Argumentative Systems in Robots

MASTER DISSERTATION

Cláudio Rodrigo Santos Gonçalves
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

September | 2012

Argumentative Systems in Robots

MASTER DISSERTATION

Cláudio Rodrigo Santos Gonçalves
MASTER IN INFORMATICS ENGINEERING

ORIENTATION
Eduardo Leopoldo Fermé

ABSTRACT

Nowadays, the development of intelligent agents intends to be more refined, using improved architectures and reasoning mechanisms. Revise the beliefs of an agent is also an important subject, due to the consistency that agents should have about their knowledge.

In this work we propose deliberative and argumentative agents using Lego Mindstorms robots, *Argumentative NXT BDI-like Agents*. These agents are built using the notions of the BDI model and they are capable to reason using the DeLP formalism. They update their knowledge base with their perceptions and revise it when necessary. Two variations are presented: the *Single Argumentative NXT BDI-like Agent* and the *MAS Argumentative NXT BDI-like Agent*.

Keywords: Argumentation, Belief Revision, BDI, DeLP, DeLP-server, DeLP-client, Multi-agent System

ACKNOWLEDGEMENTS

First, I would like to thank my parents, sister and girlfriend for all the given love and support.

I would also like to thank my supervisor, Professor Eduardo Fermé (PhD), for his guidance during this work, Professor Luís Gaspar for encouraging me to choose this topic and, in addition, Professor Guillermo Simari (PhD) for providing me the DeLP-Server.

Finally, to all my friends that in some way gave me their incentive to finish this work. Special thanks to Donna Nabower, who revised this master dissertation.

TABLE OF CONTENTS

LIST OF FIGURES

ACRONYMS

AI – Artificial Intelligence

AS – Argumentative Systems

DBR – Data-oriented Belief Revision

BDI – Belief-Desire-Intention

BR – Belief Revision

DAI – Distributed Artificial Intelligence

DeLP – Defeasible Logic Programming

de.l.p. – defeasible logic program

MAS – Multi-agent System

TMS – Truth Maintenance Systems

1. INTRODUCTION

The robots (or agents) are “instruments” increasingly used to serve the purposes of human beings. Across the years, science has been concerned to develop them further and improve their capabilities, making them increasingly autonomous, self-sufficient and intelligent. A good example is the recent robot developed by NASA, named *Curiosity*, which was sent to Mars. A very important component of the agents is the Artificial Intelligence, because it is what will define its behavior. Depending on the purpose, the agents should have implemented algorithms that help them to achieve their goals. Different contexts can be considered: game theory, search, problem solving, etc. Regardless of context, the better the algorithms, the better the agents. They will be more autonomous in their decisions and behaviors. As we can see, the Artificial Intelligence plays a crucial role in the development of agents.

But, are the agents really capable to think? Probably not because that act implies generating something from nothing, however, probably the agents can be capable to reason, because that act implies to generate conclusions from something. This means that if an agent has some knowledge, then it can reach conclusions from that knowledge. This argumentative aspect that agents can have has been studied in the last years. It provides very important mechanisms to help agents to reason about their knowledge and also to argue with other agents.

Another important aspect related to the agents’ knowledge is about the revision of their beliefs. When an agent receives new information, it is important the existence of a process that changes the agent’s beliefs, if necessary, in order to maintain the consistence of its epistemic state. Such process is called Belief Revision.

In this project we will explore the concepts of Argumentation and Belief Revision using the notions of a promising architecture based on the Belief-Desire-Intention model. This architecture is used in the development of intelligent agents and has been widely used and studied. Argumentative NXT BDI-like Agents will be proposed. They are deliberative agents with reasoning capabilities. Two variations will be considered, the Single Argumentative NXT BDI-like Agents and the MAS Argumentative NXT BDI-like Agents. These agents were developed using the Lego Mindstrorms NXT robots.

In this section, we will cover some basic topics of Artificial Intelligence. Section 2 is dedicated to Argumentation, and a case study of an argumentative formalism is presented. In section 3 the Belief Revision subject is addressed. Section 4 covers the existent links between Argumentation and Belief Revision. In section 5 the BDI model is presented as well as the underlying architecture. Section 6 presents the Argumentative NXT BDI-like Agent concept and its architecture. In section 7 it is described the development and implementation of the concept presented in the previous section. Section 8 shows some application cases for the Argumentative NXT BDI-like Agents. In section 9 we discuss and evaluate the proposed concept.

The motivation in developing this project lies in the desire to learn more about the development of intelligent agents. We hope that this project can be applied in more cases, since it is easy to use the developed agents.

1.1. Artificial Intelligence Basics

McCarthy in [1], describe Artificial Intelligence (AI) as "the science and engineering of making intelligent machines". An applicant definition says that AI is the study and design of intelligent agents, [2]. These definitions give us a pretty look on AI and its purpose, but to build an intelligent machine many complex aspects must be considered. So, it's reasonable to say that AI can be decomposed in many different fields, such as reasoning, communication, etc. Here at this subsection we aim to do some clarification about basic notions used at AI.

1.1.1. Agent

First of all lets clarify the "agent" term, because can have different meanings according to the research area that we are talking about. An applicant definition is the one from [2], that an agent is something that collects information and/or knowledge from the world through sensors and act in the environment through actuators. A representation of a general agent is shown at Figure 1.

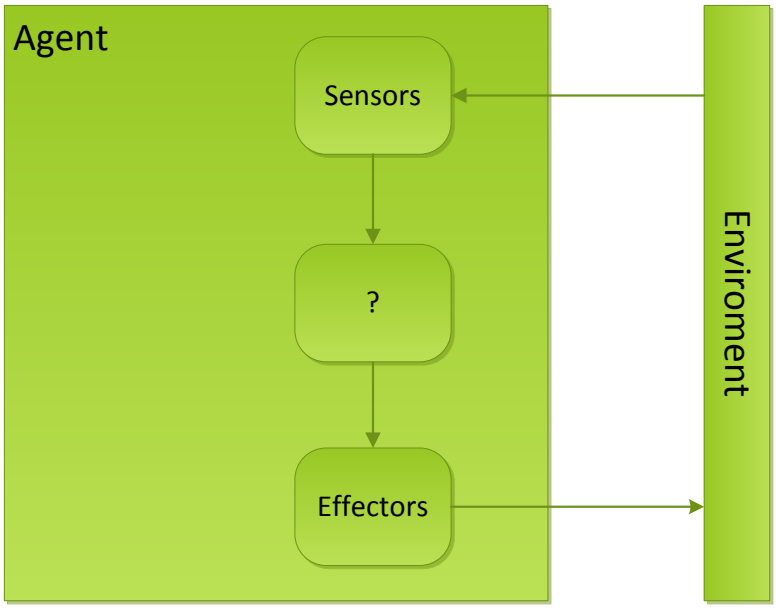


FIGURE 1 - GENERAL AGENT

The interrogation means the decision point of the agent. How and why the agent will act? This question is answered depending on the agent's purpose. Four basic types of agents are presented in [2]: simple reflex agents; reflex agents with state; goal-based agents; utility-based agents.

Another well-known definition was presented by [3], in which an agent is defined as hardware or software piece, and it can have some properties associated with.

1.1.2. PROPERTIES

In [3], the term agent was distinguished in two points. The first is a weak notion of agent, and more consensual. Second is a stronger notion of agent, which can be more controversial.

The weak notion of agent is wider, since it can be seen as a generalization of the term itself. This notion has the following properties:

- Autonomy: agents can work without external or human intervention, and have some control over their actions and internal states.
- Social ability: agents interact with each other (and eventually with humans) using some protocol,
- Reactivity: agents perceive the universe's events and react to them.
- Pro-activeness: although the agents' reactivity, they are capable to act according with their goals.

The strong agent notion came from the AI research field, where it was the intention to give the agents some human and mental capacities. To go for this stronger notion, the following properties must be added:

- **Knowledge**: the capacity to collect information in order to model the agents' world.
- **Belief**: the sentences in which the agents belief and trust.
- **Intention**: actions that the agents plans to perform in the future.
- **Obligation**: commitments that the agents take in the past and which therefore feels obliged to accomplish.
- **Emotions**: an inherent characteristic of the human being that has been attempted to implement in agents.

Some other properties can be used to characterize agents, in particular:

- **Mobility**: agents' ability to move around.
- **Veracity**: agents will not communicate false information deliberately.
- **Benevolence**: the agents can take goals of other agents, as their goals. Since those goals do not conflict with their own.
- **Rationality**: the agents will always act in order to achieve its goals.

1.1.3. ARCHITECTURES

The architecture of an agent, as stated in [4], is the methodology of its construction. From the architecture it is possible to understand how the agent can be decomposed in modules and how these modules are connected with each other. This way we can comprehend how the input data collected by the sensors will be treated and how the actions will be taken.

An agent can be constructed under different architectures depending on the purpose that it is vacationed. Three main agent's architectures, ahead addressed, are exposed in [3]: deliberative, reactive and hybrid.

1.1.3.1. Deliberative Architecture

This architecture is based on a symbolic modeling of the agent's universe (world), i.e., its decisions are based on the symbol's manipulation. Main issue has to do with the difficulty on representing real world information in a model way. Figure 2, based on [5], demonstrate a graphic representation of this architecture.

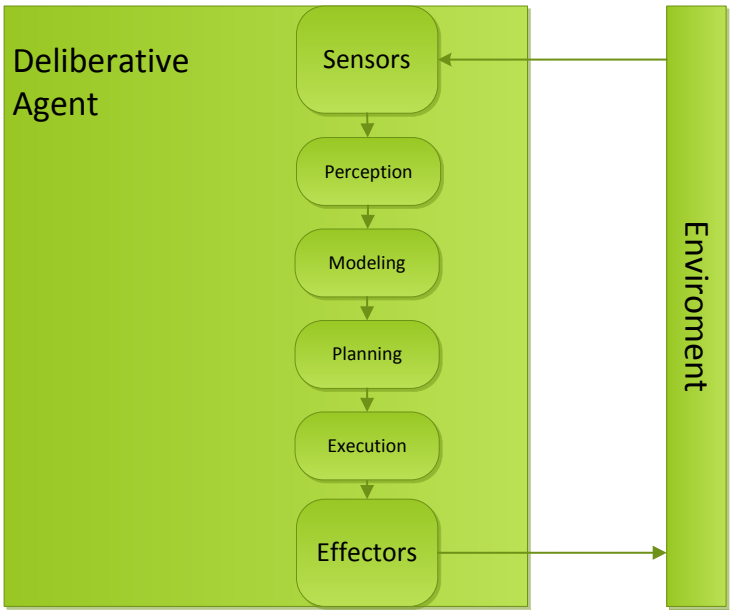


FIGURE 2 - DELIBERATIVE ARCHITECTURE

1.1.3.2. Reactive Architecture

In this architecture, the agent does not need any representation of the world, and does not have any reasoning process either. The agent's behavior is based on a set of simple rules (condition/action). Figure 3, based on [5], demonstrate how this architecture works.

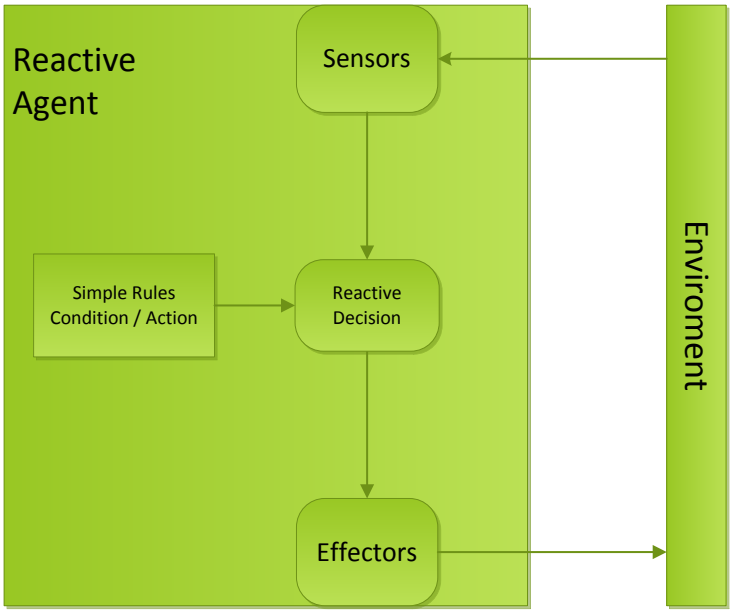


FIGURE 3 - REACTIVE ARCHITECTURE

1.1.3.3. HYBRID ARCHITECTURE

The main goal of this architecture is to gather the better of the two architectures presented before (deliberative and reactive). A hybrid agent approach must combine characteristics from the deliberative architecture, like the representation of the world in a symbolic way, and also have reactive characteristics such as responses to events from the environment, without compromising the reason process. This is normally a layered architecture where the layers are organized by abstraction levels. The higher the layer, the greater will be the abstraction level. The layers can be arranged horizontally or vertically. At the vertical organization, only the first layer interacts with the world, but in the horizontally organization all layers have contact with the world. To have a better view of the hybrid architecture, Figure 4, based on [5], exhibits the two branches of this layered architecture.

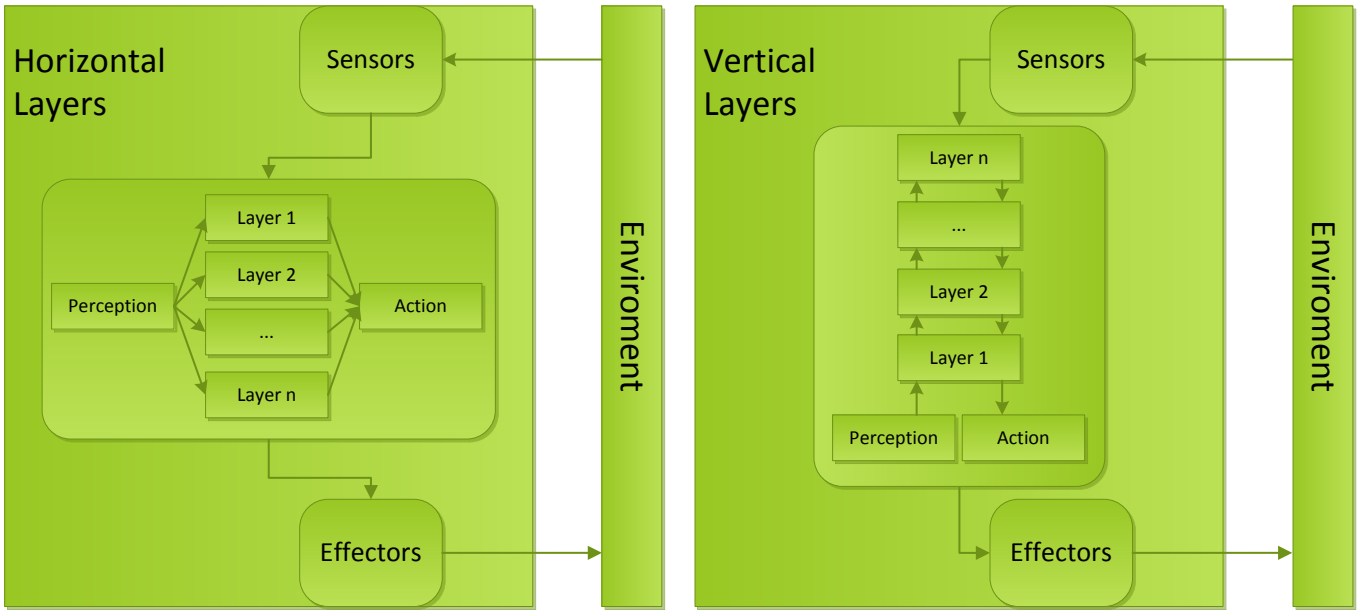


FIGURE 4 - HYBRID ARCHITECTURE

1.1.4. MULTI-AGENT SYSTEMS

The Multi-agent Systems (MAS) field is a relatively recent research area. The MAS born from the Distributed Artificial Intelligence (DAI) field, which is a subfield of AI dedicated to the development of distributed solutions in order to solve problems that require intelligence. MAS are a particular case of DIA where the agents are built in order to coordinate their knowledge, abilities and goals so they can execute action or resolve problems. The agents can work for the same global goal or for their own goals.

Although recent, it is an area that had great growth. Highlight for the creation of a European network to coordinate the investigation in this area – *AgentLink*, [6]. A very interesting event related with the MAS is an international competition of robotic soccer – *Robocup*. This kind of events stimulates the development of the area in many fields, such as communication between agents, behaviors, coordination and others.

An applicant definition can be found in [7]: MAS are systems formed by multiple agents with two important capabilities:

- **Autonomy**: agents must decide by themselves to achieve their goals.
- **Interactivity**: agents must interact with other agents to exchange information, cooperating or negotiating.

The main motivation for the MAS is related with problems that are distributed. Using MAS is possible to divide the problem into more than one agent, turning it simpler.

1.1.4.1. COMMUNICATION IN MAS

The communication is a very important aspect when agents interact. The *Social Ability* and *Interactivity* properties are intimately related with the communication. At the MAS field the communication is high level, i.e. is a language similar to the languages used by the humans. Although in this project the communication will not be used directly between agents, a brief approach to the topic will be made.

Agents must have a communication module implemented in order to send and receive messages following some defined protocol. It is clear that the communication module must be related with the reasoning/intelligent module of the agent. The communication has some characteristics related with the language used by the agents, [8]. Those characteristics must be taken into account: syntax, semantic, vocabulary, pragmatic and domain model of the speech.

As said before, it is crucial to choose a language to communicate. The most used languages are: KQML (Knowledge Query Manipulation Language), ACL (Agent Communication Language), ICL (Inter-agent Communication Language).

1.1.4.2. COOPERATIVE MAS AND COMPETITIVE MAS

It is possible to distinguish two different approaches of MAS, [8]: the cooperative and the competitive approach.

Cooperative MAS are compounded by agents interested in the benefit of the community, more than their own. There is a global concern with the performance of the system.

Competitive MAS are compounded by agents interested in their own benefit. The main goal is to achieve their satisfaction. This kind of agents is known as self-interested.

The use of the Cooperative or Competitive MAS will depend on the context. In this project it will be used a Competitive approach.

2. ARGUMENTATION

To argue is something that everyone does and it is intrinsic to the human being. Who hasn't tried to convince someone of something? Who's never debated with someone? We constantly do reasoning. For this purposes it is obvious that we need to use arguments. Since many centuries ago Argumentation has been studied from many different disciplines, such as philosophy, psychology, neuroscience, linguistic, and other.

Argumentation can be used in many areas. For instance, one area that has particular interest in argumentation is, definitely, the justice, i.e. the law and legal system. The lawyers must be experts in argumentation and have good skills to persuade the others in order to defend their clients. Some work has been developed in the justice field, but at this section the main focus is the relation between Argumentation and AI.

It is clear that Aristotle's studies about Argumentation are still up-to-date and have influence in AI. The syllogisms are a simple example of it, [9]. They are a mechanism of inference that permits us to have conclusions and arguments based on knowledge. As we talk about syllogisms, we also can talk about other issues related to Argumentation that are studied in philosophy and can be applied in AI.

Intuitively, when it comes to Argumentation, we can think that it boils down only into the interaction between agents. And the purpose of each agent is to make its position prevail, convincing the others. Not wrong, but there is another important aspect that has to do with the proper reasoning of an agent. So it is important, not only be persuasive, but also reason correctly.

In the AI field, Argumentation has been studied more strongly over the last years, and we have seen very interesting results. For instance, Dung's argumentation framework is one of the most important and influential works.

Next, we will do an approach to the argumentative systems exploring the theory and the most influent work. Then, a case study of a formalism which uses defeasible argumentation will be exposed.

2.1. ARGUMENTATIVE SYSTEMS

As presented in [10], when it comes to Argumentation, there are four main tasks associated with:

- ***Identification***: recognize the premises and conclusion of some argument and then conclude if they belong to some argumentation scheme.
- ***Analysis***: identify possible implicit premises or conclusions that need to be transformed explicit so that the argument can be evaluated.
- ***Evaluation***: intends to verify if an argument is weak or strong.
- ***Invention***: construction of new arguments to support a specific conclusion.

What is it the notion of argument? Different definitions can be found in the literature. One possible definition is that an argument is a set of statements (propositions), composed of three parts: a conclusion, a set of premises,

and an inference from the premises to the conclusion. Introduced by [11] and [12], the notion of defeasibility is connected with arguments. When a conclusion, supported by rules, is defeated by new information, it is said that such reasoning is *defeasible*. And when a conclusion is reached by this kind of defeasible reasoning, we have arguments, and not proofs.

Arguments can support other arguments, but can also attack other arguments. This leads us to the attack/refutation question. As stated in [10], an argument can be attacked by many ways. First, an argument can be attacked by other if asked for some critical question that raises doubt about its acceptability, making it temporarily default until it responds. Another way to attack an argument is to question one of the premises that compound the argument. One more way to attack an argument is to put forward counter-argument supporting the opposite (negation) conclusion of the original argument.

All of this mentioned notions had the need to be synthetized and normalized in some way. As stated in [13], Argumentative Systems (AS) are a way to formalize common-sense reasoning. The AS are formalisms that intend to support Argumentation and standardize it. In [14] it's possible to find the evolution's description of the first logical models created to support Argumentation. Some examples of it are [15], [16] and [17], but the most influential work is the Dung's abstract argument framework, presented in [18]. Dung presented a very important notion on the acceptability of arguments, and Argumentation is viewed as a special form of logic programming.

2.1.1. Dung's Abstract Framework

Dung proposed a very abstract framework, [18], where the notion of argument is also abstract leaving open the possibility of extension. In this approach, an *argumentation framework* is a pair:

$$\mathcal{AF} = \langle Args, attack \rangle$$

Where $Args$ is the set of all possible arguments, and $attack$ is a binary relation on $Args$. If$(A, B) \in attack$, means that argument A attacks argument B. In his work, the present idea about the attack relation between arguments is that an argument A will be defeated if it is possible to find at least one defeater for it that is not defeated.

This approach assumes the existence of a set of arguments ordered by a binary relation of defeat. Nevertheless, it is possible to define various notions of `argument extensions', which aim to capture various types of defeasible consequence.

2.1.2. Case Study: DeLP

Now the *Defeasible Logic Programming* (DeLP) formalism will be introduced. This work is presented and fully described in [19]. The formalism associates Logic Programming and Defeasible Argumentation and evolved from [16]. With DeLP is possible to represent information in the form of *weak rules*, and reach to warranted conclusions using a defeasible argumentation inference mechanism. The *weak rules* are very important since they are the central point of defeasibility.

2.1.2.1. LANGUAGE

Three disjoint sets compound the DeLP language: a set of *facts*, a set of *strict rules* and a set of *defeasible rules*. In the DeLP language we consider that a literal "L" is a ground atom "A" or a negated ground atom "$\sim A$" ("$\sim$" represents strong negation).

A *Fact* is a literal, i.e., a ground atom or its negation.

A *Strict Rule* is an ordered pair, "$Head \leftarrow Body$" , where $Head$ is a literal and $Body$ is a finite non-empty set of literals.

A *Defeasible Rule* is an ordered pair, "$Head -< Body$", where $Head$ is a literal and $Body$ is a finite non-empty set of literals.

A *Defeasible Logic Program* P (de.l.p.), is a possibly infinite set of facts, strict rules and defeasible rules. The facts and strict rules are grouped in a subset Π, and the defeasible rules are grouped in a subset Δ. A de.l.p. P can be denoted as $P = (\Pi, \Delta)$.

Defeasible Derivation: Let $P = (\Pi, \Delta)$ be a de.l.p. and L a ground literal. A defeasible derivation of L from P, denoted $P \mid\sim L$, consists of a finite sequence $L_1, L_2, \ldots, L_n = L$ of ground literals, and each literal L_i is in the sequence because is a fact in Π or there exists a rule R_i in P (strict or defeasible) with head L_i and body $B_1, B_2, \ldots, B_k$ and every literal of the body is an element L_j of the sequence appearing before L_i $(j < i)$. A derivation for a literal L from P is called `defeasible', because there may exist information in contradiction with L that will prevent the acceptance of L as a valid conclusion.

Strict Derivation: Let $P = (\Pi, \Delta)$ be a de.l.p. and h a literal with a defeasible derivation $L_1, L_2, \ldots, L_n = h$. It is possible to say that h has a strict derivation from P, denoted $P \vdash h$, if either h is a fact or all the rules used for obtaining the sequence $L_1, L_2, \ldots, L_n$ are strict rules. The symbol "$^-$" will be used to denote the complement of a literal with respect to strong negation, i. e. $\bar{p}$ is $\sim p$, and $\overline{\sim p}$ is p. Two literals are contradictory if they are complementary.

A *Contradictory set of rules* occurs if and only if, there exist a defeasible derivation for a pair of complementary literals from the set.

2.1.2.2. DEFEASIBLE ARGUMENTATION

We will now address the formalism itself, focusing on the defeasibility of the argumentation and how it can be made. The authors claim that the central notion of this formalism is the notion of *argument*, and they refer that an argument can be informally defined as a minimal and non-contradictory set of rules used to derive a conclusion. In DeLP, an argument will support answer to queries. Although a de.l.p. could be contradictory, answers to queries will be supported by a non-contradictory set of rules.

Argument Structure: Let h be a literal, and $P = (\Pi, \Delta)$ a de.l.p.. $\langle A, h \rangle$ is an argument structure for h, if A is a set of defeasible rules of Δ, such that:

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

Summarizing, an argument structure $\langle A, h\rangle$, or simply an argument A for h, is a minimal non-contradictory set of defeasible rules, obtained from a defeasible derivation for a given literal h. The literal h can also be called the `conclusion' supported by A.

Sub-argument structure: An argument structure $\langle B, q\rangle$ is a sub-argument structure of $\langle A, h\rangle$ if $B \subseteq A$. Note that the union of arguments is not always an argument, that is, given two argument structures $\langle A, h\rangle$ and $\langle B, q\rangle$, the set $A \cup B$ could not be proper for being used as an argument, because $A \cup B$ could be not minimal or $A \cup B \cup \Pi$ could be contradictory. Figure 5, from [19], graphically show a representation of an argument and one of its sub-arguments.

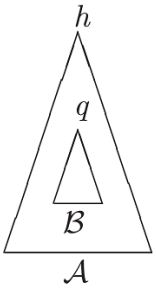


FIGURE 5 - AN ARGUMENT A, AND A SUB-ARGUMENT B

2.1.2.2.1. REBUTTALS OR COUNTER-ARGUMENTS

Although answers to queries are supported by arguments in DeLP, an argument may be defeated by other arguments. A query q will succeed if the supporting argument for it is not defeated. To establish whether A is a non-defeated argument, *argument rebuttals* or *counter-arguments* that could be defeaters for A must be taken into account. Since counter-arguments are arguments, there may be possible to find defeaters for them, and so on. That takes us into a dialectical analysis that will be elucidated next using some important notions.

Disagreement: Let $P = (\Pi, \Delta)$ be a de.l.p.. We say that two literals h and h_1 *disagree*, if and only if the set $\Pi \cup \{h, h_1\}$ is contradictory. Two complementary literals trivially disagree. However, two literals that are not complementary can also disagree if they derive complementary conclusions.

Counter-argument: $\langle A_1, h_1\rangle$ 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. If $\langle A_1, h_1\rangle$ counter-argues $\langle A_2, h_2\rangle$ at literal h, then h is called a *counter-argument point*, and the sub-argument $\langle A, h\rangle$ is called the *disagreement sub-argument*. Figure 6 (left), from [19], show a graphical representation of an argument and one of its counter-arguments and Figure 6 (right) show that an argument can also attack directly a conclusion from another.

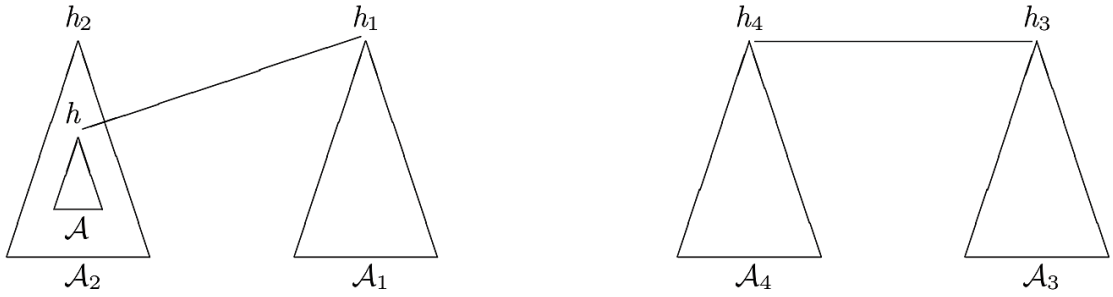


FIGURE 6 - INDIRECT ATTACK (LEFT) AND DIRECT ATTACK (RIGHT)

2.1.2.2.2. COMPARING ARGUMENTS

One of the crucial points on defeasible argumentation is the argument comparison. Some different approaches have been made through the literature. In the DeLP formalism, two methods to compare arguments are proposed: one based on specificity and other based on priorities among rules.

2.1.2.2.2.1. GENERALIZED SPECIFICITY

The main goal of this criterion is to discriminate between two conflicting arguments. The definition of specificity favors two aspects: it prefers an argument (1) with greater information content or (2) with less use of rules (more direct). This means that an argument will be deemed better than another if it is more precise or more concise.

