Charles University in Prague
Faculty of Mathematics and Physics

DIPLOMA THESIS

Milutín Krištofič

Coordination among Logic agents

Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Prof. RNDr. Petr Štěpánek, DrSc.

Study programme: Computer Science, Theoretical Computer Science

This thesis could not have been completed without the support of a number of people. I would like to start by thanking my supervisor, Prof. RNDr. Petr Štěpánek, DrSc., for his support and encouragement during writing this thesis. I would also like to thank Martin Baláž, who introduce me the defeasible logic programming, which became my inspiration. And I would like to thank my parents, for being the best parents in the world, and my friends for having so much fun while writing this thesis.

Prague, August 6, 2009 Milutín Krištofič

Contents

Title: Coordination among Logic agents
Author: Milutín Krištofič
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: prof. RNDr. Petr Štěpánek, DrSc.
Supervisor's e-mail address: `stepanek@ksi.ms.mff.cuni.cz`

Abstract: In this thesis, we study the recent models for coordination and we propose a new model based on argumentation system. This system, named Argumentation BlackBoard for Coordination (ABBC), and its extension, Abstract ABBC, are based on successful argumentation approaches, such as abstract argumentation framework and defeasible logic programming. Our aim is to present necessary argumentation elements in open multi-agent system that consists of logic agents. In this work, we also describe a brief software design for the system.
Keywords: defeasible logic programs, blackboard systems, coordination, argumentation

Název práce: Koordinace mezi logickými agenty
Autor: Milutín Krištofič
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí diplomové práce: prof. RNDr. Petr Štěpánek, DrSc.
e-mail vedoucího: `stepanek@ksi.ms.mff.cuni.cz`

Abstrakt: V predloženej práci študujeme súčasné modely pre koordináciu a navrhujeme nový model založený na argumentačnom systéme. Tento systém, nazvaný Argumentation BlackBoard for Coordination (ABBC) a jeho rozšírenie, Abstract ABBC, zakladáme na úspešných argumentačných prístupoch ako je abstraktný argumentačný framework a defeasible logic programming. Náš hlavný zámer je zachytenie potrebných argumentačných zložiek v otvorenom multi-agentnom systéme s logickými agentami. V práci je popisaný aj stručný návrh na implementáciu takého systému.
Klíčová slova: defeasible logic programs, blackboard systems, koordinace, argumentace

Chapter 1

Introduction

One of the recent trends in computing is to develop ever more intelligent systems. That means the complexity of tasks we are capable of automating and delegating to computers has grown steadily as mentioned in [27]. The most frequently used computation models (procedural, object-oriented) have disadvantages which limit their use in these intelligent systems. In past years it was mainly the problem with verification and the lack of robustness.

While the earliest computer systems were isolated entities communicating only with their human operators, contemporary computer systems are usually interconnected. The logical consequence is to develop interaction tools for computer systems to interact with each other without the help of human users. This interaction will cover argumentation, coordination, cooperation, negotiation and other human-like social activities. This may be a potential problem when developing computer systems only as a procedural computation models.

In this thesis we are concerned only about one type of interaction, the coordination among logic computation models. Firstly, we will study the recent models, preferring the most abstract one. After a deep research in the concept of coordination, we will start to develop our own model for coordination. We decided that the other type of interaction, the argumentation, can be used as a tool for the coordination. The coordination itself will be a logical argumentation among seeking a common knowledge among a group of agents, decision making, distribution of task or resources, etc.

We will also study the argumentation and will be searching for an appropriate framework for our tool. The defeasible logic programming [10] shows an interesting connection between argumentation and non-monotonic logic. It also defined many argumentation elements, but it was developed only for one agent reasoning. So we will propose a distributed extension of defeasible logic programming, that we called Argumentation BlackBoard for Coordination (ABBC). We will do only the necessary modifications in the defeasible logic programming, most elements remain, because of their universality in argumentation.

For inference in the ABBCs are used only derivations and programs from defeasible logic programming. We will try to remove this disadvantage in framework, we called the Abstract ABBC. It will be based on the abstract argumentation framework from Dung [6]. We will propose the definitions for argumentation elements, that are not defined in abstract argumentation framework. We will define them similarly as they are in ABBC and defeasible logic programming. Therefore we will receive a framework that can be used as ABBC, but can use other types of inference, syntax, rules, relations, etc.

We will also describe a brief software design for Abstract ABBC, and ABBC. We believe that the Java Agent Development Framework (JADE) [2], which is the implementation of multi-agent systems that complies with the FIPA specifications [9] in Java language, is the best option for these systems. FIPA specifications consists of FIPA Agent Communication Language (FIPA ACL) [9] for communication, that can be used for interaction between an agent and the argumentation framework.

1.1 Contribution of the diploma thesis

We propose a novel approach for coordination among logical agents. It is based on application of an argumentation system as a tool, which can be used to make deals, propositions and other elements needed for an coordination.

We develop this argumentation system, the Argumentation Black Board for Coordination (ABBC) as an extension of defeasible logic programming (DELP) [10]. We prepared a distributed version of DELP with necessary modification.

We describe also an extension of ABBC, Abstract ABBC. It is based on ABBC, DELP and the abstract argumentation framework [6]. They can obtain any non-monotonic inference operator and syntax and also they can be limited by any set of argumentation rules.

Finally we draft a brief software design for Abstract ABBC.

1.2 Structure of the document

The thesis is structured as follows. In the chapter 2 we discuss social ability in multiagent systems. We shortly describe a philosophy and the best-known frameworks for coordination and afterwards we present our coordination approach based on argumentation. We also discuss the research in argumentation and we describe the abstract argumentation framework. In the chapter 3 we show the defeasible logic programming and we propose the ABBC framework based on DELP. In the chapter 4 we propose the Abstract ABBC and its brief software design. In chapter 5 we shortly discuss the related work in abstract argumentation framework and DELP. In chapter 6 we summarize this thesis and we outline the future work on the proposed frameworks.

Chapter 2

Social Ability in Multi-agent systems

Multi-agent systems have the capacity to play a key role in current and future development of computer science and its applications, because of their robustness and capability to cope with an increasing complexity of applications. We adopt definitions from Wooldridge's paper [27]. A multi-agent system composed of multiple interacting computing element, known as *agents*. Agents are acapable of *an autonomous action* and interacting with other agents. This interaction is not based on simple exchange of data, but rather imitates the human social activity: argumentation, coordination, cooperation, negotiation and the like.

These social activities become necessary in systems that can represent our best interests, because these interests, that computer systems represent on our behalf, are not always the same. For example we need negotiation for reaching agreements with one another on matters of self-interest, or co-ordination for recognizing when agents' beliefs, goals, or actions conflict.

There are not many studies of social ability in sum. One from philosophy is a classification of human dialogues according to the objectives of the dialogue by argumentation theorists Doug Walton and Erik Krabbe in [26]. They prepared six primary dialogue types: Information-Seeking Dialogue, Inquiry Dialogues, Persuasion Dialogues, Negotiation Dialogues, Deliberation Dialogues and Eristic Dialogues. Formal models of all of these dialogue types have been developed in recent years except for the Eristic Dialogues, which is a substitute for physical fighting and therefore it is not really important to be formalized.

Thus the social ability is currently studied separately for each particular type of a dialogue. In our work we declare a tool for a coordination, but we believe that it can be used also for other types of dialogues as well.

2.1 Coordination

A formal background of the coordination is not developed as much as the one of the argumentation that will be described, because there are many different point of views on the coordination itself. We consider the coordination in the ideas of coordination models [17] that consists of three elements: (1) the *coordinables*, which represent the entities being coordinated (2) the *coordination media*, which are the abstractions enabling agent interactions (3) the *coordination laws*, which define the behaviour of the coordination media in response to interaction events.

As a definition for coordination, we refer to the one from Thomas Malone and Kevin Crowstone [14]. That coordination is managing dependencies between activities. Therefore, if there is no interdependence, there is nothing to coordinate. Examples of common dependencies are shared resources, task assignments, prerequisite constraints, but also knowledge.

We can make our first simplification, because our environment consists only of logical agents. We assume that the knowledge describes everything important and therefore it describes also every possible dependency. Our only concern will be in managing knowledge. The possible implications are left to agents themselves.

It will be useful to mention a few approaches related closely to the coordination models. We restrict our focus on those ones mentioned in survey [3], that its authors believe represent the state of the art of two different communities. Hybrid models based on tuple centers from the coordination community and interaction protocols based on some communication languages from the agent community.

A coordination through tuple centers promotes a clear separation between the specification of components of the computation and the specification of their interactions or dependencies. A tuple centre [17] is a coordination virtual machine, that can be programmed to specify the behaviour of the coordination media. They are usually presented in the form of a blackboard system. The coordinables are agents, which use coordination language like Linda [17] for interaction. These languages are focused on synchronization, exchange of simple information, notification and reaction that are represented by a set of some simple coordination primitives. The coordination media are represented by the tuple centre and programs used to specify tuple centre constitute the coordination laws.

A coordination through interaction protocols [8, 9] is using an asynchronous messages that are structured in terms of a performative verb and a content. The perfomative expresses the intent of the agent in sending the message. They use an agent communication language like KQML [8] or the FIPA ACL [9]. The agents are described as finite state machines where states identify global states of the protocol and transitions represent messages labelled with a performative. There is an extension for the interaction protocol (see [12] for KQML and [20] for FIPA ACL), where formal semantics are proposed. The coordinables in this case are agents also, the coordination medium is an agent communication language and the coordination laws are expressed through the finite state machine representing the protocol. They do not separate specification of agents and their specification of interactions as well as tuple centers.

