation of *Strict rules*, where first is checked if the queried content is a premise, after if it is a list of one element, after if it is a list of more elements, and finally if it is the *Head* of a strict rule and if the *Condition* (which derives the *Content*) is also strictly derived. These rules permits the agent queries if a content is strictly derived in its knowledge base (remember that strict knowledge is indisputable known).

From our implementation the agent can query if it has an argument to some proposition using the predicate $strict\_der(proposition)$ and $defeasible\_der(proposition)$ depending on whether the agent needs to have a strict or defeasible argument, respectively. Each rule used can be stored in a logical variable, as in logic programming, which characterizes the argument construction and, this argument, is which justify the derivation.

As described in Section 3, this implementation has a well-defined semantics called *defeasible semantics* which defines the acceptability of the arguments. The adaptation of d-Prolog does not change this semantics and the status of any argument can be defined by this formalisation.

4.1 Example

An agent has submitted a paper to AAMAS conference and believes that its paper will be accepted, so it will buy its ticket to Paris because it has an argument that conclude *go to Paris to present the paper.*

$$defeasible\_rule(go\_to\_paris\_to\_present(X), accepted(X)).$$
$$defeasible\_rule(accepted(X), submited(X)).$$
$$submited(paper).$$

The plan to buy a ticket has the following format (in Jason platform):

$$+!buyTicket(paris) : defeasible\_der(go\_to\_paris\_to\_present(paper)) \texttt{<-} buyTicket(paris).$$

Before the agent to buy its ticket, its coauthor informs it that the paper was submitted without a required field and this will invalidate its submission and so it will not go to Paris. The new knowledge received is:

$$strict\_rule(\neg go\_to\_paris\_to\_present(X), \neg accepted(X)).$$
$$strict\_rule(\neg accepted(X), incomplete(X)).$$
$$incomplete(paper).$$

The new information changes the conclusion of *go to Paris to present the paper*, and so the above plan no longer applies. The above example demonstrates our implementation, where more sophisticate reasoning are allowed, but to a simple example we argue that this is sufficient.

5 Conclusion

We have implemented argumentation-based reasoning using defeasible reasoning (defeasible logic more specifically) in an agent-oriented programming language based in BDI architecture. We used the AgentSpeak language extension of Jason to implement our approach through Prolog-like rules adapting d-Prolog, which is an implementation of the defeasible logic formalism.

Defeasible logic provides a platform to develop argumentation systems, as argued in [2], and defines the *defeasible semantics* which specifies the acceptability of arguments in these systems. We use this formalization to implement argumentation-based reasoning using defeasible reasoning in agent oriented programming language. In our approach the agents can query the existence of arguments and the acceptability of arguments in their own reasoning during the execution of plans and during the communication. So, the programming of such agents becomes more flexible and transparent.

We intend, as future work, to explore the argumentation-based reasoning developed in this work for agent interaction, where semantics for performatives, such as in the work presented in [6], can make direct reference to this reasoning mechanism, as well as it can refer to *agent attitudes* as defined in work such as [18,19]. Agent attitudes define how rigorous an agent is in accepting or asserting any proposition, depending on the arguments the agent has for that proposition. We have already taken the first steps towards that direction in [20].

Acknowledgements

Part of the results presented in this paper were obtained through research on a project titled “Semantic and Multi-Agent Technologies for Group Interaction”, sponsored by Samsung Eletrônica da Amazônia Ltda. under the terms of Brazilian federal law No. 8.248/91.

References

1. Maudet, N., Parsons, S., Rahwan, I.: Argumentation in multi-agent systems: Context and recent developments. In Maudet, N., Parsons, S., Rahwan, I., eds.: ArgMAS. Volume 4766 of Lecture Notes in Computer Science., Springer (2006) 1–16
2. Governatori, G., Maher, M.J., Antoniou, G., Billington, D.: Argumentation semantics for defeasible logic. J. Log. Comput. **14**(5) (2004) 675–702
3. Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F.: Multi-Agent Programming: Languages, Tools and Applications. 1st edn. Springer Publishing Company, Incorporated (2009)
4. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology). John Wiley & Sons (2007)
5. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away: agents breaking away. MAAMAW '96, Secaucus, NJ, USA, Springer-Verlag New York, Inc. (1996) 42–55
6. Vieira, R., Moreira, Á., Wooldridge, M., Bordini, R.H.: On the formal semantics of speech-act based communication in an agent-oriented programming language. J. Artif. Int. Res. **29**(1) (June 2007) 221–267
7. Nute, D.: Handbook of logic in artificial intelligence and logic programming. In Gabbay, D.M., Hogger, C.J., Robinson, J.A., eds.: Handbook of logic in artificial intelligence and logic programming. Oxford University Press, Inc., New York, NY, USA (1994) 353–395
8. Nute, D.: Defeasible logic. In: Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford University Press (2001) 353–395
9. Nute, D.: Defeasible Prolog. Research report (University of Georgia. Artificial Intelligence Programs). Artificial Intelligence Programs, University of Georgia (1993)
10. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77** (1995) 321–357
11. Amgoud, L., Cayrol, C.: A reasoning model based on the production of acceptable arguments. In: Proceedings of 8^{th} International Workshop on Non-Monotonic Reasoning, Breckenridge, Colorado (2000)
12. Amgoud, L., Cayrol, C.: A reasoning model based on the production of acceptable arguments. Ann. Math. Artif. Intell. **34**(1-3) (2002) 197–215
13. Atkinson, K., Bench-Capon, T.: Practical reasoning as presumptive argumentation using action based alternating transition systems. Artif. Intell. **171**(10-15) (jul 2007) 855–874
14. Bondarenko, A., Dung, P.M., Kowalski, R.A., Toni, F.: An abstract, argumentation-theoretic approach to default reasoning. Artif. Intell. **93** (1997) 63–101
15. Rahwan, I., Amgoud, L.: An argumentation based approach for practical reasoning. In Nakashima, H., Wellman, M.P., Weiss, G., Stone, P., eds.: AAMAS, ACM (2006) 347–354
16. Berariu, T.: An argumentation framework for bdi agents. In Zavoral, F., Jung, J.J., Badica, C., eds.: Intelligent Distributed Computing VII. Volume 511 of Studies in Computational Intelligence. Springer International Publishing (2014) 343–354

17. Prakken, H.: An abstract framework for argumentation with structured arguments. Argument and Computation **1**(2) (2011) 93–124
18. Parsons, S., McBurney, P.: Argumentation-based dialogues for agent coordination. group decision and negotiation. Group Decision and Negotiation (2004)
19. Parsons, S., Wooldridge, M., Amgoud, L.: An analysis of formal inter-agent dialogues. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1. AAMAS, New York, NY, USA, ACM (2002) 394–401
20. Panisson, A.R., Meneguzzi, F., Fagundes, M.S., Vieira, R., Bordini, R.: Formal semantics of speech acts for argumentative dialogues. In: Proceedings of the Thirteenth International Joint Conference on Autonomous Agents and Multiagent Systems. AAMAS '14 (2014)