Specificity: Let $P = (\Pi, \Delta)$ be a de.l.p., and let Π_G the set of all strict rules from Π. Let F be the set of all literals that have a defeasible derivation from P (F will be considered as a set of facts). Let $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ be two argument structures obtained from P. $\langle A_1, h_1\rangle$ is strictly more specific than $\langle A_2, h_2\rangle$ (denoted $\langle A_1, h_1\rangle \succ \langle A_2, h_2\rangle$) if the following conditions hold:

1. For all $H \subseteq F$: if $H_G \cup H \cup A_1 \mathrel{|\!\sim} h_1$ and $H_G \cup H \nvdash h_1$, then $H_G \cup H \cup A_2 \mathrel{|\!\sim} h_2$, and
2. There exists $H' \subseteq F$ such that $H_G \cup H' \cup A_2 \mathrel{|\!\sim} h_2$ and $H_G \cup H' \nvdash h_2$, and $H_G \cup H' \cup A_1 \not\mathrel{|\!\sim} h_1$

It is not possible to have a defeasible derivation for a literal from a set of rules without facts. From $H_G \cup A_1$, it's not possible to defeasible derive h_1. Since H is a set of facts, it could be possible to derive h_1 from $H_G \cup H \cup A_1$, so when $H_G \cup H \cup A_1 \mathrel{|\!\sim} h_1$ holds, we say that H *activates* $\langle A_1, h_1\rangle$, or H is an *activation set* of $\langle A_1, h_1\rangle$.

Equi-Specificity: Two arguments $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ are equi-specific ($\langle A_1, h_1\rangle \equiv \langle A_2, h_2\rangle$), if and only if $A_1 = A_2$, and the literal h_2 has a strict derivation from $\Pi \cup \{h_1\}$, and the literal h_1 has a strict derivation from $\Pi \cup \{h_2\}$.

2.2.2.2.2.2. ARGUMENT COMPARISON USING RULE'S PRIORITIES

Explicit priorities among rules are used to decide between competing conclusions. DeLP uses it to compare arguments that support the conclusions. A particular comparison criterion has introduced by the authors, based on rule priorities. It must be assumed that explicit priorities among defeasible rules are given with the program.

Note that a strict derivation has no counter-argument, so it will be preferred over arguments that use defeasible rules.

Priority rules: Let P be de.l.p. and "$>$" a preference relation explicitly defined among defeasible rules. Given two argument structures $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$, the argument $\langle A_1, h_1\rangle$ will be preferred over $\langle A_2, h_2\rangle$ if and only if:

1. there exists at least one rule $r_a \in A_1$, and one rule $r_b \in A_2$, such that $r_a > r_b$,
2. and there is no $r'_b \in A_2$ and $r'_a \in A_1$, such that $r'_a > r'_b$.

2.1.2.3. Defeaters and Argumentation Lines

Now, some terms and notions will be exposed in order to understand the concepts of defeaters and argumentation lines. Abstracting from the comparison criterion adopted, we will assume that exist one and it will be denoted as "$\succ$". To continue exposing the formalism it will be assumed that "$\succ$" means "strictly more specific".

Proper Defeater: 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 $\langle A_1, h_1\rangle \succ \langle A, h\rangle$. This situation represents an indirect attack, but it can also be applied to a direct attack.

Blocking Defeater: 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 $\langle A, h\rangle$, i.e. $\langle A_1, h_1\rangle \not\succ \langle A, h\rangle$, and $\langle A, h\rangle \not\succ \langle A_1, h_1\rangle$.

Defeater: 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 $\langle A_2, h_2\rangle$.

During the argumentation it is not possible to attack a sub-argument $\langle A, h\rangle$ with an argument $\langle A_1, h_1\rangle$ that is equi-specific to $\langle A, h\rangle$.

Argumentation Line: Let P be a de.l.p. and $\langle A_0, h_0\rangle$ an argument structure obtained from P. An argumentation line for $\langle A_0, h_0\rangle$ is a sequence of argument structures from P, denoted $\Lambda = [\langle A_0, h_0\rangle, \langle A_1, h_1\rangle, \langle A_2, h_2\rangle, \langle A_3, h_3\rangle, \dots]$, where each element of the sequence $\langle A_i, h_i\rangle, i > 0$, is a defeater of its predecessor $\langle A_{i-1}, h_{i-1}\rangle$. The Figure 7, from [19], represents graphically an argumentation line.

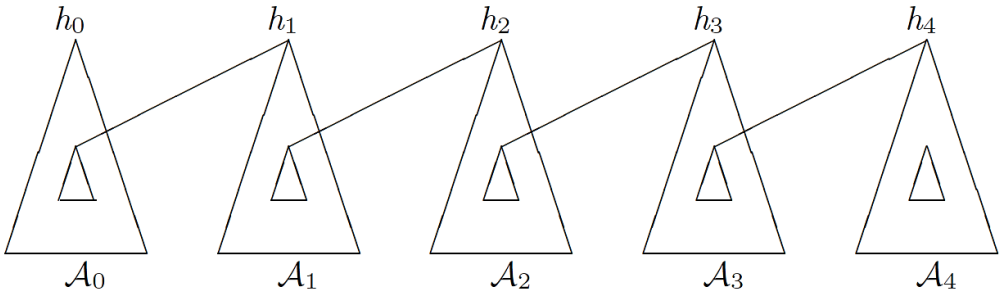


FIGURE 7 - ARGUMENTATION LINE

Supporting and Interfering argument structures: Let $\Lambda = [\langle A_0, h_0\rangle, \langle A_1, h_1\rangle, \langle A_2, h_2\rangle, \langle A_3, h_3\rangle, \dots]$ be an argumentation line. It is possible to define the set of supporting argument structures $\Lambda_S = [\langle A_0, h_0\rangle, \langle A_2, h_2\rangle, \langle A_4, h_4\rangle, \dots]$, and the set of interfering argument structures $\Lambda_I = [\langle A_1, h_1\rangle, \langle A_3, h_3\rangle, \dots]$. Given an argument structure $\langle A_0, h_0\rangle$, there can be many defeaters for $\langle A_0, h_0\rangle$, and each of them will generate a different argumentation line.

2.1.2.3.1. ACCEPTABLE ARGUMENTATION LINES

If some restrictions are not imposed, some undesirable situations can occur, which can take us into infinite argumentation lines. Now some of these situations will be described and then constrains to avoid it.

An argument structure $\langle A, h\rangle$ is said to be "self-defeating" if $\langle A, h\rangle$ is a defeater for itself. If $\langle A, h\rangle$ is a self-defeating argument structure then an argumentation line starting with $\langle A, h\rangle$ will be infinite. Figure 8, from [19], represents the situation. So, in DeLP, arguments will never be self-defeating.

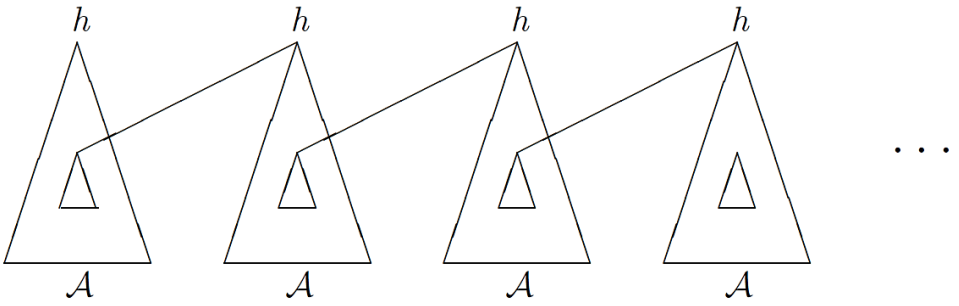


FIGURE 8 - INFINITE ARGUMENTATION LINE WITH A SELF DEFEATING ARGUMENT

Another undesirable situation happens when a pair of arguments defeats each other. They are called reciprocal defeaters. The Figure 9, from [19], represents this reciprocal situation. To avoid it, it is imperial to identify them.

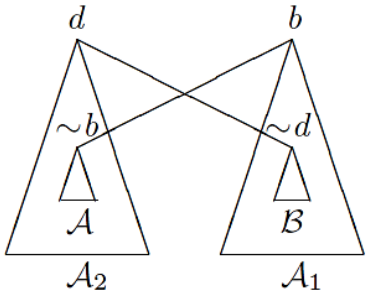


FIGURE 9 - RECIPROCAL DEFEATERS

The circular argumentation is also another bad situation. It occurs when an argument structure is reintroduced in an argumentation line. At Figure 10, from [19], is possible to see that argument A is reintroduced in order to support itself. This will lead into an infinite argumentation line. To avoid this situation the following condition must be imposed: "*no argument can be reintroduced in the same argumentation line*".

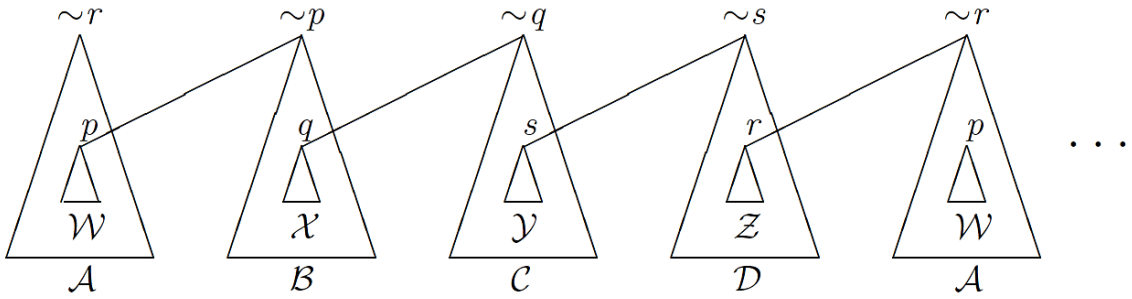


FIGURE 10 - CIRCULAR ARGUMENTATION

Even though this situation is safeguarded a more particular and subtle case can occur, which is the reintroduction of a sub-argument. Figure 11, from [19], shows that argument A is defeated by argument B, with W as the disagreement sub-argument. Later W is reintroduced as defeater. This situation is fallacious.

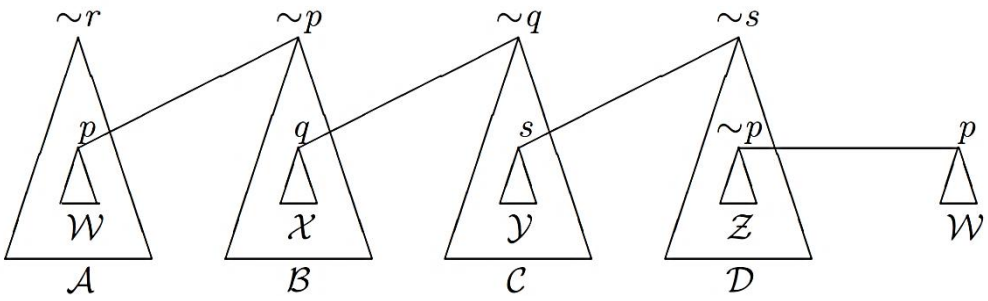


FIGURE 11 - CIRCULAR ARGUMENTATION WITH A SUB-ARGUMENT

Another undesirable situation is shown in Figure 12, where an argument A becomes both a supporting and an interfering argument of itself. It happens because the supporting argument C has a sub-argument Z for the literal r, which is contradictory with arguing in favor of $\sim r$ (argument A). The reintroduction of an argument like C should be avoided. This lead us to the concept of agreement among supporting arguments (and also interfering) in any argumentation line.

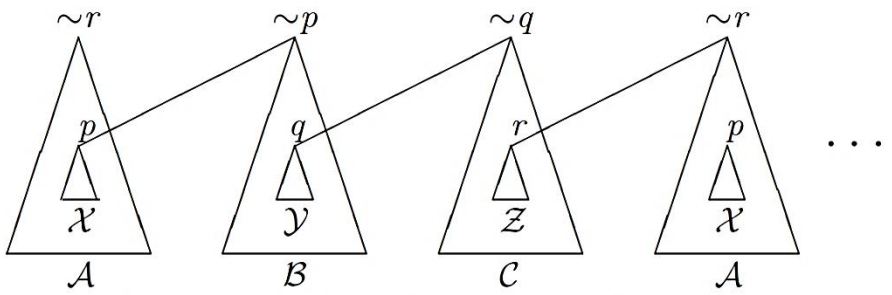


FIGURE 12 - CONTRADICTORY ARGUMENTATION LINE

Concordance: Let $P = (\Pi, \Delta)$ be a de.l.p. Two arguments $\langle A_1, h_1\rangle$ and $\langle A_2, h_2\rangle$ are concordant if the set $\Pi \cup A_1 \cup A_2$ is non-contradictory. Generally, a set of argument structures $\{A_i, h_i\}_{i=1}^{n}$is concordant if and only if $\Pi \cup \bigcup_{i=1}^{n} A_i$is non-contradictory.

The above definition is applicable to the set of supporting arguments and the set of interfering arguments. The next definition elucidates us about the notion of an *acceptable* argumentation line, which will prevent the undesirables situations mentioned above.

Acceptable argumentation line: Let $\Lambda = [\langle A_1, h_1\rangle, \ldots, \langle A_i, h_i\rangle, \ldots, \langle A_n, h_n\rangle]$ be an argumentation line. Λ is anacceptable argumentation line if:

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 subargument of an argument $\langle A_i, h_i\rangle$ appearing earlier in $\Lambda (i < k)$.
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 $\langle A_i, h_i\rangle$.

2.1.2.4. WARRANT THROUGH DIALECTICAL ANALYSIS

In DeLP, warrants are applied to literals. A literal h will be warranted if there exists a non-defeated argument structure $\langle A, h\rangle$. To establish whether $\langle A, h\rangle$ is non-defeated, the set of defeaters for A will be considered. Since each defeater D for A is itself an argument structure, the defeaters for D will in turn be considered, and so on. Therefore, more than one argumentation line could be generated. This leads us to a tree structure that it is called dialectical tree.

Dialectical Tree: Let P be a de.l.p. and $\langle A_0, h_0\rangle$ an argument structure. 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 $\Lambda = [\langle A_0, h_0\rangle, \langle A_1, h_1\rangle, \langle A_2, h_2\rangle, \ldots, \langle A_n, h_n\rangle]$ 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 defeaters for $\langle A_n, h_n\rangle$.
   For each defeater $\langle B_i, q_i\rangle$, with $(1 \leq i \leq k)$, such that, the argumentation line Λ', where $\Lambda' = [\langle A_0, h_0\rangle, \langle A_1, h_1\rangle, \langle A_2, h_2\rangle, \ldots, \langle A_n, h_n\rangle, \langle B_i, q_i\rangle]$ 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 $\langle B_i, q_i \rangle$ i such that Λ' is acceptable, then N is a leaf.

Every node (except the root) from a dialectical tree represents a defeater of its parent, and leaves correspond to non-defeated arguments. Each path from the root to a leaf matches to different acceptable argumentation lines. The dialectical tree provides a structure for considering all the possible acceptable argumentation lines that can be generated for deciding whether an argument is defeated or not. Figure 13, from [19], represents a dialectical tree for $\langle A, a \rangle$.

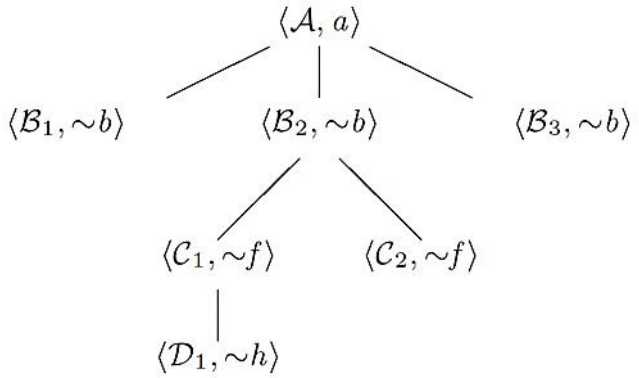


FIGURE 13 - DIALECTICAL TREE

Marking of a dialectical tree (procedure): 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"s in $T_{\langle A,h \rangle}$.
2. Let $\langle B, q \rangle$ be an inner node of $T_{\langle A,h \rangle}$. Then $\langle B, q \rangle$ will be marked as "U" (undefeated) in $T^*_{\langle A,h \rangle}$ if and only if every child of $\langle B, q \rangle$ is marked as "D" (defeated). The node $\langle B, q \rangle$ will be marked as "D" in $T^*_{\langle A,h \rangle}$ if and only if it has at least a child marked as "U".

Figure 14, from [19], show the dialectical tree of Figure 13 after applying the marking procedure.

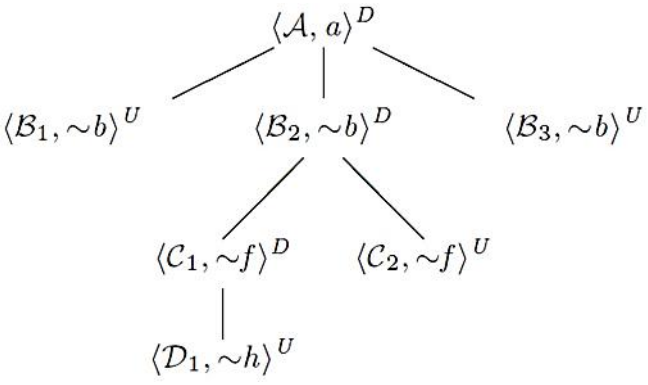


FIGURE 14 - MARKED DIALECTICAL TREE

*Warranted literal*s: 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 if and only if the root of $T^*_{\langle A,h \rangle}$ is marked as "U". Then A is a warrant for h.

Taking into account the notion of warrant, a modal operator will of belief "B" will be defined, where Bh means that h is warranted, and $\neg Bh$ means h is not warranted.

Answer to queries: Answers from 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.
- No, if $\bar{h}$.
- UNDECIDED, if $\neg Bh$ and $\neg B\bar{h}$.
- UNKNOWN, if h is not in the language of the program.

2.2.2.4.1. THE WARRANT PROCEDURE WITH PRUNING

To find out if a literal h is warranted, the warrant procedure has to find the argument structure $\langle A, h\rangle$ and the root $T^{*}_{\langle A,h\rangle}$ has to be marked as "U". To do it in an efficient way, the procedure should not explore the whole dialectical tree.

Given a program P, for a literal h there could be several argument structures $\langle A_1, h\rangle, \ldots, \langle A_i, h\rangle$. The procedure will not construct all the possible argument structures for h. It will consider each one of them in turn, exploring the associated dialectical tree. Its observable that during the marking of the dialectical tree, some nodes are not contributing to the decision procedure, i. e. it does not matter if they are marked as "U" or "D" because it will not change the marking of the dialectical tree's root.

Figure 15 (left), from [19], show that the left-most child of the root is a marked as "U", so the root is marked as "D". Now, no matter how it is marked the other two children of the root, then they can be pruned,

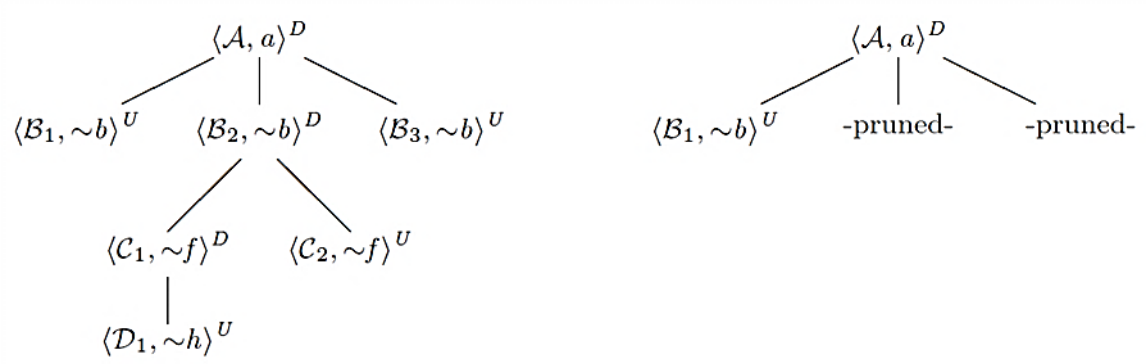


FIGURE 15 - MARKED DIALECTICAL TREE (LEFT) AND PRUNED (RIGHT)

Consequently, a dialectical tree will be generated in depth-first manner, considering (from left to right) every acceptable argumentation line. This procedure is very similar to the *minimax* algorithm used in AI for games trees.

3. BELIEF REVISION

Belief Revision (BR, known also as belief change or belief dynamics) is defined as the process of changing beliefs to adapt the epistemic state of an agent to new information. It is relatively young research field starting mainly in the 80's.

Two distinct areas contributed to the development of this topic. One was computer science, where the issue of updating of databases correctly influenced AI researchers to build more complex models of database updating. An important work developed by Jon Doyle is the Truth Maintenance Systems, [15]. The other area that contributed for the development of this topic was philosophy. The first studies were from Levi with [20] and [21], and Harper with [22]. But the most influential work is the AGM model presented in 1985 by Carlos Alchourrón, Peter Gärdenfors, and David Makinson, [23], where the authors presented a general and versatile framework for studies of belief change. From this work, many developments have been made through the last years, such as change operators, applications of the model and others. In [24] Fermé and Hanson summarized the first 25 years of studies and developments about the AGM model.

In this section we will expose the AGM model and make an approach to the safe and kernel contraction. Then we will address the belief bases topic.

3.1. AGM

As stated in [25], in the dominant Belief Revision theory (AGM model), the set representing the belief state is assumed to be a logically closed set of sentences (called a belief set), but in an alternative approach, the corresponding set is not logically closed (called a belief base). This difference will be further explored later.

In the AGM model, beliefs are represented by sentences in some formal language. Let L be a language, the sentences will be represented by lowercase letters $(p, q, r, s, \dots)$ and set of sentences by capital letters (X, Y, Z). The letters $(K, K', \dots, H, H', \dots)$ will represent set of sentences closed under logical consequence. We will assume that language contains the usual truth-functional connectives: negation $(\neg)$, conjunction $(\wedge)$, disjunction $(\vee)$, implication $(\rightarrow)$, and equivalence $(\leftrightarrow)$. The symbol $(\perp)$ denotes an arbitrary contradiction, and $(\top)$ an arbitrary tautology.

Beliefs of an agent are represented by beliefs sets closed under logical consequence, i.e. for any set X of sentences, $Cn(X)$ is the set of logical consequences of X. A consequence operation on a given language is a function Cn from sets of sentences to sets of sentences and it satisfies the following three conditions:

- **Inclusion**: $X \subseteq Cn(X)$.
- **Monotony**: If $X \subseteq Y$, then $Cn(X) \subseteq Cn(Y)$.
- **Iteration**: $Cn(X) = Cn(Cn(X))$.
- **Supraclassical**: if p can be logically derived from X, then $p \in Cn(X)$.

We say that $K \in L$ is a belief set if and only if $K = Cn(K)$. $X \vdash p$ is an alternative notation for $p \in Cn(X)$, and $X \nvdash p$ for $p \notin Cn(X)$. $Cn(\emptyset)$ is the set of tautologies.

In AGM there are three main rationality criteria:

- **Priority of the new information**: new information is always accepted;
- **Consistency**: if possible, the new epistemic state should be consistent;
- **Informational economy**: Retain, as much as possible, the pre-existent beliefs.

Three types of belief change are present in the AGM framework: contraction, expansion and revision. In contraction, a specified sentence p is removed (a belief set K is superseded by another belief set $K \div p$ that is a subset of K not containing p). In expansion a sentence p is added to K, and nothing is removed (K is replaced by a set $K + p$ that is the smallest logically closed set that contains both K and p). Expansion is characterized by $K + p = C_n(K \cup p)$. In revision a sentence p is added to K, and at the same time other sentences are removed if this is needed to ensure that the resulting belief set $K * p$ is consistent.

3.1.1. CONTRACTION

The result of contracting K by p should be a subset of K that does not imply p. So it must be considered the inclusion-maximal subsets of K that do not imply p.

Contraction of a belief set by a sentence should satisfy six basic AGM postulates:

- **Closure**: $K \div p = Cn(K \div p)$.
- **Success**: If $p \notin Cn(\emptyset)$, then $p \notin Cn(K \div p)$.
- **Inclusion**: $K \div p \subseteq K$.
- **Vacuity**: If $p \notin Cn(K)$, then $K \div p = K$.
- **Extensionality**: If $p \leftrightarrow q \in Cn(\emptyset)$, then $K \div p = K \div q$.
- **Recovery**: $K \subseteq (K \div p) + p$.

Note: Recovery is one of most criticized postulates, [24], since it says that so much is retained after p has been removed that everything can be recovered by reinclusion (through expansion) of p. This postulate, in general, is not satisfied when using belief bases.

Additionally, in order to characterize non-atomic contractions (conjunctions), two more postulates (supplementary) must be considered:

- **Conjunctive inclusion**: If $p \notin K \div (p \wedge q)$, then $K \div (p \wedge q) \subseteq K \div p$.
- **Conjunctive overlap**: $(K \div p) \cap (K \div q) \subseteq K \div (p \wedge q)$.

The first six basic contraction postulates characterizes exactly what is called par*tial meet con*traction, and all the contraction postulates (eight) characterizes exactly transitively relational *partial meet contraction*.

3.1.2. REVISION

A revision operator carries two important tasks: to add the new belief p to the belief set K; and to ensure that the resulting belief set $K * p$ is consistent (unless p is inconsistent).

The first task can be done using expansion by p. The second can be accomplished by prior contraction by its negation $\neg p$, this is, if a belief set does not imply $\neg p$, then p can be added to it without loss of consistency (Levi identity).

An operator * is a revision operator if and only if it satisfies the following six postulates:

- **Closure**: $K * p = Cn(K * p)$.
- **Success**: $p \in K * p$.
- **Inclusion**: $K * p \subseteq K + p$.
- **Vacuity**: If $\neg p \notin K$, then $K * p = K + p$.
- **Consistency**: $K * p$ is consistent if p is consistent.
- **Extensionality**: If $(p \leftrightarrow q) \in Cn(\emptyset)$, then $K * p = K * q$.
- Additionally, two supplementary postulates are part of the standard repertoire:
- **Superexpansion**: $K * (p \wedge q) \subseteq (K * p) + q$.
- **Subexpansion**: If $\neg q \notin Cn(K * p)$, then $(K * p) + q \subseteq K * (p \wedge q)$.

The first six basic revision postulates characterizes exactly par*tial meet revi*sion, and all the revision postulates (eight) characterizes exactly transitively relational *partial meet revision*.

3.1.3. LEVI AND HARPER IDENTITIES

As we have seen, the contraction and revision functions are characterized by two different sets of postulates. These sets of postulates are independent. The contraction postulates do not refer to the revision operation nor do the revision postulates refer to the contraction operation. However, the revision can be defined through contraction and vice-versa, [26]. This can be done using the Levi and Harper identities.

3.1.3.1. FROM CONTRACTION TO REVISION

Through the Levi identity it is possible to define revision in terms of contraction:

Levi identity: $K * p = (K \div \neg p) + p$.

Revision consists in two operations:

1. Contract K by $\neg p$ and obtain, if possible, a subset of K consistent with the sentence p;
2. Expand the result by the new sentence.

3.1.3.2. From Revision to Contraction

Through the Harper identity it is possible to define contraction in terms of revision:

Harper identity: $K \div p = K \cap (K * \neg p)$.

The contraction is defined by the intersection of K with the revision of K by $\neg p$.

As stated in [26] by Fermé, given these identities, we can define revision or contraction functions, and the other will be univocally defined.

3.2. Safe and Kernel Contraction

The operation of *safe contraction* was proposed by Alchourrón and Makinson in [27]. Safe contraction is based on a non-circular relation $<$ on the elements of K. An element a of K is safe with respect to p if and only if all inclusion-minimal p-implying subsets of K either do not contain a or contain some b such that $b < a$. The safe contraction $\div$ based on $<$ yields as outcome the logical closure of the set of sentences in K that are safe with respect to p. Safe contractions are partial meet contractions.

Hanson, in [28], introduced *kernel contraction.* As stated in, [24] kernel contraction is a non-relational generalization of safe contraction. Let Kp be the set of minimal p-implying subsets of K. σ is an incision function that selects sentences to be discarded. It satisfies the two basic properties:

1. $\sigma(K \Vdash p) \subseteq \bigcup(K \Vdash p)$ and
2. if $\emptyset = X \in K \Vdash p$,then $X \cap \sigma(K \Vdash p) \neq \emptyset$.

The kernel contraction $\approx_\sigma$ based on σ is defined by the relationship $K \approx_\sigma p = K \backslash \sigma(K \Vdash p)$.

Kernel contractions meet the six basic AGM postulates (partial meet contraction) and satisfy the additional condition of smoothness, namely that if $X \subseteq K$ and $Cn(X) \cap \sigma(K \Vdash p) \neq \emptyset$, then $X \cap \sigma(K \Vdash p) \neq \emptyset$.

3.3. BELIEF BASES

As said before, a belief set is closed under logical consequence, but there are beliefs that should not be taken into account, [25]. Hence, suppose that the belief set contains the sentence p, "Shakespeare wrote Hamlet". Due to logical closure it then also contains the sentence $p \vee q$, "Either Shakespeare wrote Hamlet or Charles Dickens wrote Hamlet", which is a "mere logical consequence" that should have no standing of its own.

A belief base is a set of sentences that is not closed under logical consequence. As stated in [25], the elements of a belief base represent beliefs that are held independently of any other belief or set of beliefs. Changes are performed on the belief base. The underlying intuition is that the merely derived beliefs are not worth retaining for their own sake. If one of them loses the support that it had in basic beliefs, then it will be automatically discarded.