These approaches do not involve in more detail coordination, they can be seen rather as a necessary tool for modeling coordination. We believe that it is possible to go further and not to fall in the layer of application such as LuCe [5], a tuple centre-based system integrating Prolog and Java. In our case, this further approach is achieved by the fact that our system consists only of logical agents.

2.1.1 Coordination approach with the ABBC

In our new concept for a coordination, the agents will not only be the coordinables, but they can represent also the coordination law. Some agent might be specialized for some coordination jobs, like a planner agent, an agent responsible for distribution of tasks and resources, etc. The coordination medium will be a blackboard system, that we call ABBC, Argumentation Black Board for Coordination. The coordination laws will recall in their processes ABBC as a medium for communication with other agents. The ABBC system is based on argumentation [10] and we believe that it have all necessary properties needed for the coordination.

ABBC might be implemented as tuple centers, but also as interaction protocols. Our approach, as we will show, guarantees a separation between the specification of agents and their specification of interactions. We only need to implement agents for the interactions with ABBC properly. For every new coordination process, a special agent will be developed. It will use ABBC in the same way as other agents, but it will start ABBC as frequently as needed and with a specific input.

We previously mentioned dialogue types. Deliberation dialogue is a collaboration to decide, what action or a course of action should be adopted in some situation. This is one type of coordination. A formal model for deliberation was proposed in [16]. It is an eight-stage model from the philosophy field of argumentation, hence it is a good example to be introduced in the ABBC coordination.

The authors in [16] assume that agents will not enter any dialogue until they perceive it to be in their self-interest and in this thesis we share the same assumption. The deliberation dialogue is neither an attempt by one participant to persuade any of the others to agree to an initially defined proposal. An option which is optimal for the group when considered as a whole may be seen as sub-optimal from the perspective of each of the participants of the deliberation. This could happen because a demonstration of optimality requires more information than is held by any one of the participants at the start of the dialogue, or because individual participants do not consider all the relevant criteria for an assessment.

A model from [16] consists of eight stages: (1) *Open*, where the raising of a governing question about what is to be done takes place. It is like opening ABBC with same meaning query. (2) *Inform* stage consists of a discussion of desirable goals and any constraints of the possible actions can be viewed as the ABBC argumentation about the query. (3) *Propose*, where the suggestion of possible action-options can be made by a coordination law agent and it corresponds to opening a new ABBC with a proposal. (4) *Consider* and (5) *Revise* comprise of commenting on proposals. It can be also modeled by the ABBC argumentation about the proposal. (6) *Recommend* and (7) *Confirm* are an acceptance or a non-acceptance and their confirming. We can simulate it with a third ABBC. (8) *Close*, where a coordination law agent will end its process. This stages may occur in any order and may be entered by participants as frequently as desired, but this requirement is not a problem in the ABBC coordination.

David Vengerov proposed an adaptive communication and coordination [24] through the idea of a two level architecture. At the local level, abstract agents learn to behave optimally in certain environment and at the global level, agents use obtained knowledge in their decision-making throughout the next individual phase. This idea can be implemented in our approach, because there are also both the computation inside an agent and among the agents. However, Vengerov used this architecture in a neural net, or using Q-learning. They are both sub-symbols approaches and they are very different from symbols approaches such as logic. His idea also aims, more than our approach, to learn than coordination.

There are many formal models that represent other types of social ability in the terms of argumentation. A framework based upon a system of argumentation, which permits agents to negotiate in order to establish acceptable ways of solving problems, is described in [18]. An abstract argumentation framework for decision making is used in [25]. It suggests a Dung-style general framework that takes different arguments and a defeat relation among them as an input, and returns a status for each option as an output. These works show a possibility to define some coordination activities in an argumentation concept.

2.2 Argumentation

Argumentation systems formalise non-monotonic reasoning as the construction and comparison of arguments for and against certain conclusions. In particular, argumentation is an useful and an intuitive paradigm for doing non-monotonic reasoning. Moreover it can be applied to any form of reasoning with contradictory information. The contradictions may arise from reasoning with several sources of information, or they may be caused by disagreement about beliefs. Several argumentation systems have been proposed in the literature [4]. Some of them, the so-called *rule-based systems*, use a particular logical language with *strict* and *defeasible rules*.

In [19], Prakken and Vreeswijk discuss the following five elements of argumentation systems. (1) *A logical language* is what the argumentation systems are built on. It also contains an associated notion of logical consequence and definition of the notion of an argument. (2) *Arguments* corresponds to a tentative proof in the underlying logic. Sometimes they are defined as a tree of inferences grounded in premises and sometimes as a sequence of such inferences. (3) *Conflict between arguments.* The terms attack and counterargument are also used. (4) *Defeat between arguments* is a form of evaluating conflicting pairs of arguments, or, in other words, a form of determining whether an attack is successful. It has the form of a binary relation between arguments. (5) *The dialectical status of arguments* is a definition of all ways in which the arguments interact.

In the literature, three types of conflicts are discussed. (a) *Rebutting* attack, which is symmetric and occurs, when the arguments have contradictory conclusions, as in 'Tweety flies, because it is a bird' and 'Tweety does not fly because it is a penguin'. The other two types of conflict are not symmetric. (b) *Assumption attack* makes a non-provability assumption. (c) *Undercutting* attack is when one argument challenges, not a proposition, but a rule of inference of another argument.

Argumentation systems vary in their grounds for determining the defeat relations. A frequently used criterion is the *specificity* principle, which prefers arguments based on the most specific information. It is a syntactic criterion. However, often only the domain-specific criteria are available and these are often defeasible. It is important to note that a defeat relation does not yet tell us what arguments are justified; it only tells us something about the relative strength of two individual conflicting arguments. We need a declarative or a procedural form to declare, which arguments are acceptable, not acceptable and undecided.

In our thesis, we will discuss two famous argumentation systems. The most abstract argumentation framework, introduced by Dung in [6], and, the defeasible logic programming [10], which is an approach to realise non-monotonic reasoning via dialectical argumentation. The former is described for better understanding and formalization of argumentation. It is the inspiration for our Abstract ABBC framework. The later is extended in our ABBC framework.

2.2.1 Dung's abstract argumentation framework

This section describes the abstract argumentation framework proposed by Dung in [6]. This framework leaves the internal structure of an argument completely unspecified. Dung treats the notion of an argument as a primitive, and exclusively focuses on the ways arguments interact. Thus Dung's framework is of the most abstract kind.

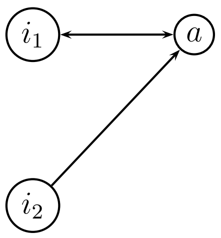
The way humans argue is based on a very simple principle which is summarized by an old saying: "The one who has the last word laughs.". The goal of an argumentation theory is to give a scientific account of this basic principle. In this point of view, the following definitions from [6] were proposed.

Definition 2.2.1 (Argumentation framework [6]). An argumentation framework is a pair $AF = \langle AR, \mathit{attacks} \rangle$, where

- AR is set of arguments
- *attacks* is a binary relation on AR

For two arguments A, B, the meaning of $attacks(A, B)$ is that A represents an attack against B. An example of abstract argumentation framework with a graph representation:

Example 2.2.1. $\langle\{i_1, i_2, a\}, \{(i_1, a), (a, i_1), (i_2, a)\}\rangle$.



Definition 2.2.2 (Conflict-free set [6])**.** A set S of arguments is said to be conflict-free if there are no arguments A, B in S such that A attacks B.

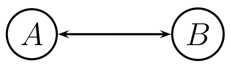
1. An argument $A \in AR$ is *acceptable* with respect to a set S of arguments iff for each argument $B \in AR$: if B attacks A then B is attacked by S.
2. A conflict-free set of argument S is *admissible* iff each argument in S is acceptable wrt S.

There are three types of semantics: preferred semantics (credulous), stable semantics and grounded semantics (skeptical).

Definition 2.2.3 (Preferred extension [6])**.** A preferred extension of an argumentation framework AF is a maximal (wrt set inclusion) admissible set of AF.

Every argumentation framework possesses at least one preferred extension. In the worst case, it is an empty set. In Example 2.2.1 it is not difficult to see that AF has exactly one preferred extension $E = \{i_1, i_2\}$. It is a possible to have more than one preferred extension, for example:

Example 2.2.2. *This example represent well-known Nixon Diamond. A represents "Nixon is anti-pacifist since he is a republican" and B represents "Nixon is a pacifist since he is a quaker".*



Definition 2.2.4 (Stable extension [6])**.** A conflict-free set of arguments S is called a stable extension iff S attacks each argument which does not belong to S.

Every stable extension is a preferred extension, but not vice versa. In Example, 2.2.1 the stable extension is $\{i_1, i_2\}$.

Definition 2.2.5 (Characteristics function [6])**.** The characteristics function, denoted by F_{AF} of an argumentation framework $AF = \langle AR, attacks\rangle$ is defined as follows:

$$F_{AF} : 2^{AR} \rightarrow 2^{AR}$$

$$F_{AF}(S) = \{A|A \text{ is acceptable wrt } S\}$$

Definition 2.2.6 (Grounded extension [6])**.** A grounded extension of an argumentation framework AF, denoted by GE_{AF} is the least fixed point of F_{AF}.