For every belief base X, there is a belief set $Cn(X)$ that represents the beliefs held according to X. On the other hand, a belief set can be represented by different belief bases. Several advantages of using belief bases can be enumerated, [26]:

1. Belief bases have more expressive power than belief sets. Example: The two beliefs bases $\{p, q\}$ and $\{p, p \leftrightarrow q\}$ have the same logical closure. They are therefore *statically equivalent*, in the sense of representing the same beliefs. On the other hand, they are not *dynamically equivalent* in the sense of behaving in the same way under operations of change. They can be taken to represent different ways of holding the same beliefs.
2. Belief bases allow us to represent resource bounded agents. The ideal agents can have infinite inferential resources. However, "real" agents do not have those unlimited resources and the use of belief bases is a progress in that direction.
3. Using belief bases, it is possible to distinguish between different inconsistent sets (an agent can be inconsistent).
4. Since the belief sets are big entities (in some cases can be infinite), for practical applications belief bases should be used.

3.3.1. BELIEF BASE CONTRACTION

Partial meet contraction, as defined in section 3.1.1, is also applicable to belief bases, [25]. $A \perp p$ is the set of maximal subsets of A that do not imply p. It is not sufficient that they do not contain p. So, $\{p \vee q, p \leftrightarrow q\} \perp p = \{\{p \vee q\}, \{p \leftrightarrow q\}\}$. Excluding Recovery, the basic postulates for partial meet contraction on belief sets are applicable for belief bases too.

Hansson, in [29], characterized partial meet contraction on belief bases. An operator $\div$ is an operator of partial meet contraction for a set A if and only if it satisfies the following four postulates:

- **Success**: If $p \notin Cn(\emptyset)$, then $p \notin Cn(A \div p)$.
- **Inclusion**: $A \div p \subseteq A$.

- **Relevance**: If $q \in A$ and $q \notin A \div p$, then there is a set A' such that $A \div p \subseteq A' \subseteq A$ and that $p \notin Cn(A')$ but $p \in Cn(A' \cup \{ q \})$.
- **Uniformity**: If it holds for all subsets A' of A that $p \in Cn(A')$ if and only if $q \in Cn(A')$, then $A \div p = A \div q$.

Another approach to contraction of belief bases has been proposed under the name kernel base contraction. As we have seen before in section 3.2, for any sentence p, a p-kernel is a minimal p-implying set, i.e. a set that implies p but has no proper subset that implies p. A contraction operation $\div$ can be based on the simple principle that no p-kernel should be included in $A \div p$. This can be obtained with an incision function, a function that selects at least one element from each p-kernel for removal. An operation that removes exactly those elements that are selected for removal by an incision function is called an operation of kernel contraction. All partial meet contractions on belief bases are kernel contractions, but there are kernel contractions that are not partial meet contractions. This means that kernel contraction is a generalization of partial meet contraction.

3.3.2. Belief Base revision

The expansion operator for belief sets, $K + p = Cn(K \cup \{p\})$, was constructed to guarantee that the outcome is logically closed. For belief bases this is not desirable, and therefore expansion on belief bases must be different. For any belief base A and sentence p, $A +' p$, the expansion of A by p is the set $A \cup \{p\}$, [25], i.e. $A +' p = A \cup \{p\}$.

Revision operators for belief bases can be constructed out of two suboperations: expansion by p and contraction by $\neg p$ (Levi identity: $A * p = (A \div \neg p) +' p$). Alternatively, the two suboperations presented before may take place in reverse order, $A * p = (A +' p) \div \neg p$. This is applicable only to belief bases. For belief bases there are two distinct ways to base revision on contraction and expansion:

Internal revision: $A * p = (A \div \neg p) +' p$.

External revision: $A * p = (A +' p) \div \neg p$.

External revision by p is revision with an *intermediate inconsistent state* in which both p and $\neg p$ are believed, whereas internal revision has an *intermediate non-committed* state in which neither p nor $\neg p$ is believed, [25].

3.3.2.1. Internal Revision

In [30], Hanson states that an operator $*$ is an operator of internal revision for a belief base A if and only if it satisfies the following postulates:

- **Consistency**: $A * p$ is consistent if p is consistent.
- **Inclusion**: $A * p \subseteq A \cup \{p\}$.
- **Relevance**: If $q \in A$ and $q \notin A * p$, then there is some A' such that $A * p \subseteq A' \subseteq A \cup \{p\}$, A' is consistent but $A' \cup \{q\}$ is inconsistent.
- **Success**: $p \in A * p$.

- **Uniformity**: If for all $A' \subseteq A$, $A' \cup \{p\}$ is inconsistent if and only if $A' \cup \{q\}$ is inconsistent, then $A \cap (A * p) = A \cap (A * q)$.

3.3.2.2. EXTERNAL REVISION

Hanson, in [31], also states that an operator $*$ is an operator of external revision if and only if it satisfies the following postulates:

- **Consistency**: $A * p$ is consistent if p is consistent.
- **Inclusion**: $A * p \subseteq A \cup \{p\}$.
- **Relevance**: If $q \in A$ and $q \notin A * p$, then there is some A' such that $A * p \subseteq A' \subseteq A \cup \{p\}$, A' is consistent but $A' \cup \{q\}$ is inconsistent.
- **Success**: $p \in A * p$.
- **Weak uniformity**: If p and q are elements of A and it holds for all $A' \subseteq A$ that $A' \cup \{p\}$ is inconsistent if and only if $A' \cup \{q\}$ is inconsistent, then $A \cap (A * p) = A \cap (A * q)$.
- **Pre-expansion**: $A + p * p = A * p$.

4. BELIEF REVISION AND ARGUMENTATION

In [32], the authors presented an article in which they explore the relationship between Belief Revision and Argumentation, through the development of a conceptual view on the topic. They defend Argumentation and Belief Revision as complementary disciplines. It is not possible to construct a good model of decision making for real world application without the support of both disciplines. In this section we will explore the relationship between them, presenting some earlier work and then the conceptual view.

4.1. EARLIER WORK

Along the years some work linking both areas have been developed. Based on [32], it will be briefly presented some of those works: Truth Maintenance Systems, Belief Revision and Epistemology, Deductive Explanations and Belief Revision, Data-oriented Belief Revision, Prioritized Revision by Arguments, Relating Reinstatement and Recovery.

4.1.1. TRUTH MAINTENANCE SYSTEMS

Doyle, in 1979, presented the truth maintenance system (TMS), which is a knowledge representation method for representing beliefs and justifications, [15]. The TMS system intends to restore consistency when new information arrives. A TMS associates a special data structure (node), with each problem solver datum (database entries, inference rules, procedures) and it records justifications (arguments) for potential inferred beliefs and also to compute the current set of beliefs (manipulating the status of nodes and evaluating justifications). The process starts when a new justification for a node is added, and runs through basic steps of Argumentation (evaluating justifications to determine the status of the nodes while checking justifications and contradictions). Another type of TMS was proposes by Kleer, [33]: assumption-based truth maintenance systems (ATMS). ATMS do not evaluate justifications. Instead, labels each datum with the corresponding sets of assumptions, which represents the contexts under which it holds. The assumptions sets are computed by the ATMS from the justifications supplied by the problem solver. This way, beliefs can be derived and used as arguments. The assumptions are primitive data, and the other data can be derived from them. Consistency is not essential in the overall database.

4.1.2. BELIEF REVISION AND EPISTEMOLOGY

In a “derivational approach”, [34], Pollock and Gilies have studied the dynamics of a belief revision system considering relations among beliefs and tried to derive a theory of a Belief Revision from a more concrete

epistemological theory. They claim one of the goals of Belief Revision is to generate a knowledge base in which each piece of information is justified (by perception) or warranted by arguments containing previously held beliefs. The trouble is that the set of justified beliefs can exhibit all kinds of logical incoherencies because it represents an intermediate stage in reasoning. Consequently, they proposed a theory of Belief Revision concerned with warrant, instead of justifications. The set of warranted propositions takes account of all possible inferences, so there is only one way to acquire new warranted propositions, which is through perception.

4.1.3. Deductive Explanations and Belief Revision

In [35], Fallapa *et al.*, based on the use of explanations, presented a non-prioritized revision operator. The main idea is that an agent must request explanations that support new inconsistent information (with respect to its knowledge) before incorporating it. The authors build on the classical distinction between *explanandum* (final conclusion) and *explanans* (sentences that support conclusion). The structure of an explanation is similar to the structure of a deductive argument. The difference resides in the fact that every belief of an explanation is undefeasible whereas some beliefs of an argument may be defeasible. Every explanation contains rules and factual knowledge. If the sentences in the explanans are more plausible than the sentences in the belief base, then the explanation is incorporated. So, explanations supporting beliefs are used for the change process. They consider kernel and partial meet revision by a set of sentences and gave representation theorems for them. The operators may partially accept the new information, so they are non-prioritized.

4.1.4. Data-oriented Belief Revision

Paglieri and Castelfranchi, in [36], joined Argumentation and Belief Revision in the same framework, following the Toulmin's layout of Argumentation. The connection to Belief Revision is made by considering Argumentation as "persuasion to belief", so Argumentation is supposed to initiate successful revision process. *Data-oriented Belief Revision* (DBR) was proposed as an alternative to the AGM model. Data and beliefs are the two basic informational categories, to account for the distinction between pieces of information that are simply gathered and stored by the agent (data), and pieces of information that agent considers truthful representations of states of the world (beliefs). Beliefs are a subset of data: this means that an agent might well be aware of a datum that he does not belief (not reliable enough). Data structures are networks of nodes (data), linked by the relations of support, contrast and union. Data can be selected or rejected based on their properties (relevance, credibility, importance and likeability). The union of data and warrant supports the claim, and the warrant is supported by its backing and contrasted by the rebuttal. Hence, rebuttal's support makes the warrant less reliable.

4.1.5. Prioritized Revision by Arguments

In [37], Rostein *et al.*, proposed an abstract theory, in which the dynamics of an argumentation framework is captured through the application of Belief Revision concepts. A *dynamic argumentation theory* is defined

including dialectical constrains. They presented argument revision techniques to describe the fluctuation of the set of active arguments. The expansion, contraction and revision operators are taken into account in the framework. Revision can be expressed in terms of expansion and contraction. The theory allows the introduction of an argument, and ensuring it can be believed later. Expansion operator is direct but the contraction operator permits more possibilities: from affecting any number of arguments in the system to keep the perturbation to minimum (AGM minimal change principle). Moguillansky *et al.*, in [38] instantiated these change operators to DeLP. A *warrant-prioritized argument revision operator (WPA)* that implements successful change is defined. When a program is revised by an argument $\langle A, \alpha \rangle$, the revised program will be such that A is an undefeated argument and α will be warranted. The main problem of the revision operator lies in the selection of arguments and the incisions that have to be made over them. A criterion for argument selection determines which arguments should not be present, and when selection is made, incisions will make those arguments "disappear" according with some minimal change principle.

4.1.6. Relating Reinstatement and Recovery

In [39], Boela *et al.*, tried to link directly Argumentation and Belief Revision on the level of abstract properties. They consider Argumentation as persuasion to belief and persuasion should be related to Belief Revision. A relation between *reinstatement and recovery* is established. Reinstatement refers to the situation that an argument that is not accepted because of the existence of an attacking argument become acceptable again when an attacker to the attacking argument exists. According to recovery, expansion by α should recover what was lost when α was contracted (AGM minimal change principle).

4.2. Conceptual view

Falappa *et al.*, in [32], presented a conceptual view on Argumentation and Belief Revision and possible connections between them. First, they went through the basic steps of reasoning. It is assumed that the current epistemic state is given and represented within some framework.

- *Receiving new information*: New information I may come in different shapes and forms. I can be a simple propositional fact (scope of basic AGM theory) or might be more complex and come with a degree of plausibility or have the form of a rule or be a complete argument or be a set of such entities,
- *Evaluating new information*: To process I, it is crucial to know its origin, since this knowledge will influence the willingness to adopt I. If I is based on observation made by the agent, so usually the agent will be convinced that I is true. But I might come from another agent, be part of official news, in personal communication or found as written material. Hence, the agent will require justification for I. Then, based on its beliefs, the agent will evaluate I and the justification in order to decide if incorporates I or not.
- *Changing beliefs*: If I is adopted, the agent must apply strategies to incorporate I consistently into its beliefs, i.e. use Belief Revision techniques to change the epistemic state.
- *Inference*: From the new epistemic state, the agent derives plausible beliefs.

These steps can also be applied to queries to which the agent is expected to reply.

While Argumentation can contribute to the evaluation step, Belief Revision can be applied in the belief change part. But it is not so simple. The evaluation can include hypothetical change process, considering what would happen if the new information were to be believed, and belief change relies on logical links between pieces of information that can be represented by arguments. Plausible beliefs can be obtained from both Argumentation and Belief Revision processes. However they focus only on parts of the dynamic reasoning process. Revision operators can be applied to beliefs and also to intentions, preferences, theories, ontologies, etc. Argumentation can be used for *negotiation, inquiry, deliberation, information seeking*. This reflection shows that the view on Argumentation and Belief Revision is complex and very interrelated. Until now, the proposed frameworks on either side might only implement some (not all) aspects of the corresponding field.

4.2.1. Comparing Both Disciplines

As stated in [32] by the authors, at first sight the differences between Argumentation and Belief Revision prevail. A good example is the syntactic and semantic foundations of both areas. For knowledge representation, in standard Belief Revision logical formulas are used. In more advanced frameworks, classical logical semantics is used. On the other hand, Argumentation theory focuses on the interactions of arguments as pieces of information that may attack one another (relations between arguments may give priority to one or another argument). The arguments are very heterogeneous, from complex argument structures to abstract objects without internal structures (Dung's framework).

Belief Revision and Argumentation, aim to resolve conflicts, usually based on logical grounds (contradictions), and to do it they make use of preference relations. Although Belief Revision theory provides a highly declarative framework (based on postulates), Argumentation theory is more concerned with practical, justification techniques.

Taking the standard AGM approach, its postulates offer a clear framework that makes fundamental views explicit. However the assumption that AGM theory focuses on deductively closed set of formulas (representing belief sets), does not fit to argumentation theory. So, the works on belief base revision or iterated epistemic state offer a much richer base for comparison, since they allow one to take deeper insights into change processes.

In Argumentation, the approaches that are based on rules, or some sort of derivation, provides logical links between what is presupposed and what is concluded. In this point, Belief Revision remains on an abstract level, describing by axioms *what* good inferences are. In turn, Argumentation is concerned with *how* and *why* conclusions are drawn.

As the authors claim in [32], the above considerations make obvious that Belief Revision and Argumentation theory are complementary areas.

4.2.2. Argumentation in Belief Revision

Falappa *et al.*, in [32], studied how some concepts and techniques from Argumentation can be used in Belief Revision theory. The first approach was the *justification-based truth maintenance systems* (TMS) from Doyle, [15]. Doyle studied the interactions between justifications when a new justification has been added, in order to find out which conclusions can be justified. The *Assumption-based truth maintenance systems* (ATMS) from Kleer, [33], are more focused with maintaining assumptions instead of implementing change processes. As in DBR, from Paglieri and Castlefranchi, [36], we have data and beliefs. A data is believed in some contexts represented by assumptions sets and it is not necessary to be consistent the entire database. It is easy to refer to contexts and move to different points in the search space representing the proper context. The TMS and ATMS works on belief bases are very early approaches to Belief Revision.

In [35], the authors combined base revision with the ATMS idea and proposed a system that uses argumentative structures in the form of explanations for non-prioritizes revision of belief bases K. The new information consists in a proposition α and the reasons to believe a proposition (rules and prerequisites A from which α can be deductively derived). A can be considered an explanans for the explanandum α. To integrate an argumentative evaluation of the new information into the revision process, the authors defined *partial acceptance revision operators*, following the steps:

- The epistemic input is the set of sentences A as explanans for the explanandum α,
- A is joined to K (possibly producing an inconsistent intermediate state),
- All possible inconsistencies of $K \cup A$ are removed, returning a consistent revised belief base $K \circ A$. This operator is an operator of *external revision* (the revision process takes place outside of the original set).

Whether α is accepted or not depends on the evaluation of its explanans A according to the current belief base. The acceptance of the explanans forces the acceptance of the explanandum in the revised set, (explanation is based on classical deduction). However, while the explanans can be explicitly included in the revised set, the explanandum may be inferred from it without actually being included. So, the distinction between explicitly given information and inferred beliefs is respected and can be implemented only when working with belief bases instead of belief sets.

The authors claim that there is a big difference between the process of revision by a set of sentences and the process of argumentation: in revision, external beliefs are compared with internal beliefs and, after a selection process, some sentences are discarded and others are accepted; besides, in Argumentation the process is more procedural: an argument, is attacked by counterarguments, defend it by counterarguments to counterarguments, and so on. The rationale behind partial acceptance revision operators matches the intentions leading argumentation in that the reasons to believe in new information are evaluated (not the new information itself).

4.2.3. Belief Revision in Argumentation

Methods from Belief Revision theory can implement dynamical features in an argumentation framework, as stated in [32]. Works from Rotstein *et al.*, [37], and Moguillansky *et al.*, [38] are good examples of

comprehensive approaches to address such a revision theory for AS. It is possible to distinguish different ways of applying Belief Revision in Argumentation, [32]:

- Changing by adding or deleting an argument or a set of arguments.
- Changing the attack (and/or defeat) relation among arguments.
- Changing the status of beliefs (as conclusions of arguments).
- Changing the type of an argument (from strict to defeasible, or vice versa).

The difference between adding/deleting an argument or set of arguments is similar to the distinction among single change (as in AGM model) and multiple change (as in multiple contraction, [40]). These changes may trigger a change in the justified conclusions and that may lead to a base revision theory which deals with changes of argumentative bases, and in which deduction is replaced by an argumentation process. Note that the method of kernel contraction to eliminate base elements (using incision functions) has particular interest, [38].

Changing the attack/defeat relation among arguments may result into a different behavior of the system. The understanding and the control of which constitutes substantially new challenges for Belief Revision theory. In [41], Boella *et al.* have proposed an approach that changes the attack relation in a dictatorial way in favor of the new information. It is possible to conceive more complex change processes using the ideas from epistemic belief change that deal with modifying relations on possible worlds. Arguments are distinct from possible worlds, but those might be considered as approaches to realize minimal change processes for general relations.

The change of the status from beliefs may be a consequence of changes in the argumentation system. As stated in [32], in argumentation frameworks where argumentation is based on conclusions, like DeLP [19], the addition of an argument may change the status of a claim (from unwarranted to warranted and vice versa). Basic concerns of the reasoning aspect of Belief Revision can be considered, and investigations on the level of Belief Revision postulates might be very useful. A good example is the relation between reinstatement (in Argumentation) and recovery (in Belief Revision) proposed in [39].

At last, changing the type of an argument from strict to defeasible, or vice versa, addresses novel issues in Belief Revision. The idea is that inconsistencies that arise when new information has to be incorporated into the stock of beliefs can be eliminated not only by removing arguments (resp. beliefs), but by turning strict beliefs into defeasible rules (resp. conditionals). This possibility, of changing the status of beliefs, induces revision operations of a new quality, with important consequences for argumentation, as arguments are formed very often by defeasible beliefs. Fallapa *et al.*, in [35], have introduced a new type of base revision that implements a dynamic classification of beliefs and is likewise interesting for argumentation and Belief Revision theory. They proposed a framework in which defeasible conditionals can be generated by revising belief structures composed of defeasible rules and undefeasible knowledge. An advantage of this framework is that preserves consistency in the undefeasible knowledge and it provides a mechanism to dynamically qualify the beliefs as undefeasible or defeasible, providing a more complete set of epistemic attitudes and extending the inference power of knowledge based systems. Falappa *et al.*, in [42], proposed to extend the application of this non-prioritized revision operator to DeLP [19].

5. THE BELIEF-DESIRE-INTENTION MODEL

The Belief-Desire-Intention (BDI) model provides a promising architecture for the development of intelligent agents. This model was proposed by Bratman in 1987, [43], as a philosophic theory for practical reasoning to explain the human behavior using beliefs, desires and intentions as mental attitudes. As explained in [44] by Nunes, The basic assumption of the BDI model is that actions are derived from a process called practical reasoning, which consists in two steps, according to Wooldridge, [45]:

- First step is the deliberation of goals, i.e. it is the selection of a set of desires that must be achieved, according to the current situation of the agents' beliefs.
- The second step is responsible for determining how these desires can be achieved through the use of the available resources.

The first step is known as *deliberation*, and the second is known as *means-ends reasoning*.

Next, a brief description of the three mental attitudes is presented, [44]:

- **Beliefs**: Represent characteristics of the environment, which are updated appropriately after each action or by perception. Can be seen as the informative component of the system (agents' knowledge).
- **Desires**: Represent the goals to be achieved. Can include priorities and the costs associated with the various goals. Can be seen as a representation of the motivational state of the system.
- **Intentions**: Represent the current action plan chosen. Captures the deliberative component of the system. Note that an action plan may include other plans.

In 1995, Rao and Georgeff, [46], adopted the BDI model for software agents presenting a formal theory and an abstract BDI interpreter that is the basis for almost all BDI systems:

BDI Interpreter
initialize-state();
repeat
    options: option-generator(event-queue);
    selected-options: deliberate(options);
    update-intentions(selected-options);
    execute();
    get-new-external-events();
    drop-unsuccessful-attitudes();
    drop-impossible-attitudes();
end repeat

The interpreter operates on the beliefs, plans and goals of the agent. The main difference is that the goals are a consistent set of desires that can be achieved together, thus avoiding the need for a complex phase deliberation of goals. The main task of the interpreter is the realization of the means-end process through the selection and implementation of plans for a specific purpose or event. This interpreter has been extended in many ways along the years.

5.1. ARCHITECTURE

In [47], Wooldridge presented a generic BDI architecture, Figure 16, which represents the practical reasoning of an agent.

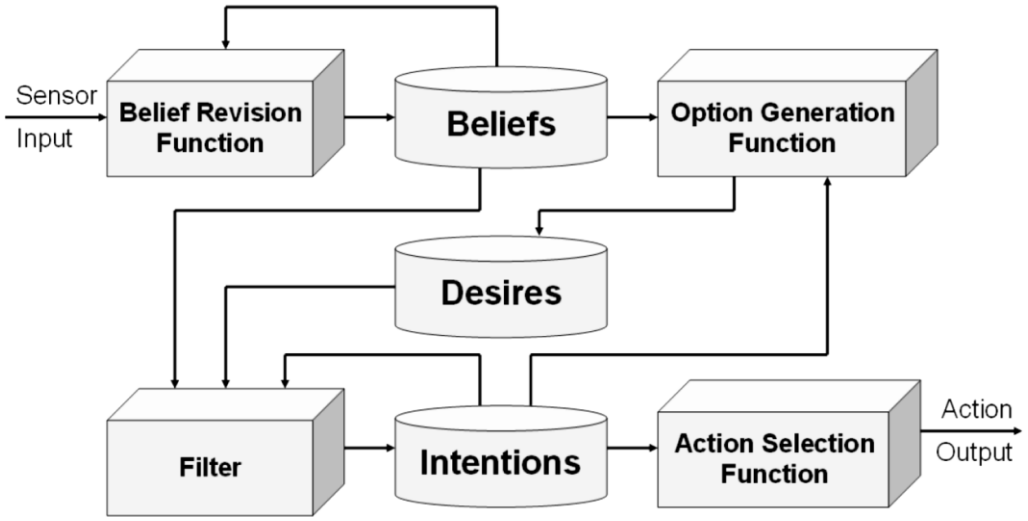


FIGURE 16 - GENERIC BDI ARCHITECTURE

As Figure 16 shows, the architecture of a BDI agent is composed by seven components:

- a set of *beliefs*, representing the information that the agent has about its environment;
- a *belief revision function*, which determines a new set of beliefs based on the perception of the input and the beliefs of the agent;
- an *option generation function*, which determines the options available to the agent (desires), based on its beliefs about the environment and its intentions;
- a set of options (*desires*) that represents the possible action plans available to the agent;
- a *filter* function, which represents the agent's deliberation process that determines the intentions of the agent based on their beliefs, desires and intentions;
- a set of *intentions*, which is the current focus of the agent, i.e. those states that the agent is determined to achieve;
- an *action selection function*, which determines an action to perform based on their current intentions.

A simple pseudo-code for a software agent loop was proposed by Wooldridge, which is similar with the following:

```
while true
        observe the world;
        update internal world model;
        deliberate about what intention to achieve next;
        use means-ends reasoning to get a plan for the intention;
        execute the plan;
end while
```

It is perfectly possible to implement complex deliberative agents by using a pseudo-code like this as a basic agent loop structure and with it the BDI framework.

Between the main criticisms and limitations of the model, we can mention the following points:

- **Learning**: BDI agents lack any specific mechanisms within the architecture to learn;
- **Multiple Agents**: BDI model does not explicitly describe mechanisms for interaction with other agents and integration into a multi-agent system;
- **Explicit Goals**: Most BDI implementations do not have an explicit representation of goals.

Finally, as stated in [42], the basic BDI model needs to be complemented with two mechanisms: one for reasoning about intentions and other for revising beliefs upon perceptions.

5.2. BDI IMPLEMENTATIONS

It is possible to find a number of implementations for the BDI architecture. These implementations are frameworks that enable the development of agents in a given language and provide a platform for implementing them, [44]. Next we will briefly address some implementations, based on [44].

5.2.1. JACK

JACK Intelligent Agents™, [48], is a framework in Java for the development of multi-agent systems. It was built by Agent Oriented Software Pty. Ltd., headquartered in Melbourne, Australia. The framework offers high performance, a lightweight implementation of the BDI architecture and an easy way to be extended to support different models of agents and specific requirements of applications. The language used by JACK (JACK Agent Language) was constructed from the Java language and extends the functionality of Java. Also supports the new programming paradigm, which is agent oriented programming.

5.2.2. JADEX

JADEX, [49], is a reasoning mechanism that is agent oriented in which rational agents are written in XML and Java. One of the main aspects of Jadex is that it does not present a new programming language. Instead, Jadex agents can be programmed in integrated development environments that are object oriented. Another important aspect is the independence of Jadex, which can be used with different platforms of agents. Jadex follows the BDI model, using beliefs, goals and plans as first class objects, which can be created and manipulated inside the agent. In Jadex, agents have beliefs that are stored on the belief base. Objectives represent specific motivation, such as, states to be achieved and influencing the behavior of the agent. To achieve its goals, the agent executes plans, which are procedural scripts coded in Java.

5.2.3. JAM

JAM, [50], is an intelligent agent architecture that was based on a number of theories about agents and frameworks for intelligent agents. It was mainly influenced by Bratmans's BDI theory. Each JAM agent has five primary components: a world model (database that represents the beliefs of the agent), a plan library (a collection of plans that the agent can use to achieve its goals), an interpreter (can be considered the brain of the agents because it is this component that reasons about what the agent should do and when), an intention structure (an internal model of goals and current activities of the agent) and an observer (can be considered a light plan which runs between the steps of the plan in order to perform certain features that are not in the normal course of the plan).

5.2.4. JASON

Jason, [51], is a platform for developing multi-agent systems based on an interpreter for an extended version of the language AgentSpeak(L), which is an agent oriented programming language based on first-order logic with events and actions. It is inspired by the BDI logic and architecture. In a program written in AgentSpeak, beliefs, desires and intentions of the agent are not explicitly represented as modal formulas. Instead, the developer must use the notation of language. The current state of the agent, which is a model of itself, its environment, and from other agents, can be seen as beliefs. Motivational states that the agent may want to achieve, based on external and internal stimuli, can be seen as desires. The choice of planes that satisfy a given stimulus can be seen as intentions.

5.2.5. 3APL

3APL is an abstract agent programming language. As stated in [52], 3APL is a programming language for implementing cognitive agents. It provides programming constructs for implementing agents' beliefs, goals, basic capabilities (such as belief updates, external actions, or communication actions) and a set of practical reasoning rules through which agents' goals can be updated or revised. The 3APL programs are executed on the 3APL platform. Each 3APL program is executed by means of an interpreter that deliberates on the cognitive attitudes of that agent.

6. ARGUMENTATIVE NXT BDI-LIKE AGENTS

The use of Argumentation and Belief Revision combined has been recently studied. In section 4, it is possible to observe how some investigators see this combination and in [32] the perspective of complementarily between them is supported by the authors. Following this line of cooperation among these two disciplines, this project intends to use both Argumentation and Belief Revision as parts of an agent's architecture.

As said in section 5, the BDI model provides a very promising architecture for the development of intelligent agents. Using the principles of the BDI model, we will develop the architecture for a deliberative and argumentative agent. We will add to the architecture a mechanism to reason about intentions and a mechanism to revise beliefs.

A programmable robotics kit called Lego Mindstorm's NXT, will be used to develop the agents functionalities described above.

In this section we will explore the agents' concept and architecture. A general flowchart about its behavior will be presented as well as the Belief Revision criteria.

6.1. CONCEPT

The BDI model presents three mental attitudes (beliefs, desires and intentions), in which we will base our agents' architecture. As said previously in section 5, the intentions are intimately related to the action plans. However, the BDI model provides a mechanism to select a plan separated from the execution of that plan. The purpose will be the implementation of such mechanism in the agents.

Additionally, the agents must be capable to reason about its intentions and check which of them are warranted. For that purpose we will use defeasible argumentation, in particular the formalism presented at the case study in section 2 – DeLP. This formalism will reason about the agents' intentions using the available knowledge. Still, it will be able to detect contradictions in the beliefs' set and must have a mechanism capable to revise it, section 3.

Considering the BDI model and the defeasible argumentation used to reason, we decided to name the agents as "Argumentative NXT BDI-Like Agents". Two types of agents will be distinguished. One is the "Single Argumentative NXT BDI-like Agent" and the other is the "MAS Argumentative NXT BDI-like Agent". It is important to mention that the main difference between them lies in the reasoning aspect, because in the MAS Agent the knowledge of the other agents will be taken into account.

For instance, the MAS agent will have a cooperative sense. Then, the reliability problems between agents will not be present. Nevertheless, they are very important in MAS agents that have the competitive perspective.

6.2. AGENTS' ARCHITECTURE

Based on the BDI architecture and pseudo-code presented by Wooldridge, see section 5, we will introduce the architecture for the Argumentative NXT BDI-like Agent. The DeLP formalism will be the base of the agents' knowledge and procedure. The beliefs of the agents will be composed by DeLP formulas, i.e. the agents' knowledge will be facts (literals), strict rules and defeasible rules, (Π, Δ). In the Belief Revision context, the knowledge of the agent will be considered a belief base, because it will not be closed under logical consequence.

The agents' intentions will be a set of chronological ordered literals, i.e. ground atoms or negated ground atoms ordered by possibly sequential actions that may be taken by the agents.

At Figure 17, it is possible to observe a global perspective from the agents' architecture. All of the agents' perceptions from the environment are considered new beliefs, assumed by the agents and introduced in the belief base. The set of intentions contain the possible motivational states of the agents. The action plans are closely related to the intentions, i.e. when an intention is warranted, it turns out the execution of the respective action plan. Note that the agents' desire is not explicit, and it is represented by the set of intentions.

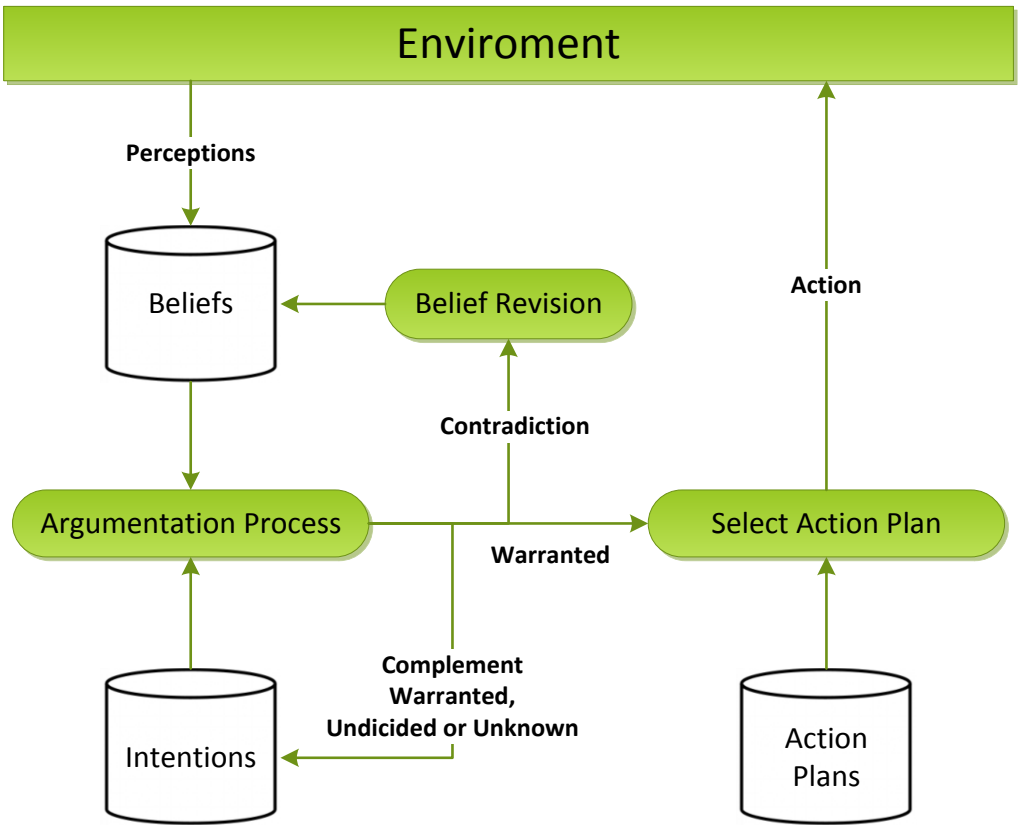


FIGURE 17 - AGENTS' ARCHITECTURE

Figure 17 shows three main processes: *Argumentation Process*, *Belief Revision* and *Select Action Plan*. The *Argumentation Process* is responsible to select the next intention from the intentions' set. Through the DeLP formalism and considering the beliefs' set, the process will be responsible to reason about the intentions' warranty. An intention will be executed if and only if it is warranted. The *Belief Revision* process is activated when the set of beliefs is considered contradictory, by the argumentation process. At last, the *Select Action Plan*

process occurs when an intention is warranted by the argumentation process. It will be responsible to select the intention's correspondent action plan and then execute it.

Using this architecture, the agents will perform always in the same way, independently of the beliefs, intentions or action plans. Those sets can be constructed according to different situations. Later, we will show some application cases as examples.

6.3. AGENTS' BEHAVIOR

The flowchart from Figure 18, graphically describes the general behavior of the agents (Single and MAS). As said before, the agents' desire is implicit by the intention's set. Although the scheme appears to be simple, the sub-processes "Revise Beliefs", "Choose intention" and the decision point after that selection are not, as we will see later.

To achieve the implicit desire it is necessary to follow the intentions' set, represented by a list of chronological ordered literals. First, the agents will check the list of intentions. If it is empty, the initial intentions will be restored, if not, it will be chosen the subsequent intention present on the list.

The next step is to verify if the chosen intention is warranted. Here lies the reasoning point, by warranting the selected intention. As seen before, the DeLP formalism is capable to warrant a literal (intention), or its complement. Then, the following situations may occur: the intention is warranted; the intention is not warranted (meaning that its complement is warranted); the formalism cannot decide (neither the intentions is warranted nor its complement); the formalism does not recognize the intention (very unlikely to happen).

If the intention is warranted, a set of actions (action plan) will be performed, in order to satisfy the intention. On the other hand, if the intention's complement is warranted or neither of them is warranted or it is an unknown intention, then the actual intention will be discarded, and the agents will check for other intentions.

Finally, if the agents realize that the belief's set is contradictory, they revise their beliefs, in order to maintain the consistence.

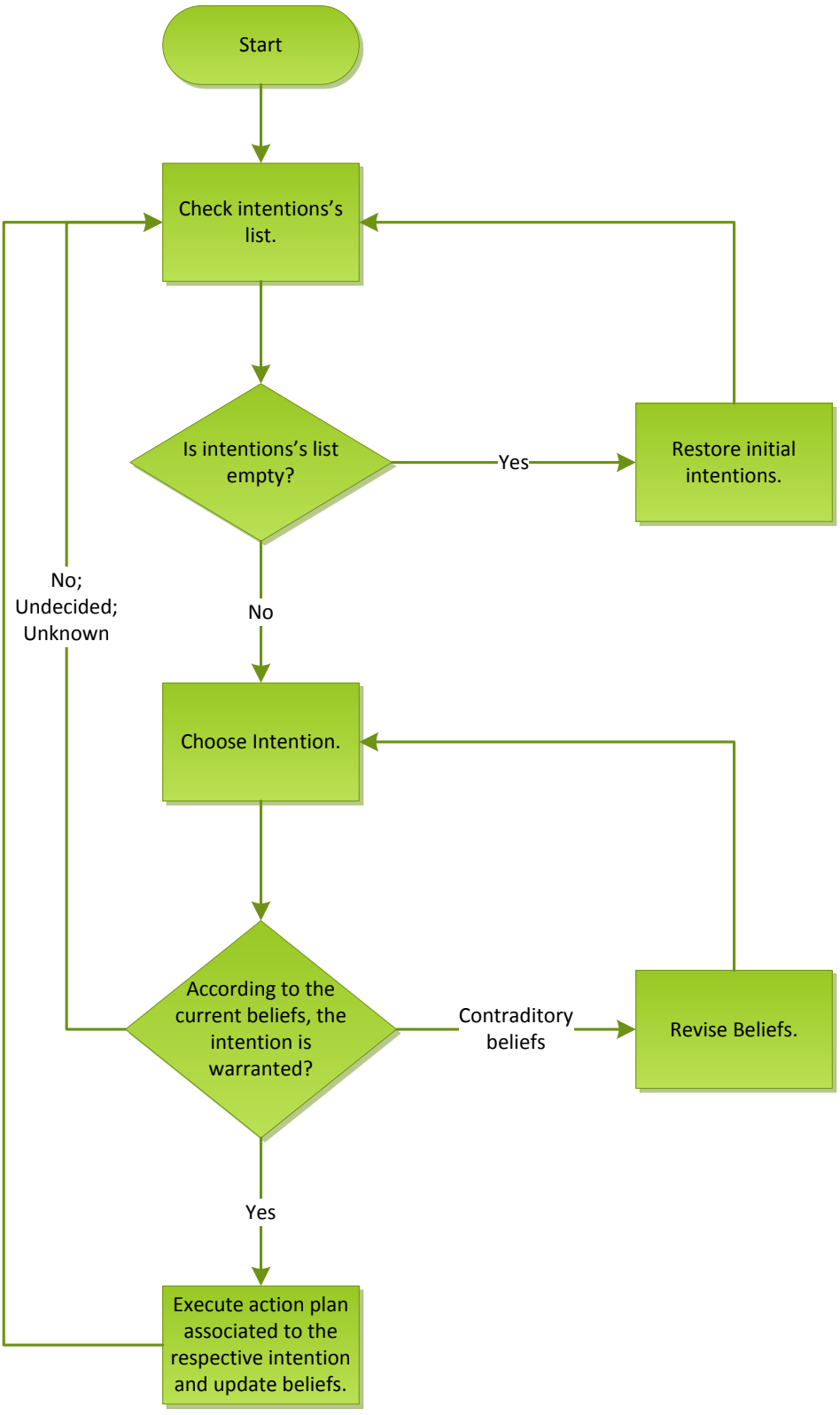


FIGURE 18 - GENERAL AGENTS' BEHAVIOR

6.4. REVISION CRITERIA

As seen in section 3, the Belief Revision field provides different ways to do revision. In this case we are working with belief bases. In [28], *Kernel Contraction* was introduced as a generalization of safe contraction. Kernel contraction implies to form all the minimal sets of beliefs that derive a certain sentence. Then, with an incision function a cut can be made to a one of those minimal sets.

This type of contraction is the base for the implemented belief revision. An important aspect is that agents will only remove facts from their beliefs. This makes sense, since they only receive new information as facts. The defeasible and strict rules will be given at the agents' configuration. Contradictions occur over the literals, i.e. it will occur over ground atoms conflicts, as the example below show.

The adopted strategy to perform revision is based on time, i.e. if the belief base has contradictions the first step is to isolate the contradictions and eliminate those beliefs that are older. With this approach it is clear that new information has priority over the older, following the first rationality criterion from AGM model.

Example:

Let *K* be a set of sentences and "*traffic_road1*" and "*~traffic_road1*" two contradictory sentences. After discovering which is the oldest, Kernel contraction should occur, throwing away the oldest one.

$$K = \{rush\_hour -> traffic\_road1, traffic\_road1, \sim traffic\_road1\}$$

Let "$traffic\_road1$" be the oldest one:

$$K \Vert traffic\_road1 = \{\{traffic\_road1\}\}$$

A possible result of $K - traffic\_road1$:

$$\{traffic\_road1\} \qquad \sigma(K \Vert traffic\_road1) = \{\}$$

$$K = \{rush\_hour -> traffic\_road1, \sim traffic\_road1\}$$

In this project we are dealing with external revision, see section 3, because the perceptions (facts) will be added to the belief base and the revision will occur after that. This means that in some moment we will be in an intermediate inconsistent state, believing in contradictory sentences. As we can see in the previously example, $traffic\_road1$ and $\sim traffic\_road1$ are believed, and the revision process will occur later.

This methodology associated to the "Single Argumentative NXT BDI-like Agent" is acceptable because all the new beliefs came from the agent itself. But for the "MAS Argumentative NXT BDI-like Agent" this method can raise doubts, so it is important to remember that this project has a collaborative perspective. The information that comes from other agents is as trustable as the information collected from the agent itself.

7. DEVELOPMENT AND IMPLEMENTATION

In this section, the practical work developed to implement the concept presented in section 6 will be described, as well as the used resources. We will show the primary differences between the Single Argumentative NXT BDI-like Agent and the MAS Argumentative NXT BDI-like Agent through the description of each one of them.

7.1. RESOURCES

The agents were constructed to be used with LEGO Mindstorms NXT, which is a robotic kit used in the AI course lectured in the University of Madeira. The programming language used to build the agents' architecture was prolog. Droide M. L. P. was the chosen platform to perform the integration of prolog with the robots. The DeLP-Server, presented in [53], is used to give the agents the ability to reason. Next, we will take a brief look into these resources.

7.1.1. LEGO MINDSTORMS NXT

LEGO Mindstorms is a robotic line of kits developed for educational purpose. The LEGO Mindstorms project started with a partnership between Lego and MIT Media Laboratory. This project combines the traditional LEGO line of toys and a more specific line of technological pieces, such as motors, sensors and programmable blocks. Until now, three main kits were launched: RCX; Lego Mindstorms NXT; Lego Mindstorms NXT 2.0 (mainly differ from the previous by the use of Floating Point operations instead of Integer, and by the introduction of a color sensor). For this project, it is used the Lego Mindstorms NXT. This kit is composed by several parts. A brief description of the main parts will be made, see Figure 19:

- **NXT Brick** – this part is considered the brain of Mindstorms robots because it can be programmed and controlled from a computer (have a microcontroller and memory included). The NXT Brick has three ports to attach motors and four ports to attach sensors. The connection to a computer can be made through USB or Bluetooth. The brick is also equipped with a display, a speaker and four buttons.
- **Ultrasonic sensor** – this sensor is able to measure the distance to a given object, until 255cm.
- **Sound sensor** – this sensor measures the sound level up to 90dB.
- **Light sensor** – this sensor is able to measure the light intensity of a nearby surface.
- **Touch sensor** – this sensor detects if it is being pressed or released.
- **Motors** - the motors have a built-in rotation sensor and they operate with different speeds and are able to synchronize with other motors.

FIGURE 19 - LEGO MINDSTORMS NXT MAIN PARTS

For more information about the Lego Mindstorms NXT consult the official website, [54]. We have chosen these robots, because they do not require electronics background and its construction can be very creative and easy.

7.1.2. DeLP-Server

The DeLP-Server, described in [53], uses defeasible logic programming to answer queries from agents, who are connected to the server. The idea brought by this paper is to help agents deliberate. Note that the server can accept connections from many agents, and each agent can connect to other servers, as Figure 20 shows. It is observable the Multi-agent System (MAS) created by the servers. However, the agents do not interact with each other.

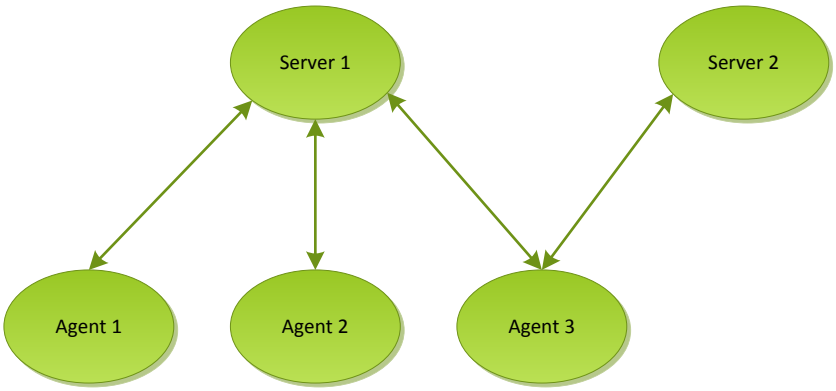


FIGURE 20 - DELP-SERVERS AND CLIENTS

The DeLP-Server can store public knowledge, which is shared with the connected agents. However, that knowledge is static, i.e. the possible changes that can happen cannot be updated to the server. This public knowledge is a DeLP-program, which is loaded when the server is launched.

Agents are considered deliberative, since they are able to reason through the submission of queries to the server, DeLP-queries. Two sources of knowledge can be considered for the answer: the knowledge stored at the server (DeLP-program); and the private knowledge from the agent. The private knowledge from the agent, which generally arises from its perceptions, is sent jointly with the query, to the server ("contextual query").

The DeLP-Server can return five different answers to the agent:

- YES: if the query is warranted;
- NO: if the complementary query is warranted;
- UNDECIDED: if neither the query nor the its complementary is warranted;
- UNKNOWN: if the query is not present in the DeLP language;
- INVALID: if the contextual set is contradictory, i.e. the agent's private knowledge sent has contradictions.

The DeLP-Server is a stand-alone program developed in prolog. It can run in a host and the agents can be connected to it, through the TCP protocol. We have chosen the DeLP-Server because implements the DeLP formalism, which is the elected framework to implement defeasible argumentation into the agents.

7.1.3. SWI-PROLOG

The SWI-Prolog is an open source implementation for the prolog programming language. It is very versatile, since it runs on different platforms, such as UNIX, Macintosh and Windows. This implementation has many features attached to it, like multithreading, GUI, Java interfacing, libraries and others.

It is important to refer that the prolog language it is related to the Artificial Intelligence field since it is based on first-order logic. With prolog it is also possible to define operators and relations. By this way it is possible to represent facts and rules. For more information about SWI-Prolog please consult the official website, [55].

We have chosen the SWI-Prolog implementation because it is very flexible and it is based on formal logic.

7.1.4. DROIDE M.L.P.

The Droide M.L.P. platform intends to be an auxiliary tool to program the LEGO Mindstorms NXT robots. The core of the platform is called NXT Software Development Kit (NXT S.D.K). It has a full set of libraries created to support six programming languages, in which the NXT brick can be configured and programmed.

This platform supports the prolog programming language, however it is necessary the installation of a separate module. It is important to refer that the firmware of the NXT brick needs to be changed to the pblua firmware. All the steps for this configuration can be found at [56] as well as the documentation, and the software can be downloaded at [57].

The choice of the Droide M.L.P. platform is obvious, since it makes possible to program the robot in prolog.

7.2. SINGLE ARGUMENTATIVE NXT BDI-LIKE AGENT

Based on the architecture presented in section 6, the Single Argumentative NXT BDI-like Agent intents to implement the concept described in that section, using the resources mentioned above.

The particularity of this first implementation is related with the nonexistence of other agents. It is an acting alone agent and only its world, beliefs and perceptions will count. However, the use of the DeLP-Server brings the possibility to use public static knowledge. This knowledge must be strictly true because it cannot be modified. The unique knowledge that the agent can modify is its belief base by revising contradictory beliefs.

7.2.1. BEHAVIOR

From the general Argumentative NXT BDI-like Agent's behavior denoted by Figure 18 and the resources previously exposed, it is possible to describe more specifically the Single Argumentative NXT BDI-like Agent's behavior.

Through Figure 21 it is possible to understand better how the agent will reason using the DeLP-Server. The first steps of the flowchart match the previous flowchart presented in section 6. However, it is possible to verify that after the selection of an intention, a contextual query will be sent to the server. That query will contain the agent's knowledge, i.e. its beliefs. Then the server will answer the query based on the public static knowledge and the agent's beliefs.

According to the answer, the agent will take some attitude. As explained before, if the answer is affirmative this means that the intention is warranted, so the agent will execute the correspondent action plan. If the answer is negative, undecided or unknown, the intention will be discarded and the next intention will be considered. If the response is invalid means that agent's knowledge has contradictions, so the agent must revise its beliefs. The revision process will occur exactly as described in section 6, i.e. it will focus on the oldest belief (fact) of the contradiction and will remove it, in this way creating a new belief base. This means the new beliefs will have priority in over of the oldest ones.

Then, the same contextual query (intention) will be submitted to the server together with the new belief base and the process will repeat.

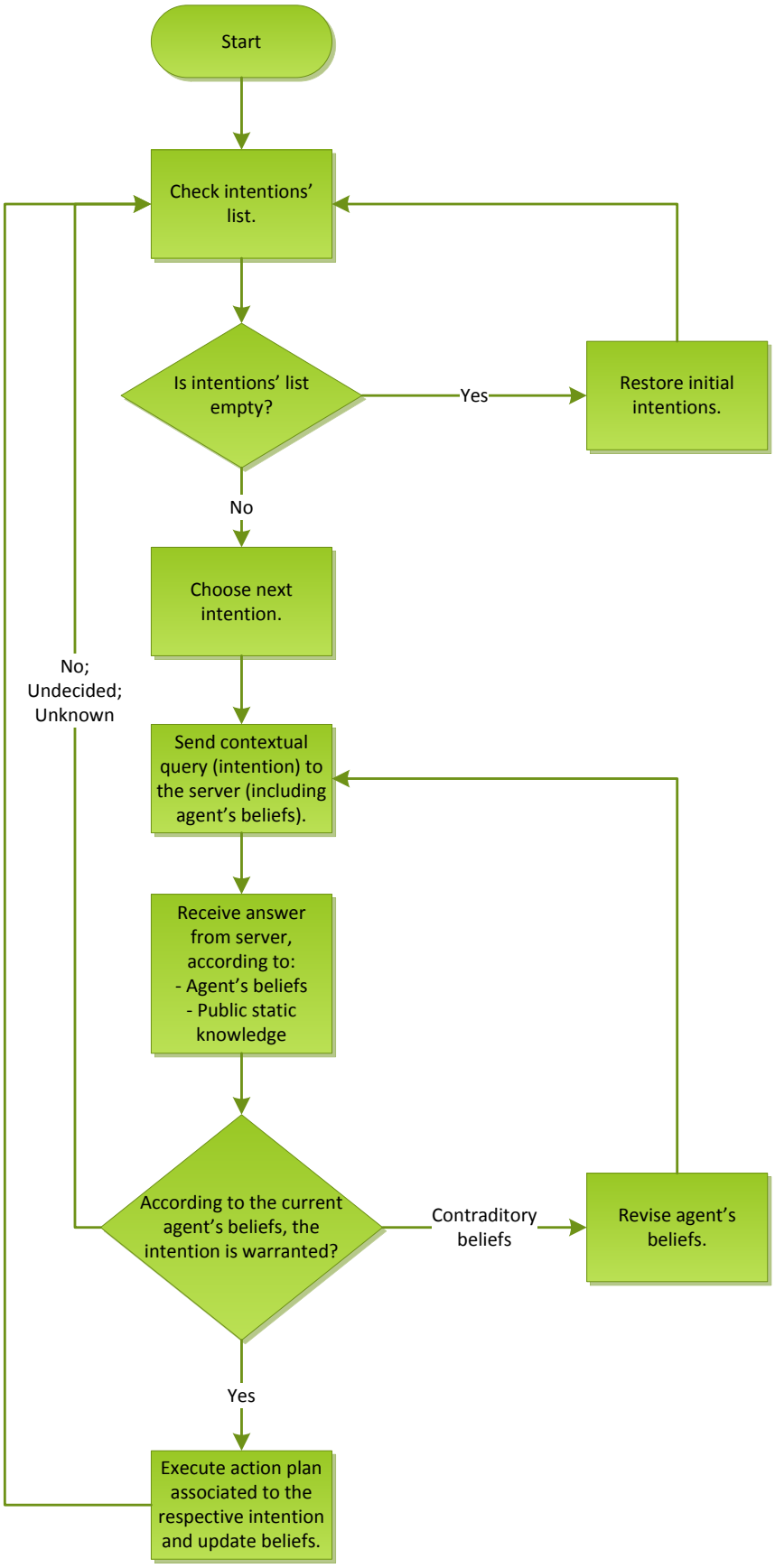


FIGURE 21 - SINGLE ARGUMENTATIVE NXT BDI-LIKE AGENT'S BEHAVIOR

7.2.2. STRUCTURE

Figure 22 shows the structure of the agent and its connection to the DeLP-Server. The DeLP-Server is represented by the "delp_server.exe". As said before, a DeLP program can be stored at the DeLP-Server. Although, it is an optional decision to maintain it empty or not. This decision will depend on the context in which the agent will act. Recall that this program is static, in other words, it is static public knowledge because once it is loaded by the server it cannot be changed.

The Single Argumentative NXT BDI-like Agent is represented by the "singleAgent.pl", which implements the architecture described in section 6 and has an algorithm that follows the flowchart from Figure 21. It is written in prolog and should not be changed.

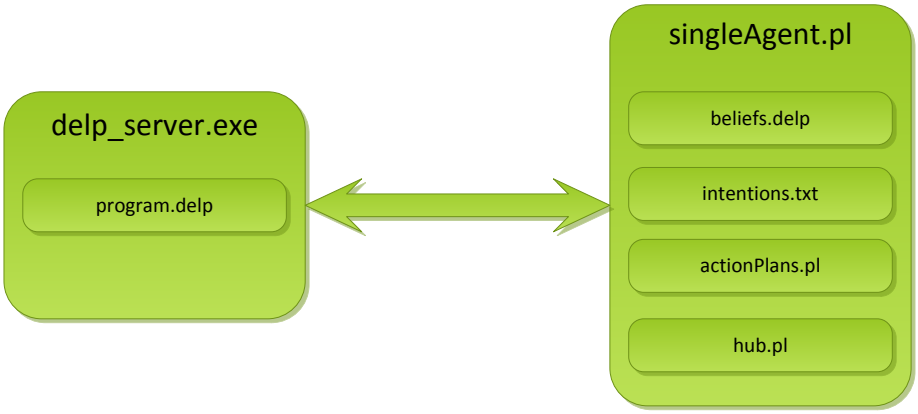


FIGURE 22 - SINGLE ARGUMENTATIVE NXT BDI-LIKE AGENT'S STRUCTURE

The DeLP-Server is maintained as originally presented by [53]. One should see that the Agent has four related main files, in which the problems can be modulated without changing the agent's architecture.

Files' description:

- **beliefs.delp** – this delp file must contain a de.l.p. and it should be consistent, as much as possible (contradictions should be avoided). This program is seen as the agent beliefs – belief base.
- **intentions.txt** – this file is a chronological ordered list of agent's intentions. This set represents possible motivational states that must be achieved to reach the overall desire. Each intention is represented by a literal.
- **actionPlans.pl** – this is a prolog file, where is described the plan of actions to perform when an intention is warranted by the server. Note that a plan can include other plans. And actions can be performed in different plans.
- **hub.pl** – this is a prolog file that has a simple function responsible to directly relate the intentions to the respective action plan. With this file it is possible to separate the mechanism to select a plan from the execution of that plan.

With this agent structure, it is possible to model problems only using the four files described above. Since the new beliefs are always added at end of file beliefs.delp, when contradictions occur it is easy to identify the beliefs that are older since they are at the beginning of the file.

7.3. MAS ARGUMENTATIVE NXT BDI-LIKE AGENT

The MAS Argumentative NXT BDI-like Agent, as the Single Argumentative NXT BDI-like Agent, is also based on the resources described above and aims to implement the concept presented in section 6.

The main difference from the Single Argumentative NXT BDI-like Agent resides, of course, at the MAS factor. The MAS Argumentative NXT BDI-like Agent should act according to its own beliefs and perceptions but also should consider the perceptions from other agents. Therefore we have created the concept of public dynamic knowledge where the agents can share knowledge.

An important aspect is the differentiation of the public static knowledge from the public dynamic knowledge. The public static knowledge, as said before, it is intrinsic to the DeLP-Server and cannot be modified. The existence of dynamic knowledge takes importance because the agents can update and collect information from the other agents.

Although the MAS Argumentative NXT BDI-like Agent follows the same architecture from Figure 18, there are certain differences comparing with the Single Argumentative NXT BDI-like Agent as regards to its design due to the introduction of public dynamic knowledge.

7.3.1. BEHAVIOR

The flowchart from Figure 23 represents the behavior of the MAS Argumentative NXT BDI-like Agent. This representation is similar to the one from the Single Argumentative NXT BDI-like Agent. The distinction lies in the fact that the MAS Argumentative NXT BDI-like Agent takes into account the public dynamic knowledge. When a query (intention) is submitted to the server, it goes along with the agent's beliefs and the public dynamic knowledge. So, the answer from the server will consider the agent's belief, the public static and dynamic knowledge.

The remaining behavior is identical to the behavior of the Single Argumentative NXT BDI-like Agent. A single note for the answer "INVALID" that could be sent by the server: this kind of answer will occur if the set of agent's beliefs joint with the set of public dynamic knowledge has contradictions. In that case, the revision process, as described in section 6, will be applied to agent's beliefs and public dynamic knowledge, in order to maintain consistency among both.