Example 2.2.2 does not have a grounded extension. A skeptical reasoner will not conclude anything.

Example 2.2.3. *Example 2.2.1 has the following grounded extension:*

$$F_{AF}(\emptyset) = \{i_2\}$$

$$F^2_{AF}(\emptyset) = \{i_2, i_1\}$$
$$F^3_{AF}(\emptyset) = F^2_{AF}(\emptyset)$$

Definition 2.2.7 (Complete extension [6])**.** An admissible set S of arguments is called a complete extension iff each argument which is acceptable wrt S, belongs to S.

A conflict-free set of arguments E is a complete extension iff $E = F_{AF}(E)$. Each preferred extension is a complete extension, but not vice versa. The grounded extension is the least (wrt set inclusion) complete extension.

In previous definitions, we have shown which arguments may be put together and support a coherent point of view. In following we mention an definition for the different status that an argument may have from [25].

Definition 2.2.8 (Argument status [25])**.** Let $AF = \langle AR, attacks \rangle$ be an argumentation framework and $\xi_1, \ldots, \xi_n$ its extension under a given semantics. Let $a \in AR$.

1. a is skeptically accepted iff there exists at least one non-empty extension and for all i $(1 \leq i \leq n)$ $a \in \xi_i$.
2. a is credulously accepted iff there exists at least one extension that contains a and one that does not contain a.
3. a is rejected iff there does not exist extension that contains a.

A direct consequence of this definition is that an argument is skeptically accepted iff it belongs to the intersections of all extension, and it is rejected iff it does not belong to the union of all extensions.

The idea of argumentation reasoning is that a statement is believable if it can be argued successfully against the attacking arguments. The beliefs of a rational agent are characterized by the relations between the "internal" arguments supporting his beliefs and the "external" arguments supporting contradictory beliefs. This double check is the inspiration for our thesis.

2.3 Communication

The agent community does not treat communication in a low-level way, such as the research in concurrent systems or computer networks. It is studied more like we think of utterances changing the world in the way that physical actions do. We can ask, how speech changes the world. In [27], there are examples from a philosopher Austin such as declaring a war, or saying 'I now pronounce you man and wife'. These utterances clearly change the state of the world in some way.

Austin identified a number of *performative verbs*, which correspond to various different types of speech acts, such as request, inform and promise. He also referred to the conditions required for the successful completion of performatives: (1) There must be a convention for the performative. (2) The procedure must be executed correctly and completely. (3) The act must be sincere.

Speech act theory treats communication as an action. It is predicated on the assumption that speech actions are performed by agents just like other actions, in the furtherance of their intentions. The agent communication languages (ACLs) FIPA ACL [9] and KQML [8], which were discussed in section 2.1 a little, were developed on the basis of the speech theory. We will discuss them again, because we believe that they can be used in an implementation of the ABBC.

In a general, a speech act can be seen to have two main components: A performative verb and a prepositional content. Sometimes there is a discussion about language and ontology, but in our case, when agents are using the same logic language, it is not interesting.

The KQML [8] is a message-based language for agent communication. It was developed by an institution DARPA. KQML defines various acceptable performatives such as ask-if, perform, tell, reply, etc. A KQML message has a perfomative and a number of parameters such as receiver, content, sender. Here is an example KQML message from [27]:

Example 2.3.1. *The intuitive interpretation of this message is that the sender is asking about the price of IBM stocks.*
(ask-one

> *: content (PRICE IBM ?price)*
>
> *: receiver stock-server*
>
> *: language LPROLOG*

)

Despite the success of KQML in the multiagent systems community, it was criticized for many issues. The performative set was not tightly constrained nor were the semantics of KQML ever rigorously defined. Thus we do not know whether the agents are using the language properly. These criticism led to the development of a new, but rather closely related language by The Foundation for Intelligent Physical Agents (FIPA), which started a work on a program of agent standards.

An FIPA ACL [9] has a basic structure, quite similar to KQML. There are 20 types of performatives in FIPA ACL and messages contain items such as sender, receiver, content and performative. The example message 2.3.1 can be an example of FIPA ACL message also. There are only two basic performatives, *Inform* and *Request*, in FIPA ACL. All others are macro definitions, defined in the terms of these.

FIPA ACL successfully separates the utterance level of a communicative act (the process of sending a message) from the level of illocution (the mental states characterizing the sender). FIPA's definition refers to agents' mental states. In [22], the authors claim that this approach supposes, in essence, that agents can read each other's minds, which they do not believe to hold for agents. They showed that ACL's formal semantic must emphasize a social agency. In this article and in the proposed future implementation, we will use a way from [7], where authors distinguish an agent's communication strategy, which may be private and will be determined by the agent's goals and beliefs, from the public protocol which lays down the conventions of communication in the terms of publicly observable events.

Chapter 3

Argumentation Black Board for Coordination

In this chapter, we will propose our novel argumentation system, an Argumentation Black Board for Coordination (ABBC). As we mentioned before, it can be used as a tool for coordination and for argumentation in same time.

An ABBC is a Black Board system (BB) with knowledge base and a query. This architectural model, where a common knowledge base, the blackboard in our approach, is iteratively updated by a diverse group of specialist knowledge sources, in our case logic agents, starting with a problem specification as a query and ending with a solution as an answer for the query. These updates are allowed by the rules of argumentation.

All elements of argumentative framework for ABBCs were obtained from defeasible logic programming (DELP). Moreover, we decided to make an distributed abstraction of DELP. The key definitions and properties from DELP are only slightly changed in ABBC, because of their universality in argumentation.

An agent will have the opportunity in any time recall an ABBC with knowledge base and a query. Each agent in the environment is notified about a newly opened ABBC and it has a choice to join to the ABBC under certain conditions. After the argumentation in the ABBC, the answer will be produced whether a query is or is not warranted in a common knowledge among a group of agents. This answer and also whole argumentation process may be very useful for all agents in the environment and used for various activities, such as coordination, argumentation, learning, etc.

3.1 Defeasible Logic Programming

In the following paragraphs we briefly describe Defeasible Logic Programming (DELP) [10]. The following definitions and examples in this section are taken from [10]. The DELP successfully combines Logic Programming and Defeasible Argumentation, that is used as an inference mechanism for warranting the entailed conclusions.

By a *literal* L we mean a ground atom A, or a negated ground atom $\sim A$. The Symbol $\sim$ represents the *strong negation*. A *fact* is a literal and a rule is an ordered pair $\langle H, B \rangle$, where H is a literal and B is a finite non-empty set of literals. A *defeasible rule* is an rule denoted $H \prec B$, and a *strict rule* is a rule denoted $H \leftarrow B$. We can write the strict rules as $L_0 \leftarrow L_1, \ldots L_n (n > 0)$ and the defeasible rules can be written similarly. In the context of Logic programming a set of strict rules is neither a Normal Logic Program nor an Extended Logic Program, then it has not be defined at all in Logic Programming, yet. A Normal Logic Program [1] has in the head of rules only an atom and an Extended Logic Program [1] can possibly have default negation in the body of rules.For this reason we will call set of strict rules and facts simply a *program*. Some example will use *schematic rules* with variables, which stand for the set of all ground instances of them. Variables will be denoted with an initial uppercase letter.

Definition 3.1.1 (Defeasible logic program [10])**.** A Defeasible Logic Program (delp) $\mathcal{P} = (\Pi, \Delta)$ is a possibly infinite set of facts, strict rules and defeasible rules, where

Π is a subset $\mathcal{P}$ of facts and strict rules, that is non-contradictory

Δ is a subset $\mathcal{P}$ of defeasible rules

A strict rule is used to represent sound knowledge, like $mammal \leftarrow dog$, and a defeasible rule used for representing tentative information. $H \prec B$ means that *"Reasons to believe in the antecedent B provide reasons to believe in the consequent H"* [21].

Example 3.1.1. *Here follows the most notorious delp.*

$$\Pi = \left\{ \begin{array}{l} bird(X) \leftarrow chicken(X) \\ bird(X) \leftarrow penguin(X) \\ {\sim}flies(X) \leftarrow penguin(X) \\ chicken(tina) \\ penquin(tweety) \\ scared(tina) \end{array} \right\} \Delta = \left\{ \begin{array}{l} flies(X) \prec bird(X) \\ {\sim}flies(X) \prec chicken(X) \\ flies(X) \prec chicken(X), \\ \hfill scared(X) \end{array} \right\}$$

As we can see the program is easy to read. We have facts and rules about birds' flying abilities.

Definition 3.1.2 (Defeasible Derivation [10])**.** Let $\mathcal{P} = (\Pi, \Delta)$ be a delp. A defeasible derivation of a ground literal L from $\mathcal{P}$, denoted $\mathcal{P} \mid\!\sim L$, consists of a finite sequence $L_1, L_2, \cdots, L_n = L$ of ground literals, and each literal L_i is in the sequence because:

- L_i is a fact in Π, or
- there exists a rule R_i in $\mathcal{P}$ (strict or defeasible) with head L_i and body $B_1, B_2, \cdots, B_k$ and every literal of the body is an element L_j of the sequence appearing before $L_i(j < i)$

In the delp from Example 3.1.1 there is the sequence of literals $c(tina)$, $b(tina)$, $f(tina)$ a defeasible derivation of the literal $f(tina)$, obtained from the following rules: $b(tina) \leftarrow c(tina)$ and $f(tina) \prec b(tina)$. There is also a defeasible derivation of $\sim f(tina)$, for example the sequence $c(tina), \sim f(tina)$.

It is easy to see, that if a delp has no facts, then no defeasible derivation can be obtained. This is possible, because a delp does not have *presumptions*, that represent a defeasible facts. They can be added as an extension to DELP, and we will use them in the next section.

We can define a *strict derivation*, as a derivation, in which only strict rules and facts are used. The literal ${\sim}flies(tweety)$ in Example 3.1.1 has a strict derivation.

Definition 3.1.3 (Contradictory set of rules [10])**.** A set of rules is contradictory if and only if, there exists a defeasible derivation for a pair of complementary literals from this set.

In Example 3.1.1 both $f(tina)$ and $\sim f(tina)$ can be defeasibly derived, so the example shows a contradictory set of rules. Garcia and Simari in [10] declared that in general, a useful defeasible logic program will be a contradictory set of rules. However in Definition 3.1.1 we require a non-contradictory set of facts and rules, otherwise we would obtain every literal.

The symbol $\bar{\ }$ will represent the complement of a literal with respect to the strong negation, i. e. $\overline{p}$ is $\sim p$, and $\overline{\sim p}$ is p.

Definition 3.1.4 (Argument Structure [10])**.** Let h be a literal, and $\mathcal{P} = (\Pi, \Delta)$ a program. We say that $\langle A, h\rangle$ is an argument structure for h, if $A \subseteq \Delta$, such that

1. there exists a defeasible derivation for h from $\Pi \cup A$
2. the set $\Pi \cup A$ is non-contradictory, and
3. A is minimal, there is no proper subset A' of A such that A' satisfies conditions (1) and (2).

Informally, an argument is a minimal and non-contradictory set of rules used to derive a conclusion. In Example 3.1.1 there are two arguments for a literal $f(tina)$: $A_1 = \langle\{flies(tina) \prec bird(tina)\}, flies(tina)\rangle$ and $A_2 = \langle\{flies(tina) \prec chicken(tina), scared(tina)\}, flies(tina)\rangle$.

It is proved in [10] that if there exists a strict derivation for q, then there exists a unique argument structure for $q : \langle\emptyset, q\rangle$. It is unique because of condition 3 in the definition. Observe that if q have a strict derivation and if there exists a defeasible derivation for $\overline{q}$ from $\Pi \cup A$, for some set A then $\Pi \cup A$ will be contradictory. No argument structure for $\overline{q}$ could be obtained. For the literals with a strict derivation, there are unique argument structures without counter arguments.

An argument structure $\langle B, q\rangle$ is a *sub-argument structure* of $\langle A, h\rangle$ if $B \subseteq A$. The union of arguments is not always an argument, because the union could be not minimal or contradictory.

Definition 3.1.5 (Disagreement [10])**.** Let $\mathcal{P} = \Pi \cup \Delta$ be a delp. We say that two literals h_1 and h_2 disagree, if and only if the set $\Pi \cup \{h_1, h_2\}$ is contradictory.

Two complementary literals trivially disagree for any set. Furthermore, two literals that are not complementary can also disagree. For example the literals a, b with $\Pi = \{(\sim h \leftarrow b), (h \leftarrow a)\}$.

Definition 3.1.6 (Counter-argument [10])**.** We say that $\langle A_1, h_1\rangle$ counter-argues, rebuts or attacks $\langle A_2, h_2\rangle$ at literal h, if and only if there exists a sub-argument $\langle A, h\rangle$ of $\langle A_2, h_2\rangle$ such that h and h_1 disagree. h is called counter-argument and $\langle A, h\rangle$ is called the disagreement sub-argument.

From Example 3.1.1, the argument $\langle\{\sim f(tina) \prec c(tina)\}, \sim f(tina)\rangle$ have counter-argument $\langle\{f(tina) \prec b(tina)\}, f(tina)\rangle$ and vice-versa.

We have defined three from five elements of argumentation: a logical language as a logic program, arguments, and conflict between arguments. These three elements are also necessary in Dung's argumentation framework. We can now prepare a set of arguments and a binary relation on them. DELP includes also the other two elements of argumentation. We are now continuing with defeat between arguments. The definition can be based on specificity or on priorities among program rules. We are showing the former one.

Definition 3.1.7 (Specificity [10])**.** Let $\mathcal{P} = (\Pi, \Delta)$ be a delp and let Π_G be the set of all strict rules from Π. Let $\mathcal{F}$ be the set of all literals that have a defeasible derivation from $\mathcal{P}$ ($\mathcal{F}$ will be considered as set of facts). Let $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ be two argument structures obtained from $\mathcal{P}$. $\langle A_1, h_1\rangle$ is strictly more specific than $\langle A_2, h_2\rangle$, if the following conditions hold:

1. For all $H \subseteq \mathcal{F}$: if $\Pi_G \cup H \cup A_1 \mid\!\sim h_1$ and $\Pi_G \cup H \not\vdash h_1$ then $\Pi_G \cup H \cup A_2 \mid\!\sim h_2$ and
2. there exists $H' \subseteq \mathcal{F}$ such that $\Pi_G \cup H' \cup A_2 \mid\!\sim h_2$ and $\Pi_G \cup H' \not\vdash h_2$ and $\Pi_G \cup H' \cup A_1 \mid\!\not\sim h_1$

This notion of specificity favors two aspects in an argument: it prefers an argument with greater information content, or with less use of rules. From Example 3.1.1 $\langle\{\sim f(tina) \prec c(tina)\}, \sim f(tina)\rangle$ is strictly more specific than $\langle\{f(tina) \prec b(tina)\}, f(tina)\rangle$, because it is more direct. And $\langle\{f(tina) \prec c(tina), s(tina)\}, f(tina)\rangle$ is strictly more specific than $\langle\{\sim f(tina) \prec c(tina)\}, \sim f(tina)\rangle$, because it is based on more informations.

We can declare two types of defeater:

Definition 3.1.8 (Proper Defeater [10])**.** Let $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ be two argument structures. $\langle A_1, h_1\rangle$ is a proper defeater for $\langle A_2, h_2\rangle$ at literal h, if and only if there exists a sub-argument $\langle A, h\rangle$ of $\langle A_2, h_2\rangle$ such that $\langle A_1, h_1\rangle$ counter-argues $\langle A_2, h_2\rangle$ at h and $\langle A_1, h_1\rangle > \langle A, h\rangle$

Definition 3.1.9 (Blocking Defeater [10])**.** Let $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ be two argument structures. $\langle A_1, h_1\rangle$ is a blocking defeater for $\langle A_2, h_2\rangle$ at literal h, if and only if there exists a sub-argument $\langle A, h\rangle$ of $\langle A_2, h_2\rangle$ such that $\langle A_1, h_1\rangle$ counter-argues $\langle A_2, h_2\rangle$ at h and $\langle A_1, h_1\rangle$ is unrelated by the preference order to $\langle A, h\rangle$ ($\langle A_1, h_1\rangle \not\succ \langle A, h\rangle$, and $\langle A, h\rangle \not\succ \langle A_1, h_1\rangle$)

Definition 3.1.10 (Defeater [10])**.** The argument structure $\langle A_1, h_1\rangle$ is a defeater for $\langle A_2, h_2\rangle$, if and only if either:

1. $\langle A_1, h_1\rangle$ is a proper defeater for $\langle A_2, h_2\rangle$; or
2. $\langle A_1, h_1\rangle$ is a blocking defeater for $\langle A_2, h_2\rangle$.

In Example 3.1.1 is $\langle\{f(tina) \prec c(tina), s(tina)\}, f(tina)\rangle$ is a proper defeater for $\langle\{\sim f(tina) \prec c(tina)\}, \sim f(tina)\rangle$.

Example 3.1.2. *This Nixon example, which we mentioned also in previous chapter, is famous for introducing reciprocal defeaters:*

$$\Pi = \left\{\begin{array}{l} l\_in\_chicago(nixon) \\ quaker(nixon) \\ republican(nixon) \end{array}\right\} \Delta = \left\{\begin{array}{l} h\_a\_gun(X) \prec l\_in\_chicago(X) \\ \sim h\_a\_gun(X) \prec l\_in\_chicago(X), \\ \hfill pacifist(X) \\ pacifist(X) \prec quaker(X) \\ \sim pacifist(X) \prec republican(X) \end{array}\right\}$$

In 3.1.2 there is $\langle\{pacifist(nixon) \prec quaker(nixon)\}, pacifist(nixon)\rangle$ as a blocking defeater for an argument structure $\langle\{\sim pacifist(nixon) \prec republican(nixon)\}, \sim pacifist(nixon)\rangle$ and vice versa.

For deciding whether an argument structure is non-defeated, all its defeaters have to be considered. Since a defeater is also an argument structure, then its defeaters may exist, and so on. In this manner, a sequence of argument structures is created, where each element of the sequence defeats its predecessor.

Definition 3.1.11 (Argumentation Line [10])**.** Let $\mathcal{P}$ a delp and $\langle A_0, h_0\rangle$ an argument structure obtained from $\mathcal{P}$. An argumentation line for $\langle A_0, h_0\rangle$ is a sequence of argument structures from $\mathcal{P}$, denoted $\Lambda = [\langle A_0, h_0\rangle, \langle A_1, h_1\rangle, \langle A_2, h_2\rangle, \langle A_3, h_3\rangle, \ldots]$, where each element of the sequence $\langle A_i, h_i\rangle, i > 0$ is a defeater of its predecessor $\langle A_{i-1}, h_{i-1}\rangle$

In argumentation theory there are several problematic situations that in our case will arise in an argumentation line leading to an infinite sequence of defeaters.