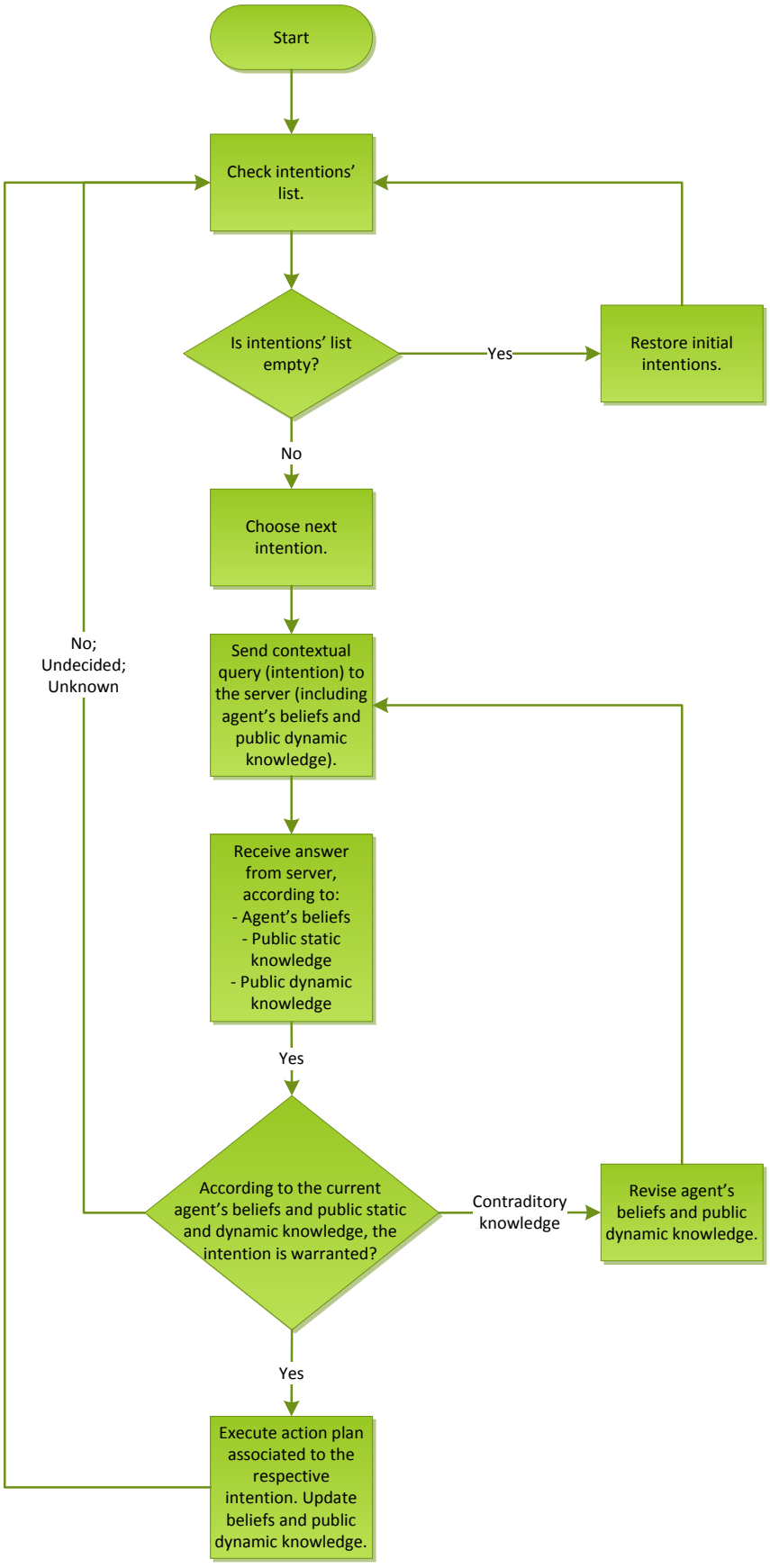


FIGURE 23 - MAS ARGUMENTATIVE NXT BDI-LIKE AGENT'S BEHAVIOR

7.3.2. STRUCTURE

Figure 24 clarifies the distinction between public static knowledge and public dynamic knowledge, as well as the relationship between MAS Agents and the DeLP-server.

The communication between agents is indirect, i.e. there is no direct exchange of information among the agents. This approach discards the use of communication protocols between the agents. All of the information that the agents consider to be important to share, must be placed at the public's dynamic knowledge file.

In Figure 24, the DeLP-Sever is represented by the "delp-server.exe". The MAS Argumentative NXT BDI-like Agents are represented by "MASAgent01.pl" and "MASAgent02.pl" and they implement the architecture described in section 6, as well as an algorithm that follows the flowchart from Figure 23. These MAS agents are written in prolog and should not be changed.

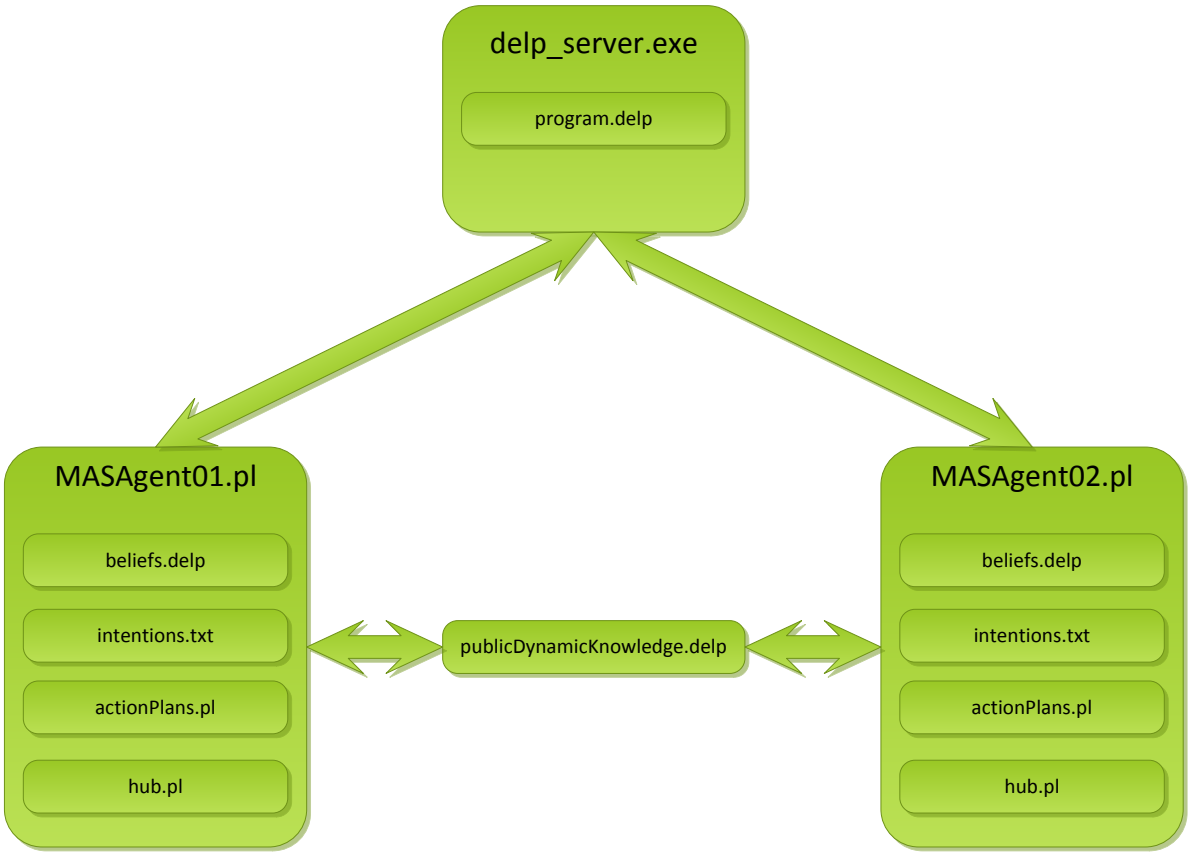


FIGURE 24 - MAS ARGUMENTATIVE NXT BDI-LIKE AGENT'S STRUCTURE

For instance, the DeLP-Server is once again maintained as originally presented by [53]. As in the Single Argumentative NXT BDI-like Agents, the MAS Agents have four related files. The descriptions of those files can be found in subsection 7.2.2.

The big difference of this structure lies in the public's dynamic knowledge file.

File description:

- **publicDynamicKnowledge.delp** – this delp file contains information shared by agents. The information must be in the form of facts. Note that this file can be contradictory since there is no control on the information deposited by the agents. Nevertheless, this file may be subject to revision by the agents as aforesaid.

Taking a look at Figure 23, it is possible to observe when an Agent sends a contextual query, it will go along with agent's belief and the public dynamic knowledge. The DeLP-Server will not distinguish the public knowledge from the agent's beliefs. Therefore, this big set of information cannot be contradictory and consequently the public dynamic knowledge will be revised every time the DeLP-Server answers a query as invalid.

As stated in section 6, the revision criterion is based on time. When contradictions occur the new information has priority over the oldest. It is not possible to adopt the same strategy of the Single Argumentative NXT BDI-like Agent to identify the oldest beliefs, since two files must be considered now: "beliefs.delp" and "publicDynamicKnowledge.delp". In this case it is necessary to attach a file containing times with respect to the beliefs. Each knowledge file must have an attached time's file that specifies when a belief has been introduced. For instance, the file "beliefsTime.txt" should be attached to the file "beliefs.delp" from each agent and "publicDynamicKnowledgeTime.txt" should be attached to the file "publicDynamicKnowledge.delp". When a belief is introduced in a knowledge file, then the current time must also be introduced in the correspondent time file.

7.4. Practical Aspects

The Single Argumentative NXT BDI-like Agent and the MAS Argumentative NXT BDI-like Agents can be found in two different folders: DeLP-Server Single NXT Agent and DeLP-Server MAS NXT Agents, respectively.

Inside of each folder it is possible to find the DeLP-Server – "delp_server.exe" – which can be launched through a simple double click, Figure 25.

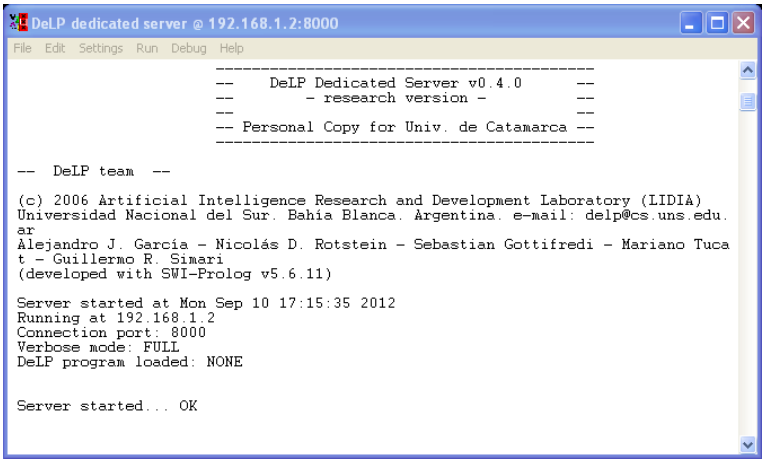


FIGURE 25 - DELP-SERVER

Again, inside of each folder, it is possible to find another folder named "AgentModel", which contains the Argumentative NXT BDI-like Agents (Single and MAS): "SingleAgent.pl" and "MASAgent.pl". It also contains the

other files that are required to modulate problems: “beliefs.delp”, “intentions.txt”, “actionPlans.pl”, “hub.pl” (and the “beliefsTime.txt” in the MAS case).

Taking the Single Agent as example, after we launch it, it is necessary to make the connection to the server using the command *connect*., Figure 26.

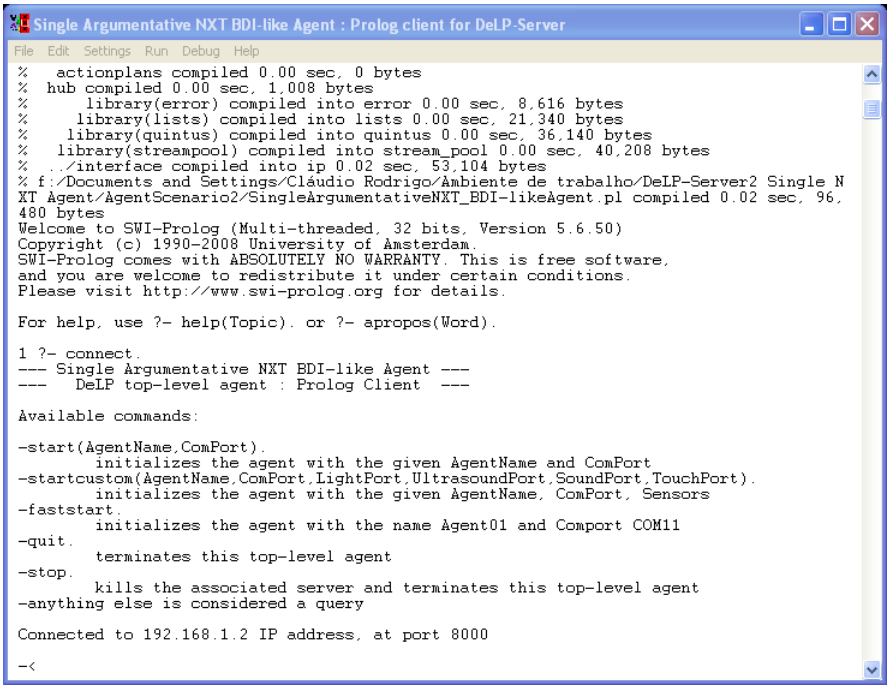


FIGURE 26 - SINGLE ARGUMENTATIVE NXT BDI-LIKE AGENT: PROLOG CLIENT

As Figure 26 shows, after connected to the server, some commands are available. We can start the agent using the command $start(AgentName, ComPort)$, where we can specify a name for the agent and the port that connects it. The command that allows the customization of the ports where the sensors are connected is $startcustom(AgentName, ComPort, LightPort, UltrasoundPort, SoundPort, TouchPort)$.. A command for quick start is available with the agent name “Agent01” and COM port 11: $faststart$. Using the command $quit$., we terminate the agent, and the command $stop$. will terminate the agent and the server. If something else is typed, it will be considered a query for the server (working as regular prolog client). This description is also applied to the MAS Agent.

With regard to files, we will now give some examples of how they can be filled. It is very important to fill them correctly in order to have a complete modulation of the problems and avoid bad behaviors from the agents.

The file “beliefs.delp” should be filled with a de.l.p., i.e. should be filled with strict rules, defeasible rules or facts. Example:

Strict rule: $go\_to\_gasPump <- \sim fuel$.
Defeasible rule: $use\_autobus -< \sim car$.
Fact: $raining <- true$.

The file “intentions.txt” should be filled with DeLP literals that will represent motivational states from the agents. It is recommended the existence of rules (in the file “beliefs.delp”) with the literals as heads of the rules. Example of an intention:

$use\_autobus$

The file “actionPlans.pl” should be filled with prolog functions that represent plans for the intentions. Example:

$use\_autobus(Robot) :-$
$write('Going\ to\ the\ bus\ stop'), nl,$
$walkToBusStop(Robot),$
$waitForBus(Robot),$
$enterIntoBus(Robot).$

The file “hub.pl” should be filled with the function $checkQuery(Query, Robot)$, which relates the intentions with the correspondent action plans. Example:

$checkQuery(Query, Robot) :-$
$Query = 'use\_autobus',$
$use\_autobus(Robot).$

The file “publicDynamicKnowledge” should be filled with facts that come from the agents’ perceptions. Note that the facts which are liable to share should be described into the action plans.

The files “belifsTime.pl” and “publicDynamicKnowledge” (for the MAS Agents), should be filled with times in the format HH,MM,SS.SSS (with hours, minutes and seconds). The first line of a time’s file corresponds to the time that the belief of the belief’s file was added. An example of the time from the initial beliefs could be:

00,00,00.000

The addition of more beliefs and respective times should be done through the file “actionPlans.pl”.

Note: When the agents are launched, some auxiliary files may be generated. It is not recommended to launch them from CD’s.

8. APPLICATION CASES

To test and evaluate the Single and MAS Argumentative NXT BDI-like Agents, two application cases were developed. The application cases are based on the transportation of passengers by taxis from the taxi rank to the airport. Three alternative roads should be considered.

Next a brief description of the environment and the interveners will be made through pictures:

- **Environment/World**: Figure 27 represents the environment used in the application cases. Rectangle 1 represents the taxi rank zone where the taxis can pick up passengers. Rectangle 2 represents the airport zone where the taxis drop the passengers. Rectangle 3 represents a garden which is a neutral zone, interdict to the agents. The green arrow represents road 1, namely Highway. Blue arrow represents road 2, namely Freeway. The red arrow represents road 3, namely Alternative Way.
- **Taxis**: In the application cases, the agents will be represented by taxis. The taxis were constructed using the Lego Mindstroms NXT (version 9797 - Educational). These robots were built using the driving base module instructions from the 9797 version. Then three sensors and a claw have been adapted: an ultrasound sensor to measure distances to objects; a light sensor to recognize the lines present in the environment; a touch sensor that works as an on/off button (irrelevant for the application cases). Two taxis were built, the taxi 1 (T1), presented in Figure 28, and the taxi 2 (T2), presented in Figure 29.
- **Passengers**: The passengers in the application cases will be represented by balls as Figure 30 shows.
- **Random agent**: Figure 31 represents a random agent, which is a casual robot that will be useful to symbolize agents that could be present in the environment.

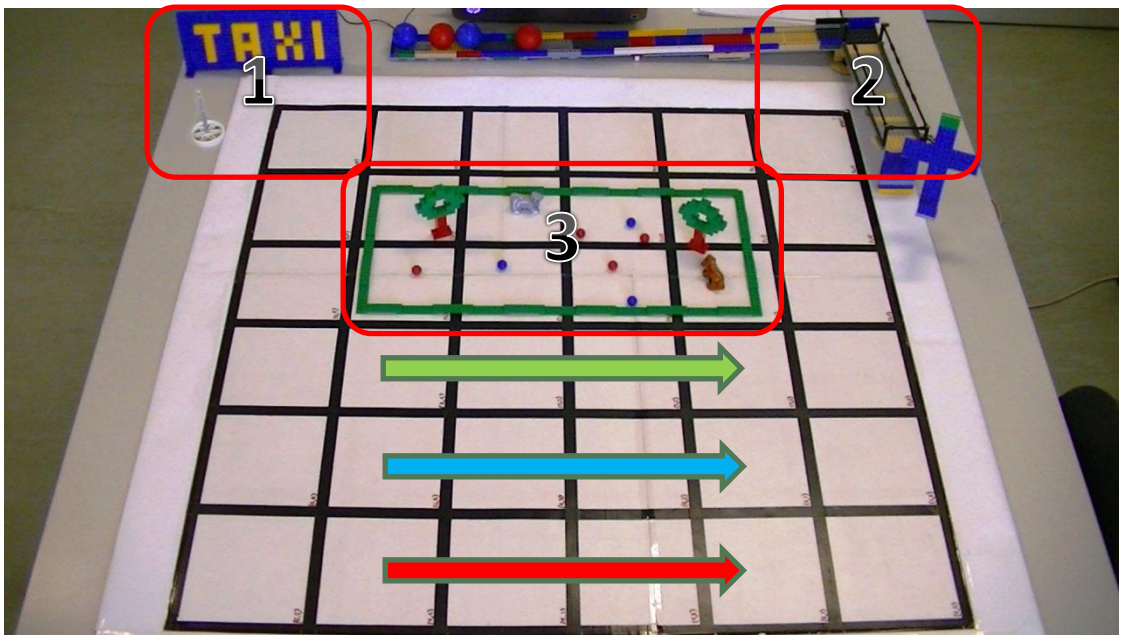


FIGURE 27 - APPLICATION CASES' ENVIRONMENT

FIGURE 28 - TAXI 1 (T1)

FIGURE 29 - TAXI 2 (T2)

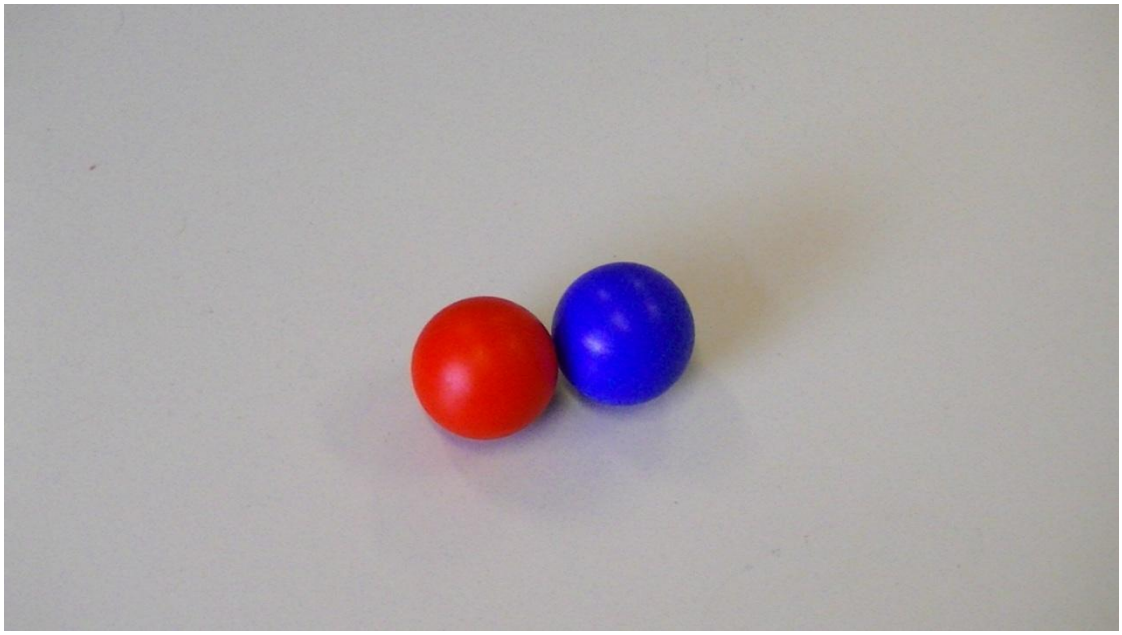


FIGURE 30 - PASSENGERS

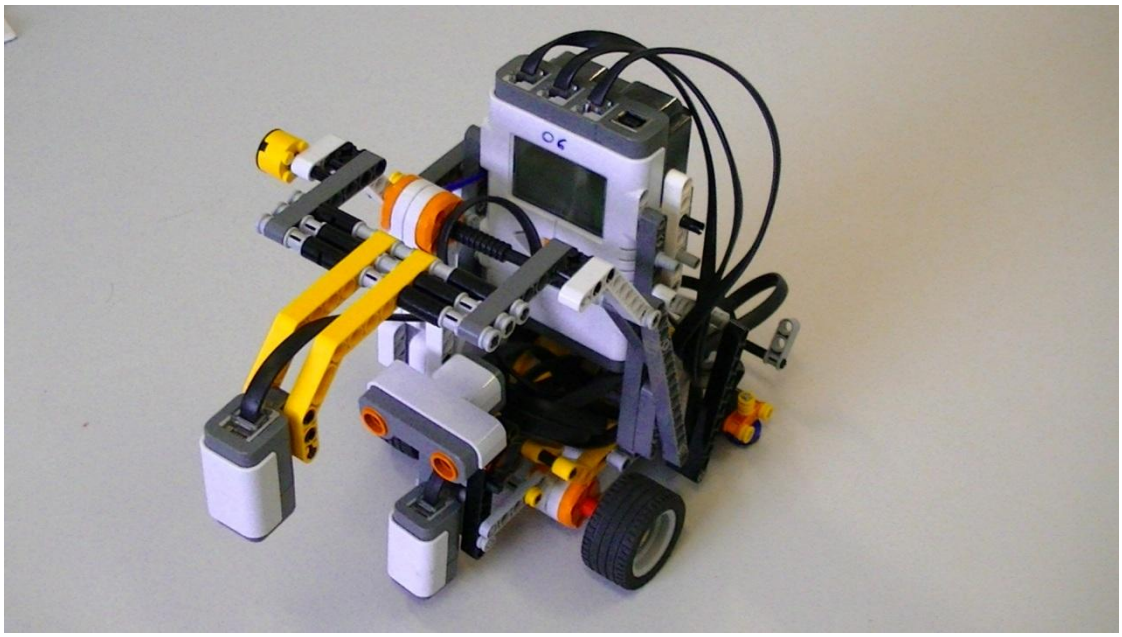


FIGURE 31 - RANDOM AGENT

Next, we will expose two application cases. The first one is for the Single Argumentative NXT BDI-like Agent and the second to MAS Argumentative NXT BDI-like Agents.

Note: In the application cases the public static knowledge will not be used to better understand the exposed situations. However, it can be used in other cases and the server will consider it to answer queries.

8.1. Application Case: Single Taxi

This application case is created to test and evaluate the architecture, reasoning capabilities and also the revision process from the Single Argumentative NXT BDI-like Agent.

8.1.1. Problem Description

The practical problem presents a Taxi (inspired in [58]). The main goal is to transport passengers from a point (Taxi Rank) to another (Airport) by the fastest way, considering three different roads: road 1 (Highway), road 2 (Freeway) and road 3 (Alternative way). Then, the agent should go back to collect more passengers.

The choice of the fastest way is made by using defeasible reasoning, i.e. the agent's decision must be supported by arguments. There are several conditions that can influence the agent's behavior. Let's explore:

- In ideal conditions the best way will be Highway>Freeway>Alternative Way (preferably in that order);
- If an agent knows that on a certain road there is traffic, then it must choose another way according to its beliefs;
- If it is a rush hour, then probably there is traffic on the Highway;

Figure 32 represents the problem and environment described above.

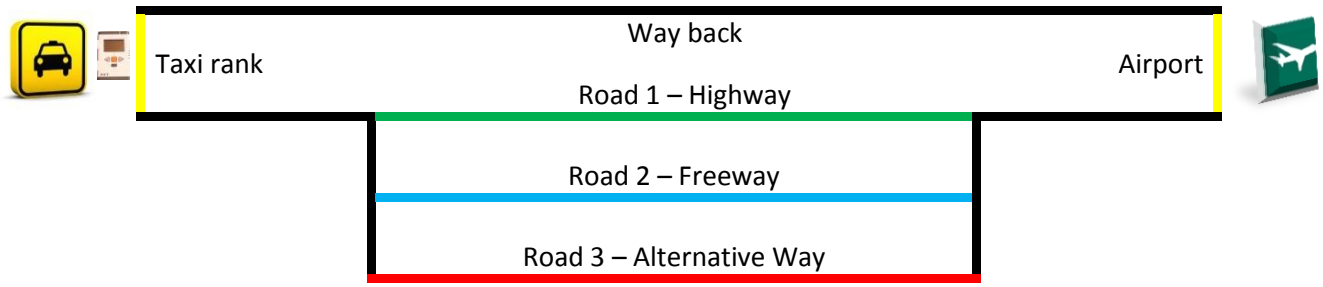


FIGURE 32 - APPLICATION CASE: SINGLE TAXI

8.1.2. Solution

The Single Argumentative NXT BDI-like Agent should be used to solve the exposed problem. As said in section 7, to model problems the use of the four main files associated with the Single Agent would be enough.

First we need to assume a global desire, which could be, for example, "Transport passengers from the taxi rank to the airport". This global desire should be summarized in a set of intentions, "intentions.txt". The beliefs of the agent should be depicted in the file "beliefs.delp". The action plans must be entered into the file "actionPlans.pl". The relation between them and the intentions should be done using the file "hub.pl".

In Appendix A it is possible to find the content of the files used to modulate the problem.

8.1.3. SCENARIOS

Using the solution present in Appendix A many situations can occur. In order to test some specific situations, we propose two different scenarios that follow. We will use storyboards to illustrate the overall behavior of the agents.

8.1.3.1. SCENARIO 1

In this first scenario we intend to demonstrate a simple situation using the files of Appendix A and the Taxi 1.

We will assume that it is not a rush hour and there is no traffic on the roads 2 and 3. So the following beliefs (facts) should be added to the belief base ("beliefs.delp") before the agent starts:

$\sim rush\_hour \ <- \ true.$
$\sim traffic2 \ <- \ true.$
$\sim traffic3 \ <- \ true.$

The storyboard (Storyboard A) for this scenario is represented in Figure 33, Figure 34 and Figure 35 by several pictures (from P1 to P10). A brief description of each picture is made, in order to understand the agent's behavior. The first picture represents the first action from the agent after being connected to the server, as sub-section 7.4 demonstrates.

Now we will take a close look at the "INVALID" answers returned by the server when contradictions occur. Considering P4 from Figure 33, for example, the contextual query $choose\_road1$ will be "INVALID", due to the contradictory beliefs 10/12 and 11/13 present in the belief base, at that point:

1. $choose\_road1 -< \ have\_passengers, \sim station, \sim airport, \sim traffic1.$
2. $choose\_road1 -< \ have\_passengers, \sim station, \sim airport, traffic1, traffic2, traffic3.$
3. $\sim choose\_road1 -< \ have\_passengers, \sim station, traffic1.$
4. $traffic1 -< \ rush\_hour.$
5. $\sim traffic1 -< \ \sim rush\_hour.$
6. $\sim rush\_hour \ <- \ true.$
7. $\sim traffic2 \ <- \ true.$
8. $\sim traffic3 \ <- \ true.$
9. $\sim airport \ <- \ true.$
10. $\sim have\_passengers \ <- \ true.$
11. $rank \ <- \ true.$
12. $have\_passengers \ <- \ true.$
13. $\sim rank \ <- \ true.$

…

So the agent will revise its beliefs and the oldest beliefs in conflict will be discarded, i.e. the beliefs 10 and 11 will be removed from the belief base. Then the contextual query $choose\_road1$ will be made again and due to the belief 1 (which is a defeasible rule) the returned answer will be "YES". This situation is similar to the other situations where "INVALID" answers are returned by the server and illustrates how the agents behave.

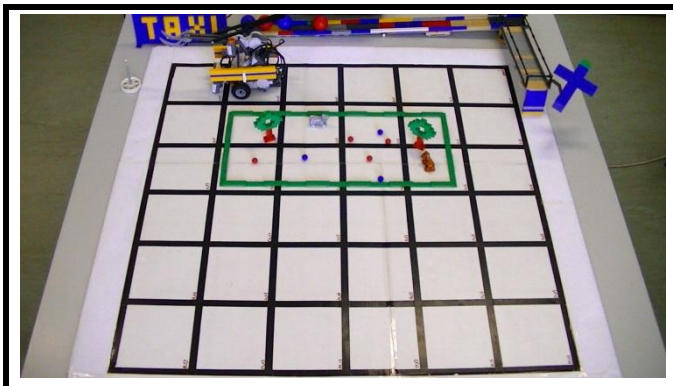
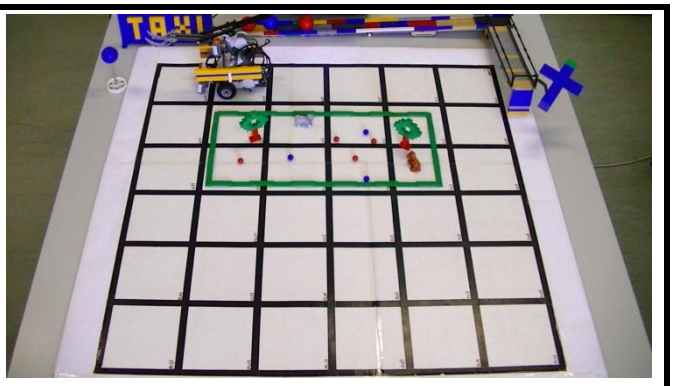
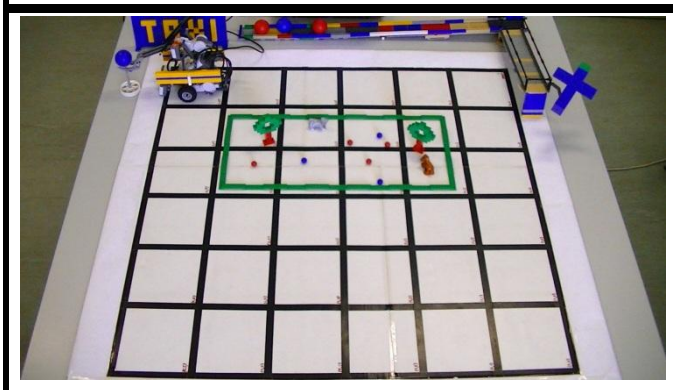
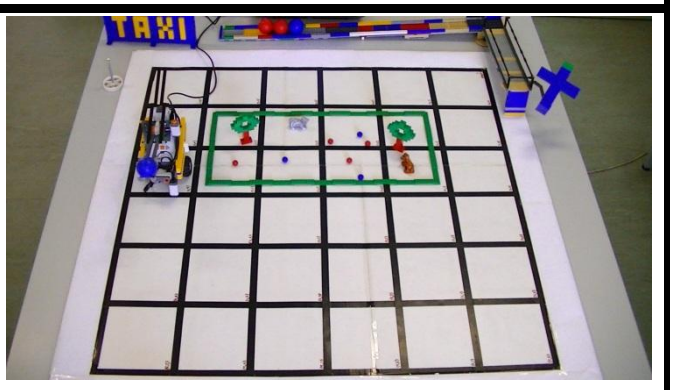
	
P1: In this first decision point, the intention $wait\_passengers$ is chosen. A contextual query with that intention is made and the answer returned by the server is "YES", considering the agent's beliefs.	**P2**: The action plan $wait\_passengers(Robot)$ started and the agent waited until passengers appeared. The agent should now pick up the passengers.
	
P3: A new belief ($have\_passengers <- true.$) is added to the belief base. Next, the agent should continue the action plan and move to the next decision point.	**P4**: A new belief ($\sim rank <- true.$) is added, since the agent is no longer in the taxi rank. The current action plan ends and the next intention is chosen ($choose\_road1$). This is the second decision point, where a contextual query is made, but now the answer from the server is "INVALID" due to the contradictions caused by the previously added beliefs. So the revision process will occur and discard the following beliefs: $\sim have\_passengers <- true.$ and $rank <- true.$, one at a time, since they are older than their complement. Then, the same contextual query ($choose\_road1$) is made again and now the answer returned by the server is "YES".

FIGURE 33 - STORYBOARD A

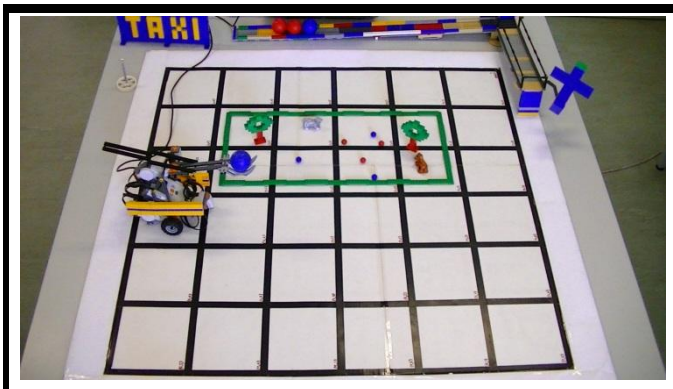
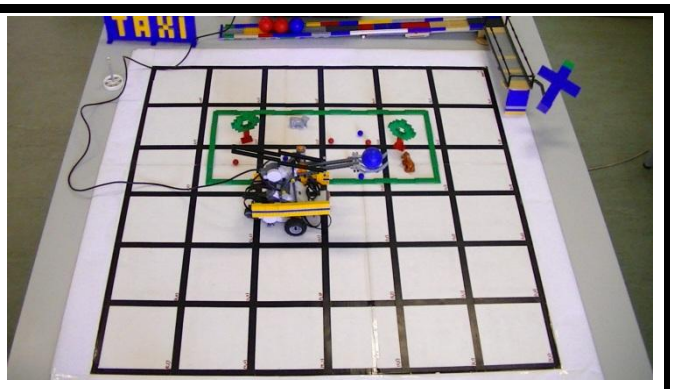
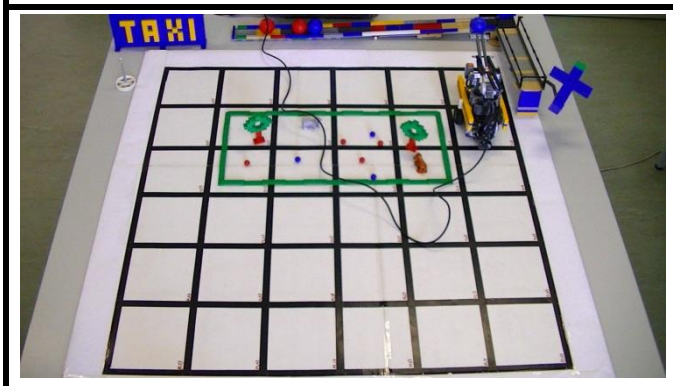
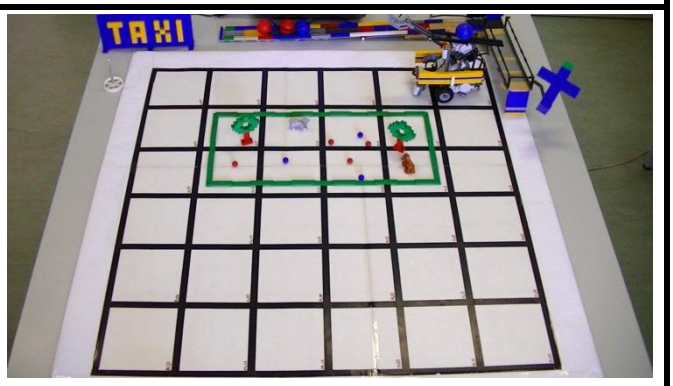
	
P5: The action plan $choose\_road1(Robot)$ starts and the agent will move on Road 1.	**P6**: During its trip through road 1, it will be checking for traffic. If it finds, then will add the belief $traffic1 <- true.$, if not, then adds the belief $\sim traffic1 <- true.$, as in this case.
	
P7: The agent arrives to a new decision point (airport) after finishing the action plan. A new belief is inserted ($airport <- true.$). The next intention is considered ($choose\_road2$) and a contextual query is submitted to the server. According to the agent's belief base the answer is "INVALID", due to the contradictory beliefs $\sim airport <- true.$ and $airport <- true.$. The agent will revise the beliefs and eliminate $\sim airport <- true.$ The same contextual query will be made to the server and the answer will be "NO". The intention is discarded and the next intention is considered ($choose\_road3$). Again, a contextual query is made and the answer from the server is "NO". The process is repeated and now the intention $drop\_passengers$ is warranted by the server with the answer "YES". The action plan $drop\_passengers(Robot)$ starts.	**P8:** During the current action plan, $drop\_passengers(Robot)$, the agent will, obviously, drop the passengers at the airport, add the new belief $\sim have\_passengers <- true.$ and then move to the next decision point.

FIGURE 34 - STORYBOARD A (CONTINUATION)

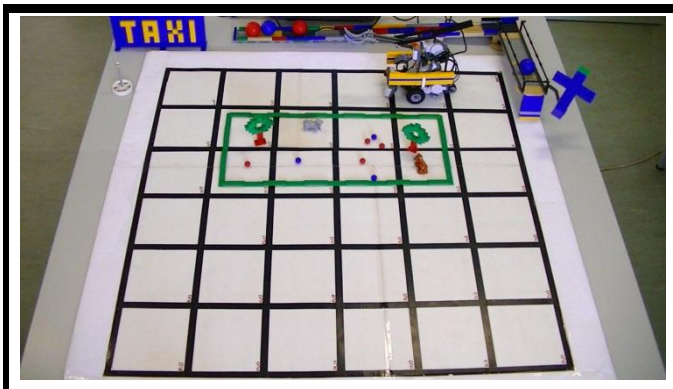
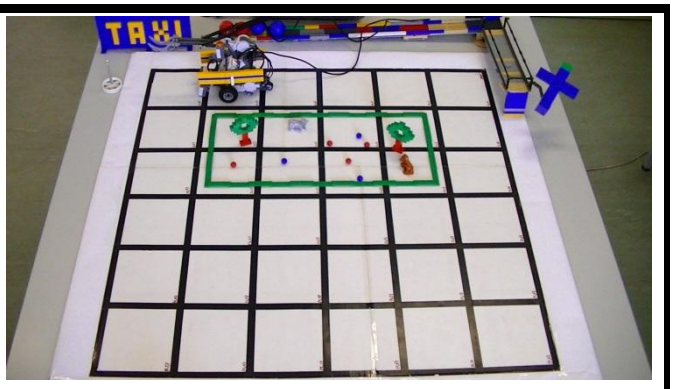
	
P9: At this decision point the next intention ($back\_to\_rank$) will return the answer "INVALID" from the server due to the contradictory beliefs $have\_passengers <- true.$(which is the oldest) and $\sim have\_passengers <- true.$. After revising its beliefs, the agent will, again, submit the contextual query to the server and receive an affirmative answer. Then the action plan $back\_to\_rank(Robot)$ will start.	**P10**: When the agent arrives to this decision point, it will add two new beliefs: $\sim airport <- true.$ and $rank <- true$. Now the agent will check for another intention, but it will realize that the list is empty. According to the flowchart from the Figure 21, when that happens, the initial intentions' list is restored and the whole process begins. This means that the next considered intention will be $wait\_passengers$, but the answer will be "INVALID". The agent will revise its beliefs and eliminate the following beliefs: $airport <- true.$ and $\sim rank <- true.$ Then the answer to the same query will be "YES" and the action plan $wait\_passengers(Robot)$ will start. Note that when the agent is waiting for passengers, if the button from the touch sensor is pressed, the agent will turn off.

FIGURE 35 - STORYBOARD A (CONTINUATION)

8.1.3.2. SCENARIO 2

In this second scenario we intend to demonstrate a different situation using the files of Appendix A. We will use the Taxi 1 and the Random agent to symbolize traffic.

We will assume that it is a rush hour and there is no traffic on the roads 2 and 3. So the following beliefs (facts) should be added to the belief base ("beliefs.delp") before the agent starts:

$rush\_hour <- true.$
$\sim traffic2 <- true.$
$\sim traffic3 <- true.$

The storyboard (Storyboard B) for this scenario is represented in Figure 36,Figure 37 and Figure 38 by several pictures (from P1 to P10). A brief description of each picture is made, in order to understand the agent's behavior. The first picture represents the first action from the agent after being connected to the server, as sub-section 7.4 demonstrates.

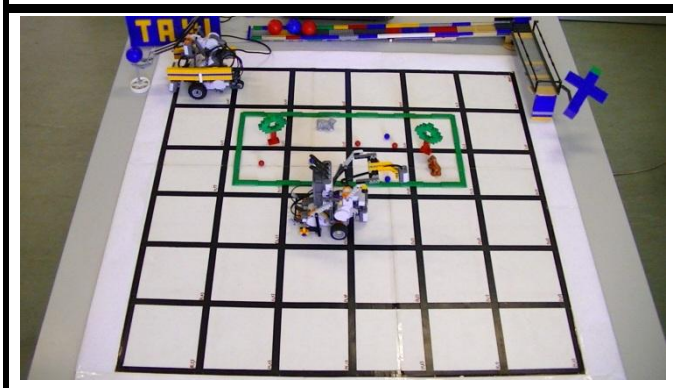
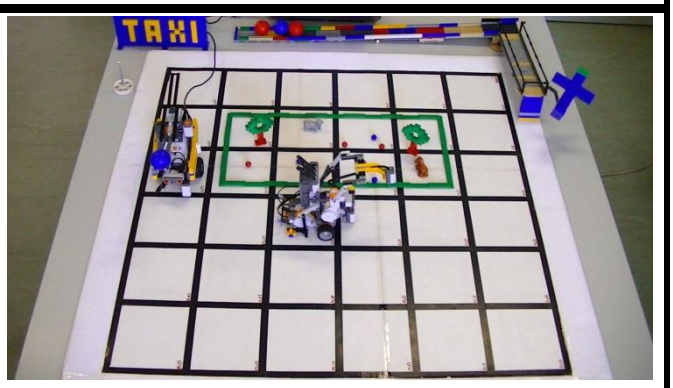
	
P1: In this first decision point, the intention $wait\_passengers$ is chosen. A contextual query with that intention is made and the answer returned by the server is "YES".	**P2**: The action plan $wait\_passengers(Robot)$ started and the agent waited until passengers appeared. The agent should now pick up the passengers.
	
P3: A new belief ($have\_passengers <- true.$) is added to the belief base. Next, the agent should continue the action plan and move to the next decision point.	**P4**: A new belief ($\sim rank <- true.$) is added, since the agent is no longer in the taxi rank. The current action plan ends and the next intention is chosen ($choose\_road1$). This is the second decision point, where a contextual query is made, but now the answer from the server is "INVALID" due to the contradictions caused by the previously added beliefs. So the revision process will occur and discard the following beliefs: $\sim have\_passengers <- true.$ and $rank <- true.$, one at a time, since they are older than their complement. Then the same contextual query ($choose\_road1$) is made again and now the answer returned by the server is "NO", since the server can warrant $\sim choose\_road1$ (through a defeasible rule). So, the next intention, $choose\_road2$, is chosen and warranted by the server.

FIGURE 36 - STORYBOARD B

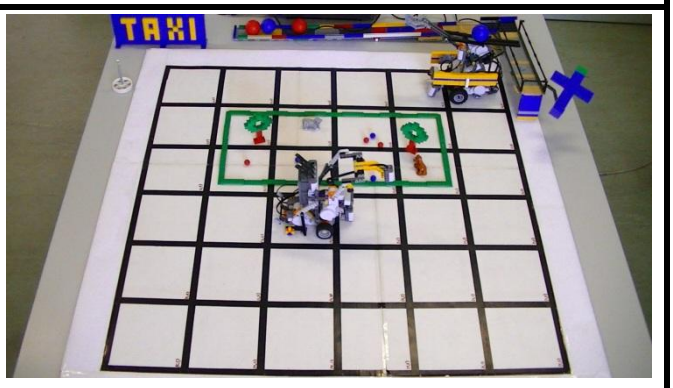
	
P5: The action plan $choose\_road2(Robot)$ starts and the agent will move on Road 2.	**P6**: During its trip through road 2, the agent will be checking for traffic. If it finds traffic, then it will add the belief $traffic2 <- true.$, if not, then it adds the belief $\sim traffic2 <- true.$, as in this case.
	
P7: The agent arrives to a new decision point (airport) after finishing the action plan. A new belief is inserted ($airport <- true.$). The next intention is considered ($choose\_road3$) and a contextual query is submitted to the server. According to the agent's belief base the answer is "INVALID", due to the contradictory beliefs $\sim airport <- true.$ and $airport <- true.$. The agent will revise the beliefs and eliminate $\sim airport <- true.$ The same contextual query will be made to the server and the answer will be "NO". The intention is discarded and the next intention is considered ($drop\_passengers$) and is warranted by the server since the answer is "YES". The action plan $drop\_passengers(Robot)$ starts.	**P8:** During the current action plan, $drop\_passengers(Robot)$, the agent will, obviously, drop the passengers at the airport, add the new belief $\sim have\_passengers <- true.$ and then move to the next decision point.

FIGURE 37 - STORYBOARD B (CONTINUATION)

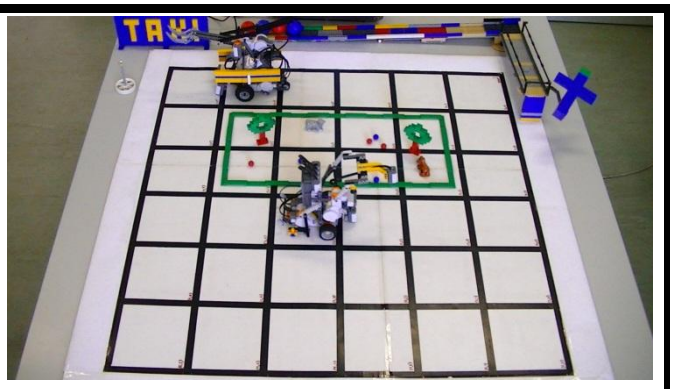
	
P9: At this decision point the next intention ($back\_to\_rank$) will return the answer "INVALID" from the server due to the contradictory beliefs $have\_passengers <- true.$(which is the oldest) and $\sim have\_passengers <- true.$. After revising its beliefs, the agent will, again, submit the contextual query to the server and receive an affirmative answer. Then the action plan $back\_to\_rank(Robot)$ will start.	**P10**: When the agent arrives to this decision point, it will add two new beliefs: $\sim airport <- true.$ and $rank <- true$. Now the agent will check for another intention, but it will realize that the list is empty. According to the flowchart from the Figure 21, when that happens, the initial intentions' list is restored and the whole process begins. This means that the next considered intention will be $wait\_passengers$, but the answer will be "INVALID". The agent will revise its beliefs and eliminate the following beliefs: $airport <- true.$ and $\sim rank <- true$. Then the answer to the same query will be "YES" and the action plan $wait\_passengers(Robot)$ will start. Note that when the agent is waiting for passengers, if the button from the touch sensor is pressed the agent will turn off.

FIGURE 38 - STORYBOARD B (CONTINUATION)

8.2. Application Case: Taxi Company

This application case is devised to test and evaluate the architecture, reasoning capabilities, revision process and the multi-agent environment using the MAS Argumentative NXT BDI-like Agents.

8.2.1. Problem Description

This practical problem presents a Taxi Company. As in the previous application case, the main goal is to transport passengers from a point (Taxi Rank) to another (Airport) by the fastest way, considering three different roads: road 1 (Highway), road 2 (Freeway) and road 3 (Alternative way). Then the taxis should go back to collect more passengers. Since the taxis are from the same company they should share information with each other to improve their performance

The choice of the fastest way is made by using defeasible reasoning, i.e. the agents' decision must be supported by arguments. Some conditions can influence the agents' behavior:

- In ideal conditions the best way will be Highway>Freeway>Alternative Way (preferably in that order);
- If an agent knows that in a certain road there is traffic, then it must choose other way according with its beliefs and the public dynamic knowledge;
- If it is a rush hour, then probably there is traffic in the Highway;

Figure 39 represents the problem and environment depicted above.

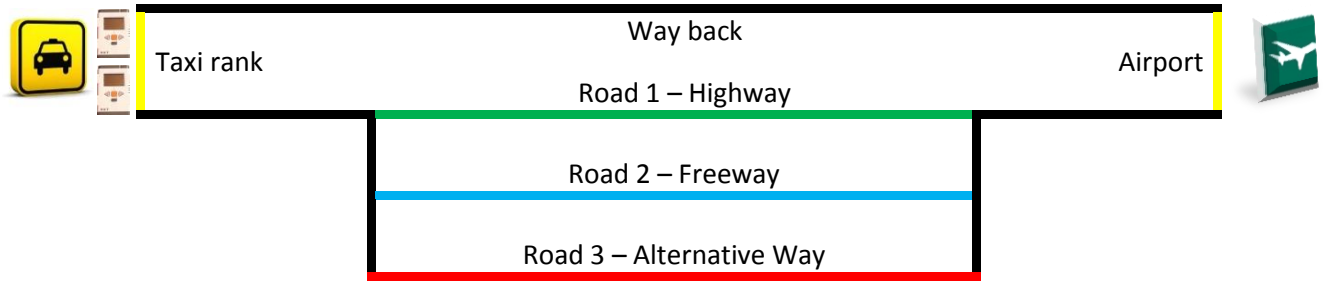


FIGURE 39 - APPLICATION CASE: TAXI COMPANY

8.2.2. SOLUTION

To solve the exposed problem we are going to use the MAS Argumentative NXT BDI-like Agents, modeling them through the main files associated with ("beliefs.delp", "beliefsTime.txt", "intentions.txt", "actionPlans.pl" and "hub.pl").

We can think in a global desire for the whole system, which could be, for example "Carry as many passengers as possible from the taxi rank to the airport." This desire can be decomposed in the agents' desires, which could be, for example, "Transport passengers from the taxi rank to the airport and share important information". The intentions should summarize this global desire through motivational states. The beliefs of each agent should be entered in the file "beliefs.delp", accompanied by the times in the file "beliefsTime.txt" (the time for each belief will be 00,00,00.000). The action plans must be described into the file "actionPlans.pl". The relation between them and the intentions should be done using the file "hub.pl".

In this application case the agents will be initially modulated with the same content in the files, i.e. in the beginning they will be copies of each other in order to facilitate the understanding of the situations. In Appendix B it is possible to find their content.

8.2.3. SCENARIOS

Two different scenarios will be exposed in order to test various situations using the solution presented in the Appendix B. To have a better understanding of the agents' behavior we have resorted to storyboards.

8.2.3.1. SCENARIO 1

In this first scenario we will make use of the Taxi 1 (T1), Taxi 2 (T2) and the Random agent.

We will assume that it is not a rush hour and there is no traffic on the roads 2 and 3 (at the beginning). So the following beliefs (facts) should be added to the belief base ("beliefs.delp"), as well as the respective times, before the agents' starts:

$\sim rush\_hour \prec - true.$
$\sim traffic2 \prec - true.$
$\sim traffic3 \prec - true.$

Besides these, we will add different beliefs to the agents so that one can start warranting the intention $wait\_passengers$ and the other $back\_to\_rank$. Thus we guarantee that they are in a row and do not hatch.

T1:
$\sim have\_passengers \prec - true.$
$\sim airport \prec - true.$
$rank \prec - true.$

T2:
$\sim have\_passengers \prec - true.$
$airport \prec - true.$
$\sim rank \prec - true.$

The storyboard (Storyboard C) for this scenario is represented in Figure 40, Figure 41, Figure 42, Figure 43 and Figure 44 by several pictures (from P1 to P20). A brief description of each picture is made in order to understand the agents' behavior. The first picture represents the first action from the agents after being connected to the server, as sub-section 7.4 demonstrates.

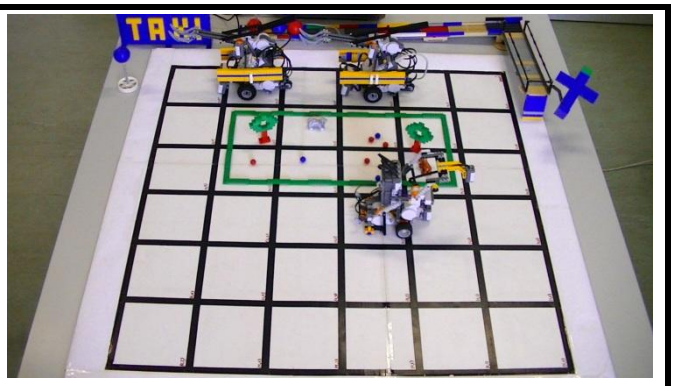
<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>P1: In this first decision point, the intention $wait\_passengers$ is chosen by T1. A contextual query with that intention is made and the answer returned by the server is “YES”, considering its beliefs and public dynamic knowledge (which is empty, for now). According to the beliefs of T2 the unique warranted intention will be $back\_to\_rank$.</td><td>P2: T1’s action plan, $wait\_passengers(Robot)$, started and the agent waited until passengers appeared. Then, T1 should pick up the passengers. T2’s action plan, $back\_to\_rank(Robot)$, is running but it cannot be completed until T1 leaves the decision point.</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>P3: T1 adds a new belief to its private knowledge ($have\_passengers <- true.$). Next, T1 should continue the action plan and move to the next decision point.</td><td>P4: A new belief ($\sim rank <- true.$) is added by T1 to its knowledge and its current action plan ends. T1 chooses the next intention ($choose\_road1$). This is the second decision point for T1. The answer from the server is “INVALID” due to the contradictions caused by the previously added beliefs. So the revision process will occur and T1 discards the beliefs $\sim have\_passengers <- true.$ and $rank <- true.$ T1 makes the same contextual query ($choose\_road1$) and the answer returned by the server is “YES”. In turn T2 ends its current action plan by arriving to the decision point and will add the following beliefs to its own private knowledge: $rank <- true.$ and $\sim airport <- true.$ The next intention ($wait\_passengers$) chosen by T2 will receive the answer “INVALID”. Although, after the revision process that intention will be warranted.</td></tr>
</table>

FIGURE 40 - STORYBOARD C

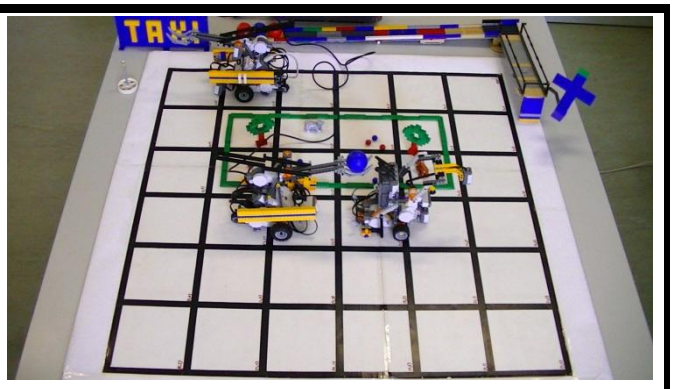
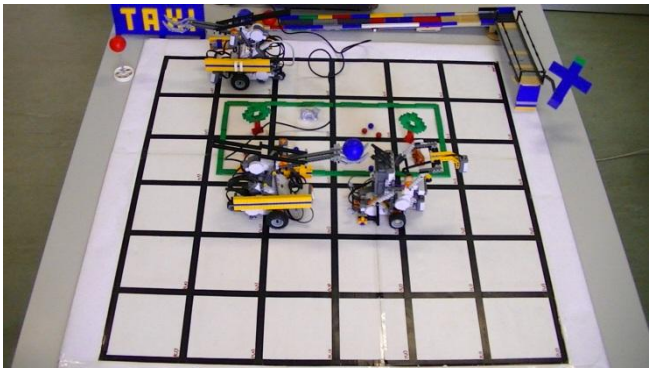
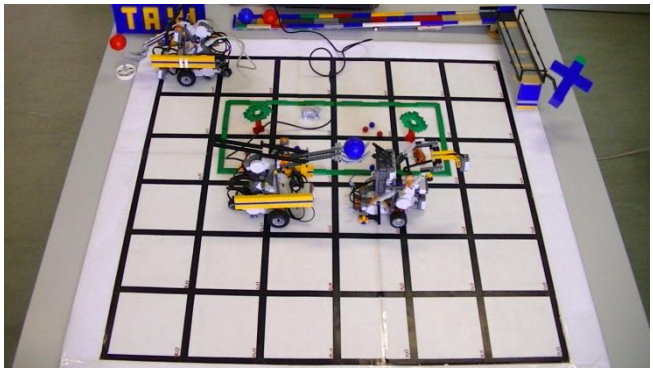
	
P5: The action plan $choose\_road1(Robot)$ from T1 starts and the agent will move on road 1. T2 started the action plan $wait\_passengers(Robot)$.	**P6**: During its trip through road 1, T1 will be checking for traffic. If it does not find any traffic then T1 will add the belief $\sim traffic1 <- true.$, otherwise adds the belief $traffic1 <- true.$, as in this case. Note, this kind of knowledge is important to be shared, so T1 will add it to its private knowledge and also to the public dynamic knowledge.
	
P7: T1 stays stopped in the traffic (represented by the random agent). T2 will continue its action plan and will pick up the passengers.	**P8:** T2 picks up the passengers and moves on to the next decision point, following its current action plan.

FIGURE 41 - STORYBOARD C (CONTINUATION)

	
P9: At this decision point the next intention ($choose\_road1$) from T2 will return the answer "INVALID". But after revising its beliefs, the answer will be "NO" due to the public dynamic knowledge (information shared by T1).	**P10**: The next intention ($choose\_road2$) will be considered by T2 and will be warranted by the server. Then the action plan $choose\_road2(Robot)$ will start.
	
P11: During its trip through road 1, T2 will be checking for traffic. If it finds traffic then T1 will add the belief $traffic1 <- true.$, otherwise it adds the belief $\sim traffic1 <- true.$, as in this case. Again, note that this kind of knowledge is important to be shared, so T2 will add it to its private knowledge and also to the public dynamic knowledge.	**P12**: T2 arrives to a new decision point (airport) after it finishes its current action plan. A new belief is inserted ($airport <- true.$) in its private knowledge. The next intention is considered ($choose\_road3$) and a contextual query is submitted to the server. The answer is "INVALID", due to contradictory beliefs. After T2 revises its beliefs the answer will be "NO". The intention is discarded and the next intention is considered ($drop\_passengers$). The process is repeated and now the intention $drop\_passengers$ is warranted by the server. T2's action plan $drop\_passengers(Robot)$ starts.