Firstly, it is a self-defeating argument, when an argument structure is a defeater for itself. Our definition of an argument structure does not allow self-defeating, because $\Pi \cup A$ must be non-contradictory.

Secondly, there are reciprocal defeaters, as we have shown in the nixon example 3.1.2. For reminding, this happens when a pair of arguments defeat each other. This situation leads to the construction of an infinite sequence of arguments, therefore they must be detected and avoided.

Thirdly, a circular argumentation is obtained when an argument structure is reintroduced again in an argumentation line to defend itself. The reciprocal defeaters can be defined as a special case of a circular argumentation. Moreover, a more subtle case of circular argumentation happens with the reintroduction of a sub-argument, that was defeated earlier in the line.

Fourth, when the same argument becomes both a supporting and an interfering argument of itself. This situation arises because the supporting argument has a sub-argument for the literal, which is contradictory with arguing in favor. This should be avoided and it is formally with notion of argument concordance.

Definition 3.1.12 (Supporting and Interfering argument structures [10])**.** Let $\Lambda = [\langle A_0, h_0\rangle, \langle A_1, h_1\rangle, \langle A_2, h_2\rangle, \langle A_3, h_3\rangle, \ldots]$ an argumentation line, we define the set of supporting argument structures $\Lambda_S = [\langle A_0, h_0\rangle, \langle A_2, h_2\rangle, \langle A_4, h_4\rangle, \ldots]$, and the set of interfering argument structures $\Lambda_I = [\langle A_1, h_1\rangle, \langle A_3, h_3\rangle, \ldots]$.

Definition 3.1.13 (Concardance [10])**.** Let $\mathcal{P} = (\Pi, \Delta)$ be a delp. Two arguments $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ are concordant iff the set $\Pi \cup A_1 \cup A_2$ is non-contradictory. More generally, a set of argument structures $\{\langle A_i, h_i\rangle\}_{i=1}^{n}$ is concordant iff $\Pi \cup \bigcup_{i=1}^{n} A_1$ is non-contradictory.

In summary, the undesirable situations mentioned above are avoided by requiring all argumentation lines to be acceptable as defined next.

Definition 3.1.14 (Acceptable argumentation line [10])**.** Let $\Lambda = [\langle A_0, h_0\rangle, \ldots, \langle A_i, h_i\rangle, \ldots, \langle A_n, h_n\rangle]$ be an argumentation line. Λ is an acceptable argumentation line iff:

1. Λ is a finite sequence

2. The set Λ_S of supporting arguments is concordant, and the set Λ_I of interfering arguments is concordant

3. No argument $\langle A_k, h_k \rangle$ in Λ is a sub-argument of an argument $\langle A_i, h_i \rangle$ appearing earlier in Λ ($i < k$)

4. For all i, such that the argument $\langle A_i, h_i \rangle$ is a blocking defeater for $\langle A_{i-1}, h_{i-1} \rangle$ if $\langle A_{i+1}, h_{i+1} \rangle$ exists, then $\langle A_{i+1}, h_{i+1} \rangle$ is a proper defeater for $\langle A_i, h_i \rangle$

In DELP, a literal will be warranted if there exists a non-defeated argument structure. As we stated before, the set of defeater for this argument will be considered and also their defeaters, and so on. Therefore more than one argumentation line could arise, leading to a tree structure that will be defined.

Definition 3.1.15 (Dialectical Tree [10]). Let $\langle A_0, h_0 \rangle$ be an argument structure from a program $\mathcal{P}$. A dialectical tree for $\langle A_0, h_0 \rangle$, denoted $T_{\langle A_0, h_0 \rangle}$, is defined as follows:

1. The root of the tree is labeled with $\langle A_0, h_0 \rangle$

2. Let N be a non-root node of the tree labeled $\langle A_n, h_n \rangle$ and $\Lambda = [\langle A_0, h_0 \rangle, \ldots, \langle A_n, h_n \rangle]$ the sequence of labels of the path from the root to N. Let $\langle B_1, q_1 \rangle, \langle B_2, q_2 \rangle, \ldots, \langle B_k, q_k \rangle$ be all the defeater for $\langle A_n, h_n \rangle$

   For each defeater $\langle B_i, q_i \rangle (1 \leq i \leq k)$ such that, the argumentation line $\Lambda = [\langle A_0, h_0 \rangle, \ldots, \langle A_n, h_n \rangle, \langle B_i, q_i \rangle]$ is acceptable, then the node N has a child N_i labeled $\langle B_i, q_i \rangle$

   If there is no defeater for $\langle A_n, h_n \rangle$ or there is no $\langle B_i, q_i \rangle$ such that Λ' is acceptable, then N is a leaf

Every node, except for the root, represents a defeater of its parent. Each path from the root to a leaf corresponds to one distinct acceptable argumentation line.

In order to decide whether the root of a dialectical tree is defeated, a marking process will be defined. The nodes will be recursively marked as D (defeated) or U (undefeated) as follows:

Definition 3.1.16 (Marking of a dialectical tree [10]). Let $T_{\langle A, h \rangle}$ be a dialectical tree for $\langle A, h \rangle$. The corresponding marked dialectical tree, denoted $T^*_{\langle A, h \rangle}$ will be obtained marking every node in $T_{\langle A, h \rangle}$ as follows:

1. All leaves in $T_{\langle A,h\rangle}$ are marked as U (undefeated) in $T^*_{\langle A,h\rangle}$
2. Let $\langle B,q\rangle$ be an inner node of $T_{\langle A,h\rangle}$.
   (a) Then $\langle B,q\rangle$ will be marked as U in $T^*_{\langle A,h\rangle}$ iff every child of $\langle B,q\rangle$ is marked as D
   (b) The node $\langle B,q\rangle$ will be marked as D (defeated) in $T^*_{\langle A,h\rangle}$ iff it has at least a child marked as U.

This warrant procedure can be optimized with pruning. We can use minimax tree from Artificial Intelligence for game trees.

Example 3.1.3. *An marked dialetical tree for chicken example 3.1.1 is the following tree:*

$$\begin{tikzcd}
\langle\{f(t) \prec b(t)\}, f(t)\rangle & U \\
\langle\{\sim f(t) \prec c(t)\}, \sim f(t)\rangle \arrow[u] & D \\
\langle\{f(t) \prec c(t), c(t)\}, f(t)\rangle \arrow[u] & U
\end{tikzcd}$$

Definition 3.1.17 (Warranted literals [10])**.** Let $\langle A,h\rangle$ be an argument structure and $T^*_{\langle A,h\rangle}$ its associated marked dialectical tree. The literal h is warranted iff the root of $T^*_{\langle A,h\rangle}$ is marked as U. We will say that A is a warrant for h.

We will define a modal operator of belief B where Bh means h is warranted and $\neg Bh$ means h is not warranted.

Definition 3.1.18 (Answer to queries [10])**.** The answers of a DELP interpreter can be defined in terms a modal operator B. In terms of B, there are four possible answers for a query h:

- YES, if Bh (h is warranted)
- NO, if $B\overline{h}$ (the complement of h is warranted)

- UNDECIDED, if $\neg Bh$ and $\neg B \sim h$ (neither h nor $\sim h$ are warranted)
- UNKNOWN, if h is not in the language of the program

The answer for chicken example and query $flies(tina)$ is YES. Of course DELP answers NO for query $\sim flies(tina)$.

An interpreter of DELP was implemented in PROLOG [28], unfortunately with some bugs. If there is an equivalence in a program such as $p \prec q$ and $q \prec p$, it will not compute anything at all.

3.2 Argumentation Blackboard for Coordination

In this section we propose definitions for our framework, ABBC. As we mentioned before, it is distributed extension of DELP [10]. Therefore, the following definitions are inspirited by DELP [10], but they are introduced from our point of view.

A program in DELP contains strict and weak rules. The boundary between these two types of rules is vague. In our framework, there is only one type of rules, but we distinguish the rules from blackboard knowledge base, which will behave as strict rules, and rules from the agent's knowledge base, which will behave as weak rules. Moreover we will not separate facts from rules as it is not separated in Logic Programming [1]. Hence facts will be represented as rules with an empty body.

Consequently, if a fact can be a rule and an agent can use the rule in an argumentation, the argumentation might use weak facts, known as presumptions. DELP in [10] does not compute with presumptions, but the ABBC does. In this point of view, the following definition of a program or knowledge base is proposed.

Definition 3.2.1 (Program)**.** A program is a set of rules. A rule is an ordered pair $\langle H, B \rangle$, where H (Head) is a literal and B (Body) is a finite set of literals. The body can be possibly empty.

A literal was described in the previous section. In ABBC we are using the same logic programming syntax as in DELP [10], except we do not need a symbol for all weak rules. We can now formally describe an ABBC.

Definition 3.2.2 (Argumentation Black Board for Coordination)**.** Let $G \subseteq \{1, \ldots, n\}$ be a group of agents, KB_{BB} be a program, $\prec$ be a binary relation on agents and q be a query consisting of one literal. We say that a tuple $BB = \langle KB_{BB}, G, \prec, q \rangle$ is an Argumentation Black Board for Coordination. It is assumed that $\forall k \in G \quad KB_{BB} \subseteq KB_k$ and the set KB_{BB} is non-contradictory.