FIGURE 42 - STORYBOARD C (CONTINUATION)

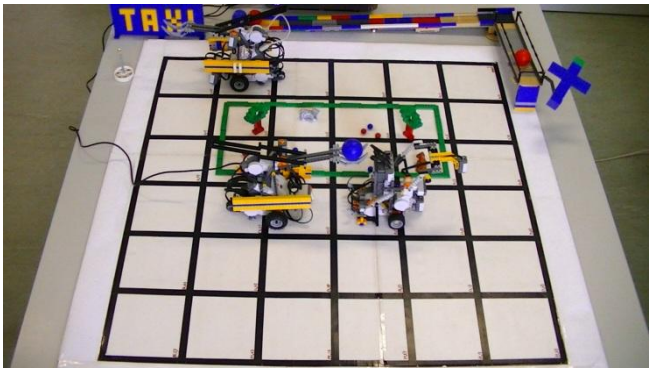
	
P13: During the current T2's action plan, $drop\_passengers(Robot)$, the agent will, obviously, drop the passengers at the airport, add the new belief $\sim have\_passengers <- true$. and then move to the next decision point.	**P14**: At this decision point, T2 will choose the next intention ($back\_to\_rank$) and the answer from the server will be "INVALID" due to the contradictory beliefs. After revision of its beliefs, T2 will, again, submit the contextual query to the server and receive an affirmative answer. Then the action plan $back\_to\_rank(Robot)$ will start.
	
P15: When T2 arrives to this decision point, it will add two new beliefs to its private knowledge: $\sim airport <- true$. and $rank <- true$. Now T2 will check for another intention, but it will realize that the list is empty. So, it will restore the initial intentions' list and the whole process begins. This means that the next intention considered will be $wait\_passengers$, but the answer will be "INVALID". The agent will revise its beliefs and thereafter the answer to the same query will be "YES". The action plan $wait\_passengers(Robot)$ will start. Note that when T2 is waiting for passengers, if the button from the touch sensor is pressed it will turn off.	**P16**: When the traffic from road 1 begins to flow, T1 will continue its path until the next decision point.

FIGURE 43 - STORYBOARD C (CONTINUATION)

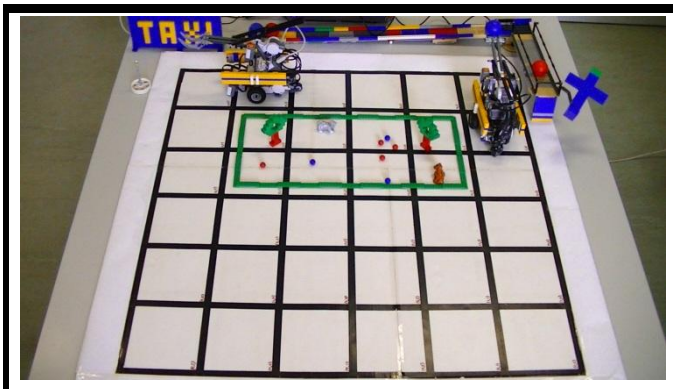
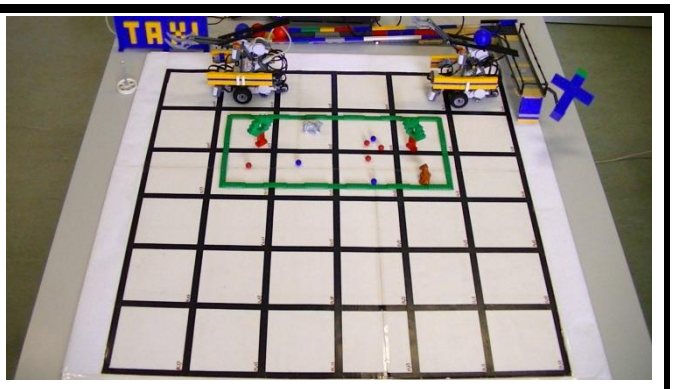
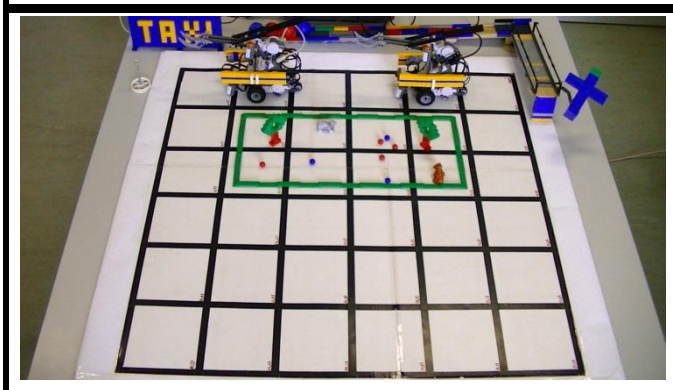
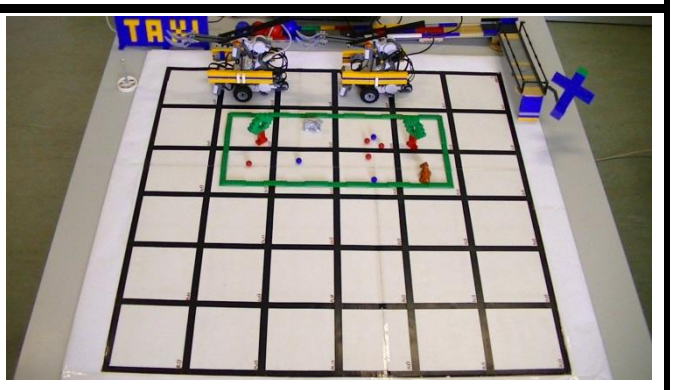
	
P17: T1 arrives to a new decision point (airport) after finishing its current action plan. A new belief is inserted ($airport <- true.$) in its private knowledge. The next intention is considered ($choose\_road2$) and a contextual query is submitted to the server. The answer is "INVALID", due to the contradictory beliefs. So, after T1 revises its beliefs the answer will be "NO". The intention is discarded and the next intention is considered ($choose\_road3$). The answer is also "NO". The process is repeated for the next intention ($drop\_passengers$) and now it is warranted by the server. T1's action plan $drop\_passengers(Robot)$ starts.	**P18**: During the current T1's action plan, $drop\_passengers(Robot)$, the agent will, obviously, drop the passengers at the airport, add the new belief $\sim have\_passengers <- true.$ to its private knowledge and then move to the next decision point.
	
P19: At this decision point, T1 will choose the next intention ($back\_to\_rank$) and the answer from the server will be "INVALID" due to the contradictory beliefs. Revising its beliefs, the T1 will, again, submit the contextual query to the server and receive an affirmative answer. Then the action plan $back\_to\_rank(Robot)$ will start.	**P20**: T2 will not arrive to the next decision point, until T1 vacates it. Meanwhile, T2 will wait during its actual action plan (it is the same situation presented in P1 from this scenario).

FIGURE 44 - STORYBOARD C (CONTINUATION)

8.2.3.2. SCENARIO 2

In this second scenario we will make use of the Taxi 1 (T1), Taxi 2 (T2) and the Random agent.

We will assume that it is a rush hour and there is no traffic on the roads 2 and 3 (at the beginning). So the following beliefs (facts) should be added to the belief base ("beliefs.delp"), as well as the respective times, before the agents' starts:

$rush\_hour \prec - true.$
$\sim traffic2 \prec - true.$
$\sim traffic3 \prec - true.$

Besides these, we will add different beliefs to the agents so that one can start warranting the intention $wait\_passengers$ and the other $back\_to\_rank$. Thus we guarantee that they are in a row and do not crash.

T1:
$\sim have\_passengers \prec - true.$
$\sim airport \prec - true.$
$rank \prec - true.$

T2:
$\sim have\_passengers \prec - true.$
$airport \prec - true.$
$\sim rank \prec - true.$

The storyboard (Storyboard D) for this scenario is represented in Figure 45, Figure 46, Figure 47, Figure 48 and Figure 49 by several pictures (from P1 to P20). A brief description of each picture is made, in order to understand the agents' behavior. The first picture represents the first action from the agents after being connected to the server, as sub-section 7.4 demonstrates.

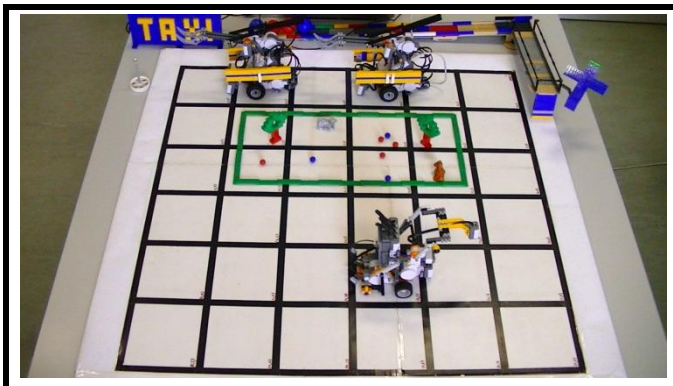
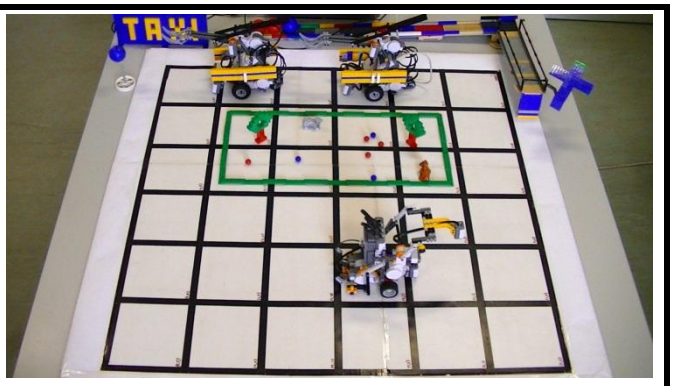
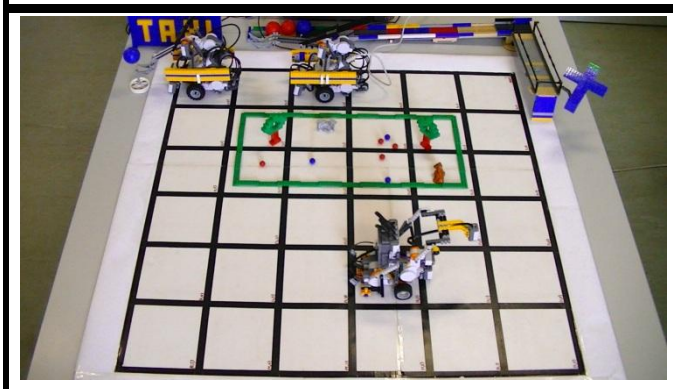
	
P1: In this first decision point, the intention $wait\_passengers$ is chosen by T1. A contextual query with that intention is made and the answer returned by the server is "YES", considering its beliefs and public dynamic knowledge (which is empty, for now). According to the beliefs of T2 the unique warranted intention will be $back\_to\_rank$.	**P2**: T1's action plan, $wait\_passengers(Robot)$, started and the agent waited until passengers appeared. Then, T1 should pick up the passengers. T2's action plan, $back\_to\_rank(Robot)$, is running but it cannot be completed until T1 leaves the decision point.
	
P3: T1 adds a new belief to its private knowledge ($have\_passengers <- true.$). Next, T1 should continue the action plan and move to the next decision point.	**P4**: A new belief ($\sim rank <- true.$) is added by T1 to its own knowledge and its current action plan ends. T1 chooses the next intention ($choose\_road1$). This is the second decision point for T1. The answer from the server is "INVALID" due to contradictory beliefs. So the revision process will occur and T1 discards the beliefs $\sim have\_passengers <- true.$ and $rank <- true.$ T1 makes the same contextual query ($choose\_road1$) and the answer returned by the server is "NO", due to the beliefs (in particular, $ush\_hour <- true.$; $traffic1 -< rush\_hour.$). Then the next intention chosen, $choose\_road2$, will be warranted. In turn T2 arrives to the next decision point and will add the following beliefs to its private knowledge: $rank <- true.$ and $\sim airport <- true.$

FIGURE 45 - STORYBOARD D

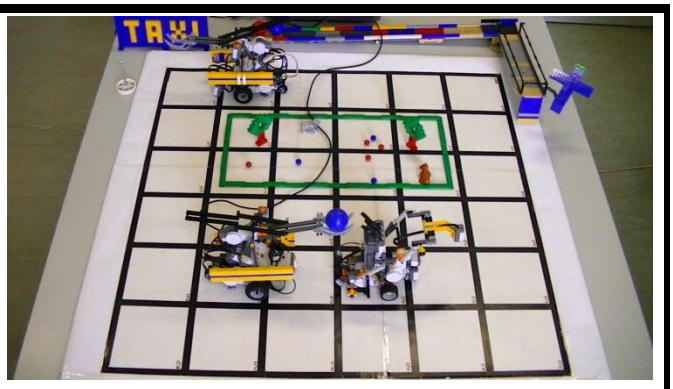
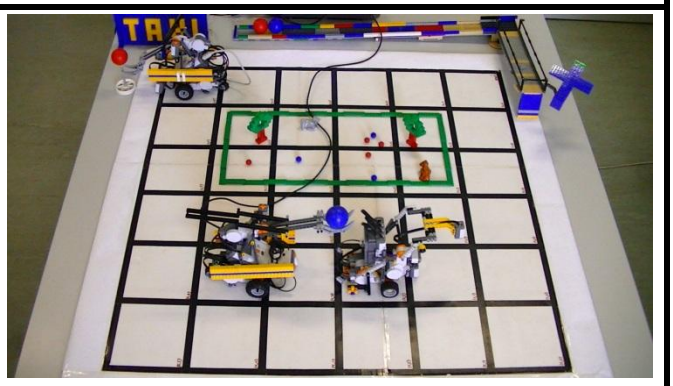
	
P5: The next intention ($wait\_passengers$) chosen by T2 will receive the answer "INVALID". Although, after the revision process, that intention will be warranted and the agent will wait until passengers appear. In turn, the action plan $choose\_road2(Robot)$ from T1 started and T1 will move on road 2. Note: As we can see, although T1 chose road 2 because it is a rush hour, there is no traffic in road 1. However its reasoning is correct.	**P6**: During its trip through road 2, T1 will be checking for traffic. If it does not find any, T1 will add the belief $\sim traffic2 <- true.$, otherwise it adds the belief $traffic2 <- true.$, as in this case. Note that this kind of knowledge is important to be shared, so T1 will add it to its private knowledge and also to the public dynamic knowledge.
	
P7: T1 stays stopped in the traffic (represented by the random agent). T2 will continue its action plan and will pick up the passengers.	**P8:** T2 picks up the passengers and moves on to the next decision point, following its current action plan.

FIGURE 46 - STORYBOARD D (CONTINUATION)

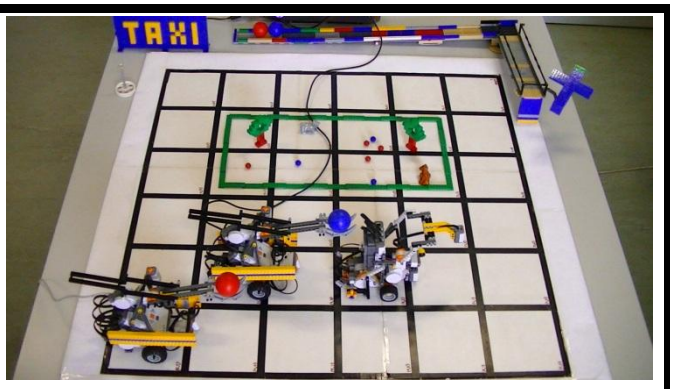
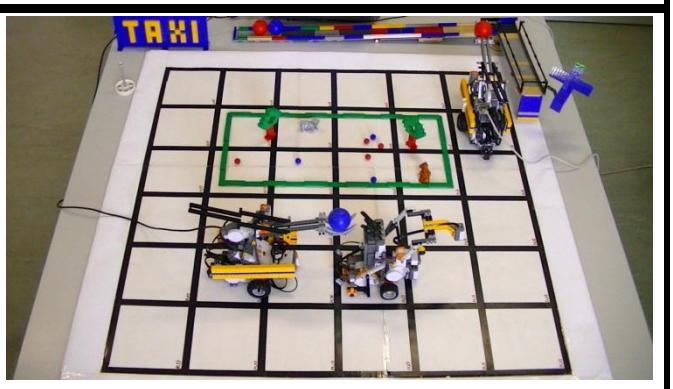
	
P9: At this decision point the next intention ($choose\_road1$) from T2 will return the answer "INVALID". But after revise its beliefs, the answer will be "NO" due to the beliefs: $rush\_hour <- true$. and $traffic1 -< rush\_hour$.). T2 will choose the next intention, $choose\_road2$, and the answer from the server will also be "NO" due to the public dynamic knowledge (information shared by T1).	**P10**: The next intention ($choose\_road3$) will be considered by T2 and will be warranted by the server. Then, the action plan $choose\_road3(Robot)$ will start.
	
P11: During its trip through road 3, T2 will be checking for traffic. If it finds traffic, then T1 will add the belief $traffic3 <- true.$, otherwise it adds the belief $\sim traffic3 <- true.$, as in this case. Again, note that this kind of knowledge is important to be shared, so T2 will add it to its private knowledge and also to the public dynamic knowledge.	**P12**: T2 arrives to a new decision point (airport) after finishing its current action plan. A new belief is inserted ($airport <- true.$) in its private knowledge. The next intention is considered ($drop\_passengers$) and a contextual query is submitted to the server. The answer is "INVALID", due to the contradictory beliefs. After T2 revises its beliefs the answer will be "YES" and the action plan $drop\_passengers(Robot)$ starts.

FIGURE 47 - STORYBOARD D (CONTINUATION)

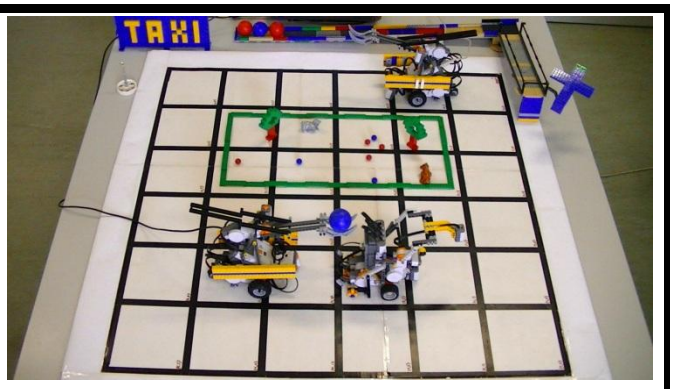
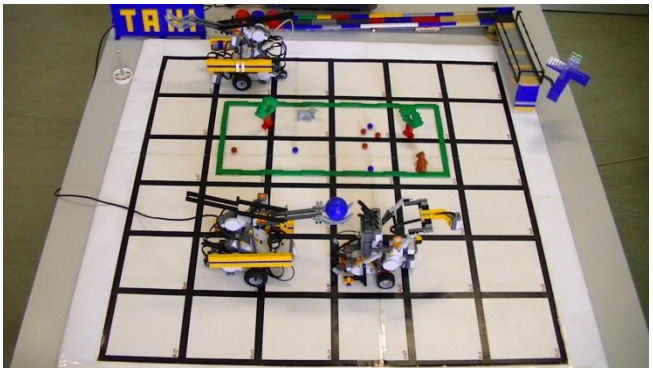
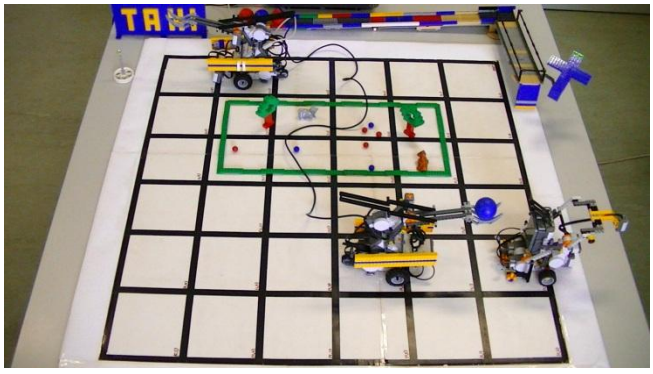
	
P13: During the current T2's action plan, $drop\_passengers(Robot)$, the agent will, obviously, drop the passengers at the airport, add the new belief $\sim have\_passengers <- true$. and then move to the next decision point.	**P14**: At this decision point, T2 will choose the next intention ($back\_to\_rank$) and the answer from the server will be "INVALID" due to the contradictory beliefs. After revising its beliefs, T2 will, again, submit the contextual query to the server and receive an affirmative answer. Then the action plan $back\_to\_rank(Robot)$ will start.
	
P15: When T2 arrives to this decision point, it will add two new beliefs to its private knowledge: $\sim airport <- true$. and $rank <- true$. Now T2 will check for another intention, but it will realize that the list is empty. So, it will restore the initial intentions' list and the whole process begins. This means that the next intention considered will be $wait\_passengers$, but the answer will be "INVALID". The agent will revise its beliefs and after that the answer to the same query will be "YES". The action plan $wait\_passengers(Robot)$ will start. Note that when the T2 is waiting for passengers, if the button from the touch sensor is pressed it will turn off.	**P16**: When the traffic from road 2 begins to flow, T1 will continue its path until the next decision point.

FIGURE 48 - STORYBOARD D (CONTINUATION)

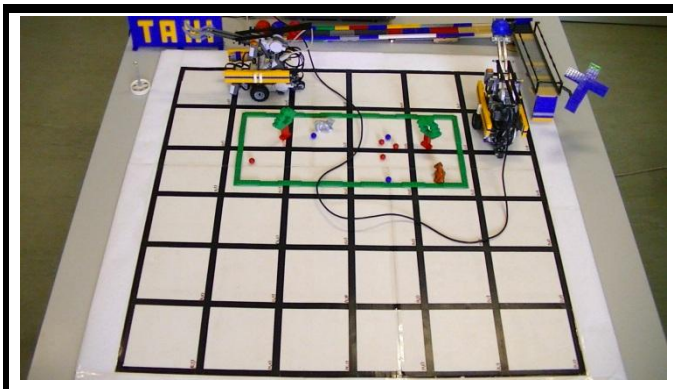
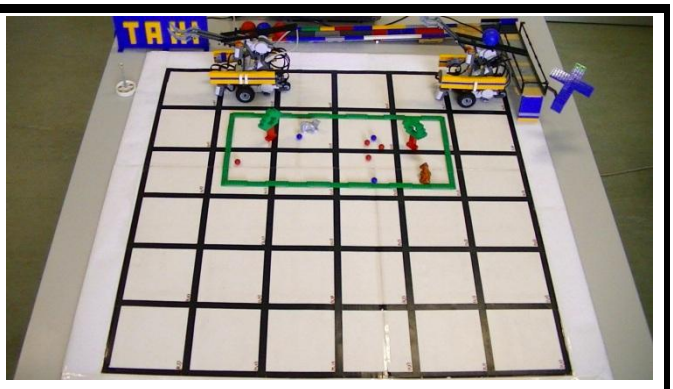
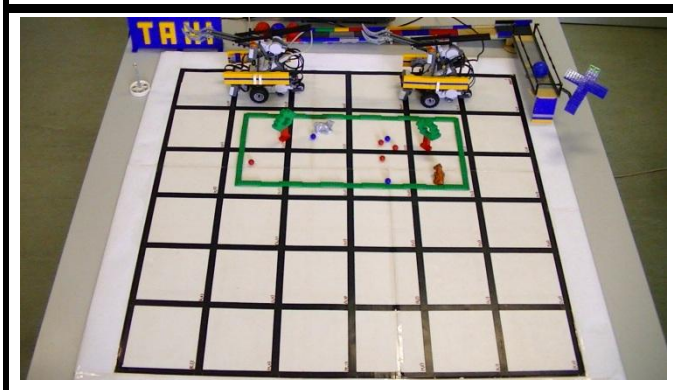
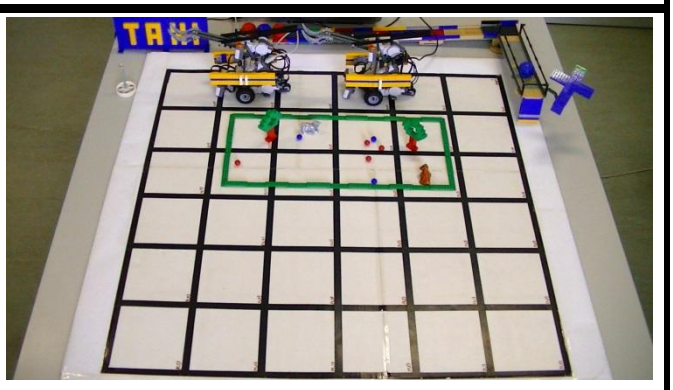
	
P17: T1 arrives to a new decision point (airport) after finishing its current action plan. A new belief is inserted ($airport <- true.$) in its private knowledge. The next intention is considered ($choose\_road3$) and a contextual query is submitted to the server. The answer is "INVALID", due to the contradictory beliefs. So, T1 revises its beliefs and the answer will be "NO". The process is repeated for the next intention ($drop\_passengers$) and now it is warranted by the server. T1's action plan $drop\_passengers(Robot)$ starts.	**P18**: During the current T1's action plan, $drop\_passengers(Robot)$, the agent will, obviously, drop the passengers at the airport, add the new belief $\sim have\_passengers <- true.$ to its private knowledge and then move to the next decision point.
	
P19: At this decision point, T1 will choose the next intention ($back\_to\_rank$) and the answer from the server will be "INVALID" due to contradictory beliefs. After revising its beliefs, T1 will, again, submit the contextual query to the server and receive an affirmative answer. Then the action plan $back\_to\_rank(Robot)$ will start.	**P20**: T2 will not arrive to next decision point, until T1 vacates it. Meanwhile, T2 will wait, during its actual action plan (it is the same situation presented in P1 from this scenario).

FIGURE 49 - STORYBOARD D (CONTINUATION)

9. CONCLUSION

In this work we have presented a concept of argumentative and deliberative robots, namely Argumentative NXT BDI-like Agents. To construct the architecture of these agents we combined the BDI model with two mechanisms: one for reasoning and the other for belief revision. The base of the reasoning mechanism is DeLP, which is a well-defined formalism. Even the agents' knowledge is represented through it. The revision process is based on kernel contraction and the criterion to eliminate beliefs is related to the primacy of the new information.

Based on this concept, two variations were presented: Single Argumentative NXT BDI-like Agent and MAS Argumentative NXT BDI-like Agent. The first one has a standalone perspective, i.e. it turned to himself and the second has a multi-agent perspective in a collaborative sense. In our opinion, if the modulation of the situations is done correctly, these agents are capable of solving many problems.

The application cases demonstrate how the Argumentative NXT BDI-like Agents can react to different situations. In the first application case it is possible to observe in a simple way how the Single Agent uses defeasible argumentation and how it revises its own beliefs. The second application case demonstrates how the MAS agents operate and how they take advantage of the public dynamic knowledge that is shared by other agents.

In future work, some aspects should be considered with the intention of improving the agents' architecture and behavior:

- Verify syntactic and semantic properties of the proposed system;
- Extend the revision process to rules;
- Add a reliability factor to the beliefs that comes from other agents;
- Consider reinstatement/recovery of beliefs;
- Implement the competitive sense with direct communication between agents.
- Use the TCP/IP protocol provided by the DeLP-Server to broaden the radius of action from the agents, i.e. build a distributed system where the agents can act in different environments and be connected with the same DeLP-Server.

Other technical aspects can also be considered. Such as the Bluetooth connection instead of USB (it is being developed by the Droide project), and also the possibility of adapting the Delp-Server to the more recent operating systems (Windows XP was used in this project).

10. BIBLIOGRAPHY

[1] J. McCarthy, "What Is Artificial Intelligence?," [Online]. Available: http://www-formal.stanford.edu/jmc/whatisai/whatisai.html.

[2] S. Russell and P. Norving, Artificial Intelligence - A Modern Approach, 3rd edition, Prentice Hall, 2010.

[3] M. Wooldridge and N. Jennings, "Intelligent agents: Theory and practice," *The Knowledge Engineering Review,* pp. 115-152, 1995.

[4] P. Maes, "The agent network architecture (ANA)," *SIGART Bull.,* vol. 2, pp. 115-120, 1991.

[5] M. Carvalho, "Agentes de Apoio à Argumentação e Decisão em Grupo," 2007.

[6] "http://www.agentlink.org," [Online].

[7] M. Wooldridge, An Introduction to MultiAgent Systems, John Wiley & Sons, 2002.

[8] L. P. Reis, *PhD Thesis, Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico, FEUP,* 2003.

[9] C. Rapp, "Aristotle's Rhetoric," The Stanford Encyclopedia of Philosophy (Spring 2010 Edition), [Online]. Available: http://plato.stanford.edu/archives/spr2010/entries/aristotle-rhetoric/.

[10] D. Walton, "Argumentation Theory: A Very Short Introduction," in *Argumentation in Artificial Intelligence*, Springer US, 2009, pp. 1-22.

[11] J. L. Pollock , Knowledge and Justification, Princeton University Press, 1974.

[12] D. Nute, "Defeasible Resoning," in *Aspects of Artificial Intelligence*, Noewell, Kluwer Academic Publishers, 1988, pp. 251-288.

[13] C. I. Chesñevar and G. R. Simari, "Computational Models for Argumentation in Multiagent Systems," in *EASSS*, Utrecht, Netherlands, 2005.