ABBC's knowledge base KB_{BB} must be a subset of agent's knowledge base, if the agent want to participate in an argumentation. The agent should add the missing rules from KB_{BB}, but it is important that its knowledge base will not become a contradictory set of rules. Therefore KB_{BB} is a set of rules, that any participating agent's knowledge base do not contradict. A binary relation $\prec$ represents a priority among agents, but it can be empty and in this case, the agents will be equal.

Example 3.2.1. *We introduce an example for ABBC, that was transformed from [13], where it was represented to show a problem with updates in logic programming.*

$$\begin{array}{ll} KB_{BB} = \left\{ \begin{array}{l} sleep \leftarrow \sim tv\_on \\ watch\_tv \leftarrow tv\_on \end{array} \right\} & 1 = \left\{ \begin{array}{l} \sim tv\_on \leftarrow power\_failure \\ power\_failure \leftarrow true \end{array} \right\} \\ 2 \prec 1 & \\ q = watch\_tv & 2 = \left\{ \begin{array}{l} tv\_on \leftarrow in\_evening \\ in\_evening \leftarrow true \end{array} \right\} \end{array}$$

We have a KB_{BB} about behaviours of people in the evening. The first agent is a power maintenance agent and it knows, when something is plugged. The second agent is a home agent, who knows a behaviour of its owners. Of course the first agent information is more relevant according to the priority relation $\prec$. The query is watch_tv.

We expect that the agent's knowledge base will be unknown for the ABBC. The ABBC will obtain only some messages with requests, or with an argument for participating in argumentation. An argument has the same structure as in 3.1.4.

Notation 3.2.1 (Argument)**.** Let h be a literal, A be an set of rules and $i \in G$ be an agent. Then $\langle A, h, i \rangle$ denotes an argument for h from agent i.

The ABBC will have to decide for each argument $\langle A, h, i \rangle$ if the argument is correct according to the ABBC knowledge base KB_{BB} and the set of rules A. We require the same criteria as DELP, i.e. $A \cup KB_{BB}$ is non-contradictory,

A is minimal and there is a derivation of h using $A \cup KB_{BB}$ as a set of rules. By modification of definition of derivation and argument structure from [10], we get the following definitions:

Definition 3.2.3 (Derivation)**.** Let $BB = \langle KB_{BB}, G, \prec, q\rangle$ be an ABBC. A derivation of a ground literal L from BB and some set A of rules, denoted $KB_{BB} \cup A \mid\!\sim L$, consists of a finite sequence $L_1, L_2, \cdots, L_n = L$ of ground literals, and each literal L_i is in the sequence because:

- L_i is a fact (a rule with empty body) in $KB_{BB} \cup A$, or
- there exists a rule R_i in $KB_{BB} \cup A$ with head L_i and body $B_1, B_2, \cdots, B_k$ and every literal of the body is an element L_j of the sequence appearing before $L_i (j < i)$.

Definition 3.2.4 (Acceptable argument)**.** Let $\alpha = \langle A, h, i\rangle$ be an argument and $BB = \langle KB_{BB}, G, \prec, q\rangle$ be an ABBC. We say that α is an acceptable argument in BB iff:

1. there exists a derivation for h from $KB_{BB} \cup A$
2. the set $KB_{BB} \cup A$ is non-contradictory, and
3. A is minimal, there is no proper subset A' of A such that A' satisfies conditions (1) and (2).

In example 3.2.1 an acceptable argument is shown $\langle\{\sim\!tv\_on \leftarrow power\_failure, power\_failure \leftarrow true\}, \sim\!tv\_on, 1\rangle$ from agent 1 and $\langle\{tv\_on \leftarrow in\_evening, in\_evening \leftarrow true\}, watch\_tv, 2\rangle$ from agent 2.

The ABBC will be empty in the beginning and it will accept only arguments supporting query q, or complementary literal $\overline{q}$. In this thesis we will call them base arguments.

Definition 3.2.5 (Base argument)**.** Let $\alpha = \langle A, h, i\rangle$ be an acceptable argument and $BB = \langle KB_{BB}, G, \prec, q\rangle$ be an ABBC. We say that α is an base argument for q ($\overline{q}$) in BB iff

1. $h = q$ ($h = \overline{q}$) and
2. there is no argument $\beta = \langle B, f, j\rangle$, where $B = A$, used as base argument for q ($\overline{q}$), or β is not used as an supporting argument in the dialectical tree for q ($\overline{q}$).

In Example 3.2.1, the acceptable argument $\langle\{tv\_on \leftarrow in\_evening, in\_evening \leftarrow true\}, watch\_tv, 2\rangle$ from agent 2 is a base argument, because the query is $watch\_tv$.

There can be many different base arguments for a literal and they will become roots of trees. Afterwards, agents are allowed to send a counter-argument defined in 3.1.6. These counter-arguments will be accepted, if they satisfy the conditions from 3.2.8.

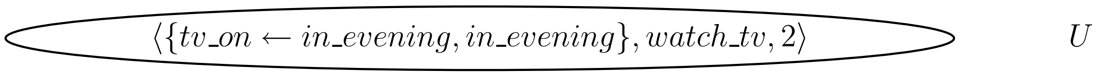
During whole ABBC process we will work with a marked dialectical forest, which is a modification of 3.1.15 and 3.1.16, with the following properties:

Definition 3.2.6 (Marked Dialectical tree)**.** Let $\langle A_0, h_0, i_0\rangle$ be an argument. A marked dialectical tree for $\langle A_0, h_0, i_0\rangle$ is defined as follows:

1. The root of the tree is labeled with $\langle A_0, h_0, i_0\rangle$.
2. Each tree path from the root to the leaf is an acceptable argumentation line.
3. All leaves are marked as U (undefeated).
4. Let $\langle B, q, i\rangle$ be an inner node.
   (a) Then $\langle B, q, i\rangle$ will be marked as U iff every child of $\langle B, q, i\rangle$ is marked as D (defeated).
   (b) The node $\langle B, q, i\rangle$ will be marked as D iff it has at least a child marked as U.

Definition 3.2.7 (Marked dialectical forest)**.** Let q be a query consisting of one literal. A marked dialectical forest F for q is a disjoint union of marked dialectical trees where the root of each dialectical tree is labeled with a base argument $\langle A_0, h_0, i_0\rangle$ for q or $\overline{q}$.

Example 3.2.2. *A marked dialetical forest for Example 3.2.1 is for instance the following tree:*


$\langle\{tv\_on \leftarrow in\_evening, in\_evening\}, watch\_tv, 2\rangle$
U

Finally, we can define acceptable defeaters, which are the key objects in the ABBC, together with base arguments. Only these two components can change the outcome of an argumentation in ABBC.

Definition 3.2.8 (Acceptable Defeater)**.** Let $\alpha = \langle A, h, i\rangle$ be an acceptable argument and F be a dialectical forest for a query q. We say that α is the defeater for F, iff

1. there exists an undefeated node $\beta = \langle A_j, h_j, i_j\rangle$ in F, where α is a blocking defeater for β or α is a proper defeater of β, and
2. the sequence from root of the tree to β and α $[\langle A_0, h_0, i_0\rangle, \ldots, \langle A_j, h_j, i_j\rangle, \langle A, h, i\rangle]$, is an acceptable argumentation line.

Example 3.2.3. *In Example 3.2.1 is shown the acceptable argument* $\langle\{\sim tv\_on \leftarrow power\_failure, power\_failure \leftarrow true\}, \sim tv\_on, 1\rangle$ *from agent* 1 *and an acceptable defeater for the argument* $\langle\{tv\_on \leftarrow in\_evening, in\_evening \leftarrow true\}, watch\_tv, 2\rangle$ *from agent* 2*. The marked dialectical forest for tv example can be following:*

$$\begin{tikzcd} \langle\{tv\_on \leftarrow in\_even, in\_even\}, watch\_tv, 2\rangle & D \\ \langle\{\sim tv\_on \leftarrow power\_fail, power\_fail\}, \sim tv\_on, 1\rangle \arrow[u] & U \end{tikzcd}$$

The acceptable defeater will be added to the marked dialectical tree as the successor of the nodes it defeats. The ABBC will change marks of the arguments in a marked dialectical forest (MDF) according to the definition of MDF. The query is validated during the whole process. It starts as UNDECIDED and changes according the following definition. The warrant is defined in 3.1.17.

Definition 3.2.9 (Answers to queries)**.** The answers of an ABBC interpreter can be

- YES, if q is warranted and $\overline{q}$ is not warranted,
- NO, if $\overline{q}$ is warranted and q is not warranted,

- UNDECIDED, otherwise.

In Example 3.2.3, the answer for query *watch_tv* and this marked dialectical forest is UNDECIDED. However the agent 1 can decide that there is not $power_{f}ailure$ and it can send a defeater for his last argument. This can be useful to extend an ABBC with default negation, to have rules such as $\sim watch\_tv \leftarrow not \quad watch\_tv$. A default negation, *not* F, is assumed by default, i.e., it is assumed in the absence of sufficient evidence to the contrary. In DELP, it means that there is no warrant as discussed in [10]. We plan to extend ABBC with default negation, but an extended DELP, which contains the default negation, is in preparation too.

Chapter 4

Abstract ABBC

In this chapter, we will define an Abstract ABBC, a framework which is more abstract than ABBC, but less abstract than the Dung's framework [6]. It was inspirated by the fact that we could change some definitions of ABBC to obtain a different behaviour. The Abstract ABBC abstracts from the details of syntax, inference rules and some other details. The framework will consist only of the necessary parts of argumentation, which will be discussed as argumentation elements, as it was mentioned in section 2.2.

The first argument element is *a logical language*. We want to stay in logic programming, but we want to provide possibility of a choice among different kind of logic programming programs such as definite logic program [1], DELP language [10], or disjunctive logic programs [15]. A common alphabet for all logic languages may be defined

Definition 4.0.10 (Alphabet of Abstract ABBC)**.** An alphabet $\mathcal{A}$ of an Abstract ABBC is a disjoint set of constants, variables, predicate symbols with associated arity, and function symbols with associated arity.

An Abstract ABBC can be used with many different languages. They must be defined by a grammar and can use punctuation symbols and logical symbols, such as conjunction and disjunction. With this definition, each logic programming language is a language of Abstract ABBC.

Definition 4.0.11 (Language of Abstract ABBC)**.** Let $\mathcal{A}$ be an alphabet of an Abstract ABBC and G be a grammar over $\mathcal{A}$ and alphabet of logical symbols and punctuation symbols. We say that a language L is a language of abstract ABBC if $L = L(G)$.

A query is a literal in DELP and ABBC, and it is a sequence of literals in Prolog as well. By this way we will have the possibility to define queries as we would like.

Notation 4.0.2 (Query language)**.** A Query language L_Q is a language of Abstract ABBC.

Example 4.0.4. *In this example, we will show how ABBC language can be defined such a language of Abstract ABBC. ABBC has following an alphabet of logical and the punctuation symbols $\mathcal{A}_S = \{\sim, \leftarrow\} \cup \{,\}$. Let $\mathcal{A}$ be an alphabet of Abstract ABBC, C be a set of constants from $\mathcal{A}$, V be a set of variables from $\mathcal{A}$. We define the corresponding grammar for ABBC as follows:*

1. *Set of terms over $\mathcal{A}$ is $T = C \cup V \cup \{f(t_1, \ldots, t_n)\}$, where f is function symbol from $\mathcal{A}$ and $t_i \in T$, is a set of terms over $\mathcal{A}$.*
2. *$A = \{p(t_1, \ldots, t_n)\}$, where p is predicate symbol from $\mathcal{A}$ and $t_i \in T$, is a set of atoms over $\mathcal{A}$.*
3. *$I = A \cup \{\sim a\}$, where $a \in A$, is a set of literals over $\mathcal{A}$.*
4. *$H = I$, is a set of heads over $\mathcal{A}$.*
5. *$B = \{true\} \cup \{i_1, \ldots, i_n\}$, where $i_1, \ldots, i_n$ are literals, is a set of bodies over $\mathcal{A}$.*
6. *$R = \{h \leftarrow b\}$, where $h \in H, b \in B$, is a set of rules over $\mathcal{A}$.*

The language of ABBC is same as a language of Abstract ABBC, defined as the set of rules. Queries in ABBC have the same structure as queries from a query language, which is defined as the set of literals.

We expect that a language is usually defined as a set of clauses, therefore the definition of a program follows:

Definition 4.0.12 (Program)**.** Let L be a language of Abstract ABBC, we say that finite subset of L is a program of language L.

We can now define Abstract ABBC, that can be defined in each language, and also the consequence operator.

Definition 4.0.13 (Non-monotonic consequence operator)**.** Let L be a language of an Abstract ABBC. We say that a function $Cn : \mathcal{P}(L) \rightarrow \mathcal{P}(L)$ is a non-monotonic consequence operator, if for every $E \subseteq L$ is $Cn(Cn(E)) \subseteq Cn(E)$.

In ABBC a non-monotonic consequence operator is defined by a defeasible derivation.

Definition 4.0.14 (Abstract Argumentation Black Board for Coordination)**.** Let L be a language of Abstract ABBC, L_Q be a query language, Cn be a non-monotonic consequence operator and let $<$ (a criterion on arguments), ξ (an acceptance function), S (a semantic function) be a defined below in 4.0.16, 4.0.18 and 4.0.19. Let $G \subseteq \{1, \ldots, n\}$ be a group of agents, KB_{BB} be a program of L, $\prec$ be a binary relation on agents, $q \in L_Q$. We say that a tuple $BB = \langle L, L_Q, Cn, <, \xi, S, KB_{BB}, G, \prec, q \rangle$ is an Abstract Argumentation Black Board for Coordination. It is assumed that $\forall k \in G \quad KB_{BB} \subseteq KB_k$ and the set KB_{BB} is non-contradictory.

Note that in Dung's argumentation framework [6], the concept of the *argument* is not defined. In many argumentation systems, arguments have similar structure as a set of logical formulae and the sentence which can be proved from them. In the following, we shall use the following definition:

Notation 4.0.3 (Argument)**.** Let $BB = \langle L, L_Q, Cn, <, \xi, S, KB_{BB}, G, \prec, q \rangle$ be an Abstract ABBC, $h \in L_Q$ and A be a program of L, $i \in G$ is agent. Then $\langle A, h, i \rangle$ is an argument for h from agent i

Definition 4.0.15 (Acceptable argument)**.** Let $\alpha = \langle A, h, i \rangle$ be an argument and $BB = \langle L, L_Q, Cn, <, \xi, S, KB_{BB}, G, \prec, q \rangle$ be an Abstract ABBC. We say that α is an acceptable argument in BB iff:

1. $h \in Cn(KB_{BB} \cup A)$
2. the set $KB_{BB} \cup A$ is non-contradictory, and
3. A is minimal, there is no proper subset A' of A such that A' satisfies conditions (1) and (2).

There are three types of *conflict between arguments* as described in section 2.2. We will use only two of them: rebutting attack and assumption attack, which are defined in the definition of counter argument in 3.1.6.

We do not need the undercutting attack, which is an attack on a rule of inference, because in each argument, the same inference rule is used.

A *Defeat between arguments* similarly to DELP is defined by specificity or by domain-specific criterion. We will define it abstractly as a relation.

Definition 4.0.16 (Criterion for arguments)**.** Let L be a language of abstract ABBC, L_Q be a query language, $\prec$ is a binary relation on agents. We say that a relation $<$ on $2^L \times L_Q \times N \times 2^L \times \prec$ is a criterion for arguments. We say that an acceptable argument $\langle A_1, H_1, i_1 \rangle$ is more specific than $\langle A_2, h_2, i_2 \rangle$ if $\langle A_1, h_1, i_1, KB_{BB}, \prec \rangle < \langle A_2, h_2, i_2, KB_{BB}, \prec \rangle$ and $\langle A_2, h_2, i_2, KB_{BB}, \prec \rangle \not< \langle A_1, h_1, i_1, KB_{BB}, \prec \rangle$

We will define an argumentation graph, which will represent the status of an argumentation. In ABBC we used a marked dialectical tree, because of the limitation for an accepting argument. Such a graph will have two types of edges, one for blocking defeaters and second for proper defeaters.

Definition 4.0.17 (Argumentation Graph)**.** Let A be a set of arguments and V, W be relations on $A \times A$. We say that a tuple $G = \langle A, V, W \rangle$ is an argumenation graph, if $\alpha \in A$ counter-argues $\beta (\in A)$ iff $(\alpha, \beta) \in V$ and moreover α is more specific than β, iff $(\alpha, \beta) \in W$.

Definition 4.0.18 (Acceptance function)**.** Let $G = (A, V, W)$ be an argumentation graph and α be an acceptable argument. An acceptance function ξ is a function $\xi : \Sigma \times \mathcal{A} \to \{0, 1\}$, where $\mathcal{A}$ is a set of all arguments and Σ is a set of all paths. ξ for every argument path λ in a graph decides: an argument α can be added, iff $\xi(\lambda, \alpha) = 1$.

As the *Dialectical status of arguments*, the user may choose from one of Dung's semantics. They can be represented as a function on an argumentation graph.

Definition 4.0.19 (Semantic function)**.** Let $G = (A, V, W)$ be an argumentation graph. We say that a function $S : G \to \mathcal{P}(\mathcal{A})$, where $\mathcal{A}$ is a set of all arguments, is semantic function. $S(G)$ will produce a set of sets of warranted arguments.

Then we can define answer to queries as follows.

Definition 4.0.20 (Answer to queries)**.** Let $BB = \langle L, L_Q, Cn, <, \xi, S, KB_{BB}, G, \prec , q \rangle$ be an Abstract ABBC and $G = (A, V, W)$ be an argumentation graph. The answers of an ABBC interpreter can be

- skeptically YES, if q is in all $\mathcal{A} \in S(G)$
- credulously YES, if there exists at least one $\mathcal{A} \in S(G)$, which contains q
- skeptically NO, if $\overline{q}$ is in all $\mathcal{A} \in S(G)$
- credulously NO, if there exists at least one $\mathcal{A} \in S(G)$, which contains $\overline{q}$
- UNDECIDED, otherwise

4.1 Software design of Abstract ABBC

In this section, we will introduce a software design for Abstract ABBC. In this thesis, we are interested only in the theoretical properties of coordination and argumentation. Therefore, our description of the software design will be brief without concrete implementation details.

Abstract ABBC, as a tool for coordination among logical agents, can be implemented in various multi-agent systems. The most popular and widespread software multi-agent technology platform available today is Java Agent DEvelopment Framework (JADE) [2]. JADE is written completely in Java, therefore it can benefit from the huge set of language features and third-party libraries. Containing the standard FIPA-ACL communication language, it provides the best opportunity to implement the Abstract ABBC. In the following, we will discuss sending and receiving FIPA ACL messages, an agent life-cycle, which are all implemented as the basic services and infrastructure in JADE.

Abstract ABBC is defined as universal as it is possible. We do not expect that the user to define all properties of the Abstract ABBC in each its recall of the Abstract ABBC. Therefore, we assume that there will be a few prototypes of Abstract ABBC implemented, in our case we will discuss ABBC as a prototype of Abstract ABBC. So that logic agents will point to prototype of Abstract ABBC.

Additionally a service will be created in JADE for creating the Abstract ABBCs as new agents. The logic agents in open multi-agent system will have a choice to ask this service for creating an Abstract ABBC any time during their life-cycle. A logic agent will send a FIPA ACL message similar to the one in example 2.3.1. This message will contain a request performative and a content with a query, the prototype of Abstract ABBC and a set of rules. Then the set of rules will become knowledge base of the Abstract ABBC. The service will check consistency of rules through a consistency module and inform through a FIPA ACL message about accepting or refusing the creation of the Abstract ABBC. The newly opened Abstract ABBC will broadcast a FIPA ACL inform message about its presence to all agents.

We do not need to go into the details of software design of the logical agents as this is left to the users of the Abstract ABBC. We assume that their creators prepared them for making arguments, requesting to create an ABBC, or possibly managing coordination. We do not design the whole multi-agent system, only a coordination tool as an Abstract ABBC.

Abstract ABBC's functional requirements, which will be communicated through FIPA ACL messages, are :

- sending responses for agent's requests such as:
  - dialetical status of the argumentation,
  - rules in knowledge base,
  - list of undefeated arguments,
  - list of agents involved in the argumentation.
- managing arguments that the Abstract ABBC received from logical agents.

As it was already mentioned above, Abstract ABBC consists of a graph structure to capture the whole argumentation about the query. Each node in the structure will embody an argument defined in 4.0.3 and the nodes will be connected according to the defeat relation. The graph structure can vary according to the type of the Abstract ABBC. For example the ABBC's graph structure is a forest, i.e. a disjoint union of trees, which can be implemented in faster data structures than a graph. Also, generally in the Abstract ABBC, we need to have a module to compute the dialetical status of the argumentation from the entire graph, when it is requested. In ABBC type, this dialectical status is computed only through exploring the marks of roots of the trees, which represent undefeated or defeated status of the query.

The similar situation happens when we design the response to a request for a list of undefeated arguments. Generally, this is a method on the entire argumentation structure, which needs to evaluate arguments at first and then search for an undefeated one. In ABBC type, the forest structure only needs to search, using for example depth-first search (DFS), all trees for undefeated arguments, as they are marked in the tree.

If the agent is submitting an argument that becomes accepted, an Abstract ABBC will add an agent's ID to the list of agents. This list can be represented as an array, where each item represents an agent's ID and the time when its last argument was accepted. This list can be replied when a logical agent requests a list of agents involved in the argumentation.

Managing the arguments is one of the most important functionalities of the Abstract ABBC. When an Abstract ABBC receives an argument from a logical agent, it must check its acceptability. In general, an acceptable argument must be added to each path in the argumentation graph and the Abstract ABBC must check the validity of this path according to the defined rules. It must be, however, optimized for each type of the Abstract ABBC as it is in the corresponding ABBC. The ABBC prototype will compare the argument with all undefeated arguments in its argumentation structure. If the argument becomes the acceptable defeater, it will be added to the marked dialectical tree in the ABBC type as the successor of the nodes, which the argument defeats. The marks of the arguments in ABBC argumentation structure will be changed only by traversing through the parents of the nodes from the added defeater to root of the tree.

Chapter 5

Related Work and Discussion

In [23], Matthias Thimm and Gabriele Kern-Isberner proposed a distributed framework that enables DELP. Their main goal is to model a distributed argumentation for common conclusion of a group of agents. As an exemplary application, they described a jury court where every juror has a personal opinion about the guilt or innocence of the accused person.

They used a centralized approach of organisational structuring, in opposition to our open architecture. There is an interface of the system and an contact for queries, called moderator, that coordinates the argumentation process and analyzes the answer to the given query. Although it is a distributed system,a s the knowledge base is separated to global and several local knowledge bases, their system is not as much distributed and open as ours. It is more a distribution of rules to modules than an open multi-agent system.

An extension of an abstract argumentation framework [6] was developed by Srdjan Vesic in [25]. This two-step argumentation process, called a general argumentation framework for decision making, aims to prepare a framework for decision making among agents. It, however, remained in abstract argumentation and it does not specify the structure of arguments, and the defeat and conflict relations are defined only as given. The agents discuss options in their decision and argue about them. This is made by the same process as in the abstract argumentation framework. We believe that this can be processed in Abstract ABBC as one type of coordination. Our system is more specific in argumentation and it can be used for many types of coordination.

These works should be reimplemented in approach of ABBCs.

Chapter 6

Conclusion and Future Work

The ABBC extends the defeasible logic Programming (DELP) [10] and provides possibility of argumentation among logical agents. The argumentation is an important tool for model coordination, therefore the ABBC as the argumentation framework is an appropriate tool for agent coordination in multi-agent system.

The Abstract ABBC specifies some yet unspecified elements in the abstract argumentation framework [6] and enables argumentation in all possible non-monotonic languages with any inference operator. It consists only of the most necessary and universal elements of argumentation. In future it can be used for experiments with argumentation parameters, that may lead to new insights into logic-based argumentation and disclosures new application areas. The other extensions of ABBC, or types of the Abstract ABBC should be developed. We mentioned a possible extension with a default negation.

In this thesis, we studied various theories and models for coordination and argumentation among the logical agents and as a result, we proposed two theoretical frameworks, ABBC and Abstract ABBC. They would need to be studied by the means of complexity theory to optimize their processes. Afterwards they need to be implemented. With an existing implementation, coordination methods based on ABBC and Abstract ABBC tools should be proposed.

ABBC and Abstract ABBC should be implemented firstly in JADE [2], a software framework for multi-agent systems, because JADE complies with the FIPA specifications, which describe the FIPA ACL for communication. It is also the most widespread framework in multi-agent community.

Bibliography

[1] Baral C. and Gelfond M.: *Logic programming and knowledge representation* Journal of Logic Programming, 1994, Vol. 19, 73-148.

[2] Bellifemine F., Caire G., Greenwood D.: *Developing MultiAgent Systems with JADE*, 2007

[3] Bergenti F., Ricci A.: *Three Approaches to the Coordination of Multiagent Systems*, 2002

[4] Caminada M., Amgoud L.: *On the evaluation of argumentation formalisms*, 2007

[5] Denti E., Omicini A.: *Engineering Multi-Agent Systems in LuCe*, 1999

[6] Dung P.M.: *On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-persons games*, 1995

[7] Endriss U., Maudet N., Sadri F., Toni F.: *Logic-based Agent Communication Protocols*, 2005

[8] Finin T., Fritzson T., McKay D., and McEntire R.: *KQML as an agent communication language*, 1994

[9] FIPA. *FIPA 97 specification part 2: Agent communication*, 1998.

[10] Garcia A. J., Simari G. R.: *Defeasible Logic Programming - An Argumentative Approach*, 2004

[11] Huhns M. N., Stephens L. M.: *Multiagent Systems and Societies of Agents* in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence edited by Weiss G., 1999, pp. 79-120

[12] Labrou Y. and Finin T.: *Semantics for an agent communication language*, 1998

[13] Leite J.A.: *Evolving knowledge bases*, 2003

[14] Malone T., Crowston K.: *The Interdisciplinary Study of Coordination*, 1994

[15] Minker J., Seipel D.: *Disjunctive Logic Programming: A Survey And Assessment* 2002

[16] McBurney P., Hitchcock D., Parsons S.: *The eightfold way of deliberation dialogue*, 2004

[17] Papadopoulos G. A., Arbab F.: *Coordination models and languages*, 1998

[18] Parsons S., Sierra C., Jennings N.: *Agents that reason and negotiate by arguing*, 1998

[19] Prakken H., Vreeswijk G.: *Logics for defeasible argumentation*, 2002

[20] Sadek S. D.: *Dialogue acts are rational plans*, 1991

[21] Simari, G. R., Loui R. P.: *A Mathematical Treatment of Defeasible Reasoning and its Implementation.* Artificial intelligence, 1992, Vol. 53, 125–157.

[22] Singh M. P.: *Agent Communication Languages: Rethinking the Principles*, 1998

[23] Thimm M., Kern-Isberner G.: *A Distributed Argumentation Framework using Defeasible Logic Programming (Extended Version)*, 2008

[24] Vengerov D.: *Adaptive Communication and Coordination in Multi agent systems*, 2000

[25] Vesic S.: *An abstract argumentation framework for decision making*, 2008

[26] Walton D. N., Krabbe E. C. W.: *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*, 1995.

[27] Wooldridge M. J.: *An introduction to Multi-agent systems*, Wiley, 2001

[28] http://cs.uns.edu.ar/∼ ajg/DeLP.html