[14] C. I. Chesñevar, A. G. Maguitman and R. P. Loui, "Logical models of argument," *ACM Computing Surveys (CSUR),* vol. 32, no. 4, pp. 337-383, 2000.

[15] J. Doyle, "A Truth Maintenance System," in *Artificial Intelligence*, 1979, pp. 12:231-272.

[16] G. Simari and R. P. Loui, "A Mathematical Treatment of Defeasible Reasoning and its Implementation," *Artif. Intell.,* pp. 125-157, 1992.

[17] G. Vreeswijk, "Abstract argumentation systems," *Artif. Intell.,* pp. 225-279, 1997.

[18] P. M. Dung, "On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games," *Artif. Intell.,* pp. 321-358, 1995.

[19] A. J. García and G. R. Simari, "Defeasible logic programming: an argumentative approach," *Theory and Practice of Logic Programming,* vol. 4, pp. 95-138, 2004.

[20] I. Levi, "Subjunctives, dispositions and chances," *Synthese,* vol. 34, no. 4, pp. 423-455, 1977.

[21] I. Levi , The Enterprise of Knowledge, Cambridge: MIT Press, 1980.

[22] W. L. Harper, "Ramsey test conditionals abd iterated belief change," in *Foundations and Philosophy of Epistemic Applications of Probability Theory*, Dordrecht, D. Reidel Publishing Co., 1976, pp. 117-136.

[23] C. Alchourrón, P. Gärdenfors and D. Makinson, "On the logic of theory change: Partial meet contraction and revision functions," *The Journal of Symbolic Logic,* vol. 50, pp. 510-530, 1985.

[24] E. Fermé and S. Hansson, "AGM 25 Years," *Journal of Philosophical Logic,* vol. 40, no. 2, pp. 295-331, 2011.

[25] S. O. Hansson, "Logic of Belief Revision," in *The Stanford Encyclopedia of Philosophy (Fall 2011 Edition)*, E. N. Z. (ed.), Ed.

[26] E. L. Fermé, "Tópicos Avançados em Revisão de Crenças," 2008.

[27] C. Alchourrón and D. Makinson, "On the logic of theory change: Safe contraction.," *Studia Logica,* vol. 44, p. 405–422, 1985.

[28] S. O. Hansson, "Kernel contraction," *Journal of Symbolic Logic,* vol. 59, p. 845–859.

[29] S. O. Hansson, "A Survey of Non-Prioritized Belief Revision," *Erkenntnis,* vol. 50, pp. 413-427, 1999.

[30] S. O. Hansson, A Textbook of Belief Dynamics. Theory Change and Database Updating, Dordrecht: Kluwer Academic Publishers, 1999.

[31] H. Sven Ove, A Textbook of Belief Dynamics. Theory Change and Database Updating, Dordrecht: Kluwer Academic Publishers, 1999.

[32] M. A. Falappa, G. Kern-Isberner and G. R. Simari, "Belief revision and argumentation," in *Argumentation in artif icial intelligence*, US: Springer, 2009, p. 341–360.

[33] J. d. Kleer, "An assumption-based TMS," *Artificial Intelligence,* vol. 28, no. 2, pp. 127-162, 1986.

[34] J. L. Pollock and A. S. Gillies, "Belief Revision and Epistemology," *Synthese,* vol. 122, no. 1, pp. 69-92, 2000.

[35] M. A. Falappa, G. Kern-Isberner and G. R. Simari, "Belief Revision, Explanations and Defeasible Reasoning," *Artificial Intelligence,* vol. 141, pp. 1-28, 2002.

[36] F. Paglieri and C. Castelfranchi, "The Toulmin Test: Framing Argumentation within Belief Revision Theories,"

in *Arguing on the Toulmin Model*, Springer Netherlands, 2006, pp. 359-377.

[37] N. D. Rotstein, M. O. Moguillansky, M. A. Falappa, A. J. García and G. R. Simari, "Argument Theory Change: Revision Upon Warrant," in *Proceedings of the 2008 conference on Computational Models of Argument: Proceedings of COMMA 2008*, The Netherlands, 2008.

[38] M. O. Moguillansky, N. D. Rotstein, M. A. Falappa, A. J. García and G. R. Simari, "Argument theory change applied to defeasible logic programming," in *Proceedings of the 23rd conference on Artificial intelligence, AAAI 2008*, 2008.

[39] G. Boella, C. d. C. Perera, A. Tettamanzi and L. v. d. Torre, "Dung Argumentation and AGM Belief Revision," in *ArgMAS 2008*, Estoril, Portugal, 2008.

[40] A. Fuhrmann and S. O. Hansson, "A Survey of Multiple Contractions," *Journal of Logic, Language, and Information,* vol. 3, pp. 39-75, 1994.

[41] G. Boella, C. Pereira, A. Tettamanzi and L. van der Torre, "Making Others Believe What They Want," in *Artificial Intelligence in Theory and Practice II*, Springer Boston, 2008, pp. 215-224.

[42] M. Falappa, A. Garcia and G. Simari, "Belief dynamics and defeasible argumentation in rational agents," in *NMR*, 2004, pp. 164-170.

[43] M. E. Bratman, Intention, Plans, and Practical Reason, Cambridge, Massachusetts: Harvard University Press (Reprinted by CLSI Publications), 1987.

[44] I. O. d. Nunes, “Implementação do Modelo e da Arquitetura BDI,” PUC-Rio Departamento de Informática, Rio de Janeiro, Brasil, 2007.

[45] M. Wooldridge, Reasoning about Rational Agents, MIT Press, 2000.

[46] A. S. Rao e M. P. Georgeff, “BDI Agents: From Theory to Practice,” em *Proceedings of the First International Conference on Multiagent Systems*, San Francisco, 1995.

[47] M. Wooldridge, *Intelligent agents,* 1999.

[48] N. Howden, R. Rönnquista, A. Hodgson e A. Lucas, “Jack intelligent agents TM: Summary of an agent infrastructure,” em *The Fifth International Conference on Autonomous Agents*, Montreal, Canada, 2001.

[49] A. Pokahr, L. Braubach and W. Lamersd, "Jadex: Implementing a BDI-Infrastructure for JADE Agents," *EXP - in search of innovation,* vol. 3, pp. 76-85, 2003.

[50] M. J. Huber, "JAM: a BDI-theoretic mobile agent architecture," in *Proceedings of the third annual conference on Autonomous Agents*, 236-243, 1999.

[51] R. H. Bordini, J. F. Hübner e M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak using Jason, John Wiley & Sons, 2007.

[52] “3APL - An Abstract Agent Programming Language,” 2003. [Online]. Available: http://www.cs.uu.nl/3apl/.

[53] A. J. García, N. D. Rotstein, M. Tucat and G. R. Simari, "An argumentative reasoning service for deliberative agents," in *Proceedings of the 2nd international conference on Knowledge science, engineering and management*, 2007.

[54] “LEGO.com MINDSTORMS : Home,” LEGO, [Online]. Available: http://mindstorms.lego.com/en-us/Default.aspx.

[55] “SWI-Prolog's home,” SWI-Prolog, [Online]. Available: http://www.swi-prolog.org/.

[56] "Droide M.L.P. Documentation," [Online]. Available: http://dme.uma.pt/projects/droide/nxtsdk_doc/index.html.

[57] "Droide M.L.P. Software," [Online]. Available: http://dme.uma.pt/projects/droide/nxtsdk_soft/index.html.

[58] S. Gottifredi, A. Garcia and G. Simari, "Defeasible Knowledge and Argumentative Reasoning for 3APL Agent Programming," in *Declarative Programming Paradigms and Systems for NMR*, 2008, pp. 170-178.

[59] T. J. M. Bench-Capon and P. E. Dunne, "Argumentation in artificial intelligence," *Artif. Intell.,* vol. 171, pp. 619-641, 2007.

[60] G. Simari and C. I. Chesñevar, "Computational Models for Argumentation in Multiagent Systems," in *EASSS*, 2005.

11. APPENDIX A

This appendix has the content of the four main files that modulate the problem of the application case: Single Taxi.

11.1. BELIEFS

The content of the file “beliefs.delp” is presented on the following page and has been printed directly from the IDE.

```
1   wait_passengers -< rank,~have_passengers.
2   ~wait_passengers -< ~rank,have_passengers.
3   ~wait_passengers -< ~rank,~have_passengers,airport.
4   choose_road1 -< have_passengers,~rank,~airport,~traffic1.
5   choose_road1 -<
    have_passengers,~rank,~airport,traffic1,traffic2,traffic3.
6   ~choose_road1 -< have_passengers,~rank,traffic1.
7   ~choose_road1 -< ~have_passengers,~rank.
8   choose_road2 -< have_passengers,~rank,traffic1,~airport,~traffic2.
9   ~choose_road2 -< have_passengers,~rank,traffic1,airport.
10  ~choose_road2 -< have_passengers,~rank,traffic1,traffic2.
11  ~choose_road2 -< have_passengers,~rank,airport.
12  ~choose_road2 -< ~have_passengers,~rank.
13  choose_road3 -< have_passengers,~rank,traffic1,traffic2,~airport.
14  ~choose_road3 -< have_passengers,~rank,traffic1,traffic2,airport.
15  ~choose_road3 -< have_passengers,~rank,airport.
16  ~choose_road3 -< ~have_passengers,~rank.
17  drop_passengers -< have_passengers,airport.
18  ~drop_passengers -< ~have_passengers,~rank.
19  back_to_rank -< airport,~have_passengers.
20  traffic1 -< rush_hour.
21  ~traffic1 -< ~rush_hour.
22  ~have_passengers <- true.
23  ~airport <- true.
24  rank <- true.
25
```

11.2. INTENTIONS

The content of the file “intentions.txt” is presented on the following page and has been printed directly from the IDE.

```
1  wait_passengers
2  choose_road1
3  choose_road2
4  choose_road3
5  drop_passengers
6  back_to_rank
```

11.3. ACTION PLANS

The content of the file “actionPlans.pl” is presented on the following pages and has been printed directly from the IDE.

```
1   %%%%%%%%%%%%%%%%%%%%%%% MAIN ACTION PLANS %%%%%%%%%%%%%%%%%%%%%%%
2   % Actions for the Wait Passengers Intention
3   wait_passengers(Robot):-
4       write('Actions for WaitPassengers: Check Passengers, Pick up
        Passangers, Start Ride.'),nl,nl,
5       checkPassengers(Robot),
6       pickPassengers(Robot),
7       updateBeliefsWaitPassengers,
8       startRide(Robot).
9
10  % Actions for the Choose Road 1 Intention
11  choose_road1(Robot):-
12      write('Actions for ROAD 1:  Move on to Road 1, Check traffic.'
        ),nl,nl,
13      moveonRoad1(Robot),
14      turnLeft(Robot),
15      moveSquares(Robot,2),
16      updateRoad1airport.
17
18  % Actions for the Choose Road 2 Intention
19  choose_road2(Robot):-
20      write('Actions for ROAD 2:  Move on to Road 2, Check traffic.'
        ),nl,nl,
21      moveonRoad2(Robot),
22      turnLeft(Robot),
23      moveSquares(Robot,3),
24      updateRoad2airport.
25
26  % Actions for the Choose Road 3 Intention
27  choose_road3(Robot):-
28      write('Actions for ROAD 3:  Move on to Road 3, Check traffic.'
        ),nl,nl,
29      moveonRoad3(Robot),
30      turnLeft(Robot),
31      moveSquares(Robot,4),
32      updateRoad3airport.
33
34  % Actions for the Drop Passengeres Intention
35  drop_passengers(Robot):-
36      write('Actions for DROP OFF PASSENGERS:  let passengers out ,
        turn back.'),nl,nl,
37      turnLeft(Robot),
38      upclaw(Robot),
39      moveSquares(Robot,1),
40      downclaw(Robot),
41      updatePassengersOff.
```

```
42
43  % Actions for the Back To rank Intention
44  back_to_rank(Robot):-
45      write('Actions for BACK TO rank:  MOVE ON TO THE TAXI LINE.'),
        nl,nl,
46      moveSquares(Robot,3),
47      updaterank,
48      write('Ride ends here.'),nl.
49  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50
51  %%%%%%%%%%%%%%%%%%%%%% OTHER ACTION PLANS %%%%%%%%%%%%%%%%%%%%%%
52  % Loop function that will be waiting for the ultrasound or touch
    sensor
53  checkPassengers(Robot):-
54      nxt_pl_read(Robot,2,Result),Result < 30,
55      write('Now have passengers!'),nl,nl.
56
57  checkPassengers(Robot):-
58      nxt_pl_isPressed(Robot,4,Result),Result = @(true),
59      nxt_pl_shutdown(Robot),
60      halt.
61
62  checkPassengers(Robot):-checkPassengers(Robot).
63
64  % Actions to pick up passengers
65  pickPassengers(Robot):-
66      downclaw(Robot),
67      moveSquares(Robot,1),
68      upclaw(Robot),
69      nxt_pl_backwardDegrees(Robot,15,90).
70
71  % Updating new beliefs
72  updateBeliefsWaitPassengers:-
73      open('beliefs.delp', append, Stream),
74      write(Stream, ('have_passengers <- true.')),
75      nl(Stream),
76      write(Stream, ('~rank <- true.')),
77      nl(Stream),
78      close(Stream).
79
80  % Actions to start ride until choose road
81  startRide(Robot):-
82      write('Now Starting Ride.'),nl,nl,
83      nxt_pl_turnLeftTurns(Robot, 20, 1),
84      moveSquares(Robot,2).
85
```

```
86  % Moves claw down
87  downclaw(Robot):-
88      nxt_pl_moveDegrees(Robot,'c',28,45).
89
90  % Moves claw up
91  upclaw(Robot):-
92      nxt_pl_moveDegrees(Robot,'c',-28,45).
93
94  % Actions to follown on Road 1
95  % X represents the number of squares - NT represents Not Traffic
    variable
96  moveonRoad1(Robot):-
97      X is 0,
98      NT is 0,
99      turnLeft(Robot),
100     checkTraficAndLinesRoad1(Robot,X,NT).
101
102 % Loop function that stops at the end of the road
103 checkTraficAndLinesRoad1(Robot,X,NT):-
104     X < 4,
105     nxt_pl_read(Robot,2,Result),Result > 30,
106     nxt_pl_forward(Robot,10),
107     traficlightblackvalueRoad1(Robot,X,NT).
108
109 checkTraficAndLinesRoad1(Robot,X,NT):-
110     X = 4,
111     updateRoad1NOTTraffic1(NT),
112     cls(Robot),
113     nxt_pl_write(Robot,'Ending ride'),
114     write('Ending Ride.'),nl,nl.
115
116 % Loop function to be always alert to traffic and count squares
117 traficlightblackvalueRoad1(Robot,X,NT):-
118     nxt_pl_read(Robot,2,Result),Result < 30,
119     nxt_pl_stop(Robot),
120     updateRoad1Traffic1,
121     inc0(X,A,NTF),
122     %Do not icrement X, but set NT,
123     checkTraficAndLinesRoad1(Robot,A,NTF).
124
125 traficlightblackvalueRoad1(Robot,X,NT):-
126     nxt_pl_read(Robot,1,Result),Result > 550,
127     %nxt_pl_stop(Robot),
128     inc(X,A),
129     checkTraficAndLinesRoad1(Robot,A,NT).
130
```

```
131  traficlightblackvalueRoad1(Robot,X,NT):-traficlightblackvalueRoad1(
     Robot,X,NT).
132
133  % Adds 1 to X
134  inc(X,Z):-
135      Z is X+1.
136
137  % Adds 0 to X and set NT (not traffic).
138  inc0(X,Y,NT):-
139      Z is X+1,
140      Y is Z-1,
141      NT is 1.
142      %write(Z),nl.
143
144  % Updating new beliefs
145  updateRoad1airport:-
146      open('beliefs.delp', append, Stream),
147      write(Stream, ('airport <- true.')),
148      nl(Stream),
149      close(Stream).
150
151  % Updating new beliefs
152  updateRoad1Traffic1:-
153      open('beliefs.delp', append, Stream),
154      write(Stream, ('traffic1 <- true.')),
155      nl(Stream),
156      close(Stream).
157
158  % Updating new beliefs
159  updateRoad1NOTTraffic1(NT):-
160      NT = 0,
161      open('beliefs.delp', append, Stream),
162      write(Stream, ('~traffic1 <- true.')),
163      nl(Stream),
164      close(Stream),
165      write('At Road 1 there is no traffic.').
166
167  % Information menssage
168  updateRoad1NOTTraffic1(NT):-
169      NT = 1,
170      write('At Road 1 there is traffic.').
171
172  % Moves 1 Square until black line
173  move1Square(Robot):-
174      nxt_pl_forward(Robot,10),
175      blackLineStop(Robot).
```

```
176
177   % Moves K Squares
178   moveSquares(Robot,K):-
179       K = 1,
180       nxt_pl_forward(Robot,10),
181       blackLine(Robot),
182       nxt_pl_stop(Robot),
183       !
184       ;
185       K > 1,
186       nxt_pl_forward(Robot,10),
187       blackLine(Robot),
188       K1 is K - 1,
189       moveSquares(Robot,K1).
190
191   % Loop function to check black line
192   blackLine(Robot):-
193       nxt_pl_read(Robot,1,Result),Result > 550.
194
195   blackLine(Robot):-blackLine(Robot).
196
197   % Loop function to check black line
198   blackLineStop(Robot):-
199       nxt_pl_read(Robot,1,Result),Result > 550,
200       nxt_pl_stop(Robot).
201
202   blackLineStop(Robot):-blackLineStop(Robot).
203
204   % Function to turn left
205   turnLeft(Robot):-
206       nxt_pl_forwardDegrees(Robot,10,180),
207       nxt_pl_turnLeftTurns(Robot, 20, 1).
208
209   % Actions to followon Road 1
210   % X represents the number of squares - NT represents Not Traffic
      variable
211   moveonRoad2(Robot):-
212       X is 0,
213       NT is 0,
214       moveSquares(Robot,1),
215       turnLeft(Robot),
216       checkTraficAndLinesRoad2(Robot,X,NT).
217
218   % Loop function that stops at the end of the road
219   checkTraficAndLinesRoad2(Robot,X,NT):-
220       X < 4,
```

```
221       nxt_pl_read(Robot,2,Result),Result > 30,
222       nxt_pl_forward(Robot,10),
223       traficlightblackvalueRoad2(Robot,X,NT).
224
225   checkTraficAndLinesRoad2(Robot,X,NT):-
226       X = 4,
227       updateRoad2NOTTraffic2(NT),
228       cls(Robot),
229       nxt_pl_write(Robot,'Ending ride'),
230       write('Ending Ride.'),nl,nl.
231
232   % Loop function to be always alert to traffic and count squares
233   traficlightblackvalueRoad2(Robot,X,NT):-
234       nxt_pl_read(Robot,2,Result),Result < 30,
235       nxt_pl_stop(Robot),
236       updateRoad2Traffic2,
237       inc0(X,A,NTF),
238       %Do not icrement X, but set NT,
239       checkTraficAndLinesRoad2(Robot,A,NTF).
240
241   traficlightblackvalueRoad2(Robot,X,NT):-
242       nxt_pl_read(Robot,1,Result),Result > 550,
243       inc(X,A),
244       checkTraficAndLinesRoad2(Robot,A,NT).
245
246   traficlightblackvalueRoad2(Robot,X,NT):-traficlightblackvalueRoad2(
      Robot,X,NT).
247
248   % Updating new beliefs
249   updateRoad2airport:-
250       open('beliefs.delp', append, Stream),
251       write(Stream, ('airport <- true.')),
252       nl(Stream),
253       close(Stream).
254
255   % Updating new beliefs
256   updateRoad2Traffic2:-
257       open('beliefs.delp', append, Stream),
258       write(Stream, ('traffic2 <- true.')),
259       nl(Stream),
260       close(Stream).
261
262   % Updating new beliefs
263   updateRoad2NOTTraffic2(NT):-
264       NT = 0,
265       open('beliefs.delp', append, Stream),
```

```
266         write(Stream, ('~traffic2 <- true.')),
267         nl(Stream),
268         close(Stream),
269         write('At Road 2 there is no traffic.').
270
271     % Information menssage
272     updateRoad2NOTTraffic2(NT):-
273         NT = 1,
274         write('At Road 2 there is traffic.').
275
276     % Actions to followon on Road 1
277     % X represents the number of squares - NT represents Not Traffic
        variable
278     moveonRoad3(Robot):-
279         X is 0,
280         NT is 0,
281         moveSquares(Robot,2),
282         turnLeft(Robot),
283         checkTraficAndLinesRoad3(Robot,X,NT).
284
285     % Loop function that stops at the end of the road
286     checkTraficAndLinesRoad3(Robot,X,NT):-
287         X < 4,
288         nxt_pl_read(Robot,2,Result),Result > 30,
289         nxt_pl_forward(Robot,10),
290         traficlightblackvalueRoad3(Robot,X,NT).
291
292     checkTraficAndLinesRoad3(Robot,X,NT):-
293         X = 4,
294         updateRoad3NOTTraffic3(NT),
295         cls(Robot),
296         nxt_pl_write(Robot,'Ending ride'),
297         write('Ending Ride.'),nl,nl.
298
299     % Loop function to be always alert to traffic and count squares
300     traficlightblackvalueRoad3(Robot,X,NT):-
301         nxt_pl_read(Robot,2,Result),Result < 30,
302         nxt_pl_stop(Robot),
303         updateRoad3Traffic3,
304         inc0(X,A,NTF),
305         %Do not icrement X, but set NT,
306         checkTraficAndLinesRoad3(Robot,A,NTF).
307
308     traficlightblackvalueRoad3(Robot,X,NT):-
309         nxt_pl_read(Robot,1,Result),Result > 550,
310         inc(X,A),
```

```
311       checkTraficAndLinesRoad3(Robot,A,NT).
312
313   traficlightblackvalueRoad3(Robot,X,NT):-traficlightblackvalueRoad3(
      Robot,X,NT).
314
315   % Updating new beliefs
316   updateRoad3airport:-
317       open('beliefs.delp', append, Stream),
318       write(Stream, ('airport <- true.')),
319       nl(Stream),
320       close(Stream).
321
322   % Updating new beliefs
323   updateRoad3Traffic3:-
324       open('beliefs.delp', append, Stream),
325       write(Stream, ('traffic3 <- true.')),
326       nl(Stream),
327       close(Stream).
328
329   % Updating new beliefs
330   updateRoad3NOTTraffic3(NT):-
331       NT = 0,
332       open('beliefs.delp', append, Stream),
333       write(Stream, ('~traffic3 <- true.')),
334       nl(Stream),
335       close(Stream),
336       write('At Road 3 there is no traffic.').
337
338   % Information menssage
339   updateRoad3NOTTraffic3(NT):-
340       NT = 1,
341       write('At Road 3 there is traffic.').
342
343   % Updating new beliefs
344   updatePassengersOff:-
345       open('beliefs.delp', append, Stream),
346       write(Stream, ('~have_passengers <- true.')),
347       nl(Stream),
348       close(Stream).
349
350   % Updating new beliefs
351   updaterank:-
352       open('beliefs.delp', append, Stream),
353       write(Stream, ('~airport <- true.')),
354       nl(Stream),
355       write(Stream, ('rank <- true.')),
```

```
356         nl(Stream),
357         close(Stream).
358
359     % Clears NXT Screen
360     cls(Robot):-
361         nxt_pl_clear(Robot).
362     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

11.4. HUB

The content of the file “hub.pl” is presented on the following page and has been printed directly from the IDE.

```
1   :-[actionPlans].
2
3   % This function is a switch that associate the intentions (queries)
     to the respective action plans
4   checkQuery(Query, Robot):-
5           Query = 'wait_passengers',
6           wait_passengers(Robot),!
7           ;
8           Query = 'choose_road1',
9           choose_road1(Robot),!
10          ;
11          Query = 'choose_road2',
12          choose_road2(Robot),!
13          ;
14          Query = 'choose_road3',
15          choose_road3(Robot),!
16          ;
17          Query = 'drop_passengers',
18          drop_passengers(Robot),!
19          ;
20          Query = 'back_to_rank',
21          back_to_rank(Robot).
```

12. APPENDIX B

This appendix has the content of the files that modulate the problem of the application case: Taxi Company.

12.1. BELIEFS

The content of the file “beliefs.delp” is presented on the following page and has been printed directly from the IDE.

```
1   wait_passengers -< rank,~have_passengers.
2   ~wait_passengers -< ~rank,have_passengers.
3   ~wait_passengers -< ~rank,~have_passengers,airport.
4   choose_road1 -< have_passengers,~rank,~airport,~traffic1.
5   choose_road1 -<
    have_passengers,~rank,~airport,traffic1,traffic2,traffic3.
6   ~choose_road1 -< have_passengers,~rank,traffic1.
7   ~choose_road1 -< ~have_passengers,~rank.
8   choose_road2 -< have_passengers,~rank,traffic1,~airport,~traffic2.
9   ~choose_road2 -< have_passengers,~rank,traffic1,airport.
10  ~choose_road2 -< have_passengers,~rank,traffic1,traffic2.
11  ~choose_road2 -< have_passengers,~rank,airport.
12  ~choose_road2 -< ~have_passengers,~rank.
13  choose_road3 -< have_passengers,~rank,traffic1,traffic2,~airport.
14  ~choose_road3 -< have_passengers,~rank,traffic1,traffic2,airport.
15  ~choose_road3 -< have_passengers,~rank,airport.
16  ~choose_road3 -< ~have_passengers,~rank.
17  drop_passengers -< have_passengers,airport.
18  ~drop_passengers -< ~have_passengers,~rank.
19  back_to_rank -< airport,~have_passengers.
20  traffic1 -< rush_hour.
21  ~traffic1 -< ~rush_hour.
22
```

12.2. BELIEFS' TIMES

The content of the file "beliefsTime.txt" is presented on the following page and has been printed directly from the IDE.

```
1   00,00,00.000
2   00,00,00.000
3   00,00,00.000
4   00,00,00.000
5   00,00,00.000
6   00,00,00.000
7   00,00,00.000
8   00,00,00.000
9   00,00,00.000
10  00,00,00.000
11  00,00,00.000
12  00,00,00.000
13  00,00,00.000
14  00,00,00.000
15  00,00,00.000
16  00,00,00.000
17  00,00,00.000
18  00,00,00.000
19  00,00,00.000
20  00,00,00.000
21  00,00,00.000
22
```

12.3. INTENTIONS

The content of the file “intentions.txt” is presented on the following page and has been printed directly from the IDE.

```
1  wait_passengers
2  choose_road1
3  choose_road2
4  choose_road3
5  drop_passengers
6  back_to_rank
```

12.4. ACTION PLANS

The content of the file “actionPlans.pl” is presented on the following pages and has been printed directly from the IDE.

```
1   %%%%%%%%%%%%%%%%%%%%%%%% MAIN ACTION PLANS %%%%%%%%%%%%%%%%%%%%%%%%
2   % Actions for the Wait Passengers Intention
3   wait_passengers(Robot):-
4       write('Actions for WaitPassengers: Check Passengers, Pick up
        Passangers, Start Ride.'),nl,nl,
5       checkPassengers(Robot),
6       pickPassengers(Robot),
7       updateBeliefsWaitPassengers,
8       startRide(Robot).
9
10  % Actions for the Choose Road 1 Intention
11  choose_road1(Robot):-
12      write('Actions for ROAD 1:  Move on to Road 1, Check traffic.'
        ),nl,nl,
13      moveonRoad1(Robot),
14      turnLeft(Robot),
15      moveSquares(Robot,2),
16      updateRoad1airport.
17
18  % Actions for the Choose Road 2 Intention
19  choose_road2(Robot):-
20      write('Actions for ROAD 2:  Move on to Road 2, Check traffic.'
        ),nl,nl,
21      moveonRoad2(Robot),
22      turnLeft(Robot),
23      moveSquares(Robot,3),
24      updateRoad2airport.
25
26  % Actions for the Choose Road 3 Intention
27  choose_road3(Robot):-
28      write('Actions for ROAD 3:  Move on to Road 3, Check traffic.'
        ),nl,nl,
29      moveonRoad3(Robot),
30      turnLeft(Robot),
31      moveSquares(Robot,4),
32      updateRoad3airport.
33
34  % Actions for the Drop Passengeres Intention
35  drop_passengers(Robot):-
36      write('Actions for DROP OFF PASSENGERS:  let passengers out ,
        turn back.'),nl,nl,
37      turnLeft(Robot),
38      upclaw(Robot),
39      moveSquares(Robot,1),
40      downclaw(Robot),
41      updatePassengersOff.
```

```
42
43  % Actions for the Back To rank Intention
44  back_to_rank(Robot):-
45      write('Actions for BACK TO rank:  MOVE ON TO THE TAXI LINE.'),
        nl,nl,
46      moveSquares(Robot,3),
47      updaterank,
48      write('Ride ends here.'),nl.
49  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50
51  %%%%%%%%%%%%%%%%%%%%%% OTHER ACTION PLANS %%%%%%%%%%%%%%%%%%%%%%
52  % Loop function that will be waiting for the ultrasound or touch
    sensor
53  checkPassengers(Robot):-
54      nxt_pl_read(Robot,2,Result),Result < 30,
55      write('Now have passengers!'),nl,nl.
56
57  checkPassengers(Robot):-
58      nxt_pl_isPressed(Robot,4,Result),Result = @(true),
59      nxt_pl_shutdown(Robot),
60      halt.
61
62  checkPassengers(Robot):-checkPassengers(Robot).
63
64  % Actions to pick up passengers
65  pickPassengers(Robot):-
66      downclaw(Robot),
67      move1Square(Robot),
68      upclaw(Robot),
69      nxt_pl_backwardDegrees(Robot,15,90).
70
71  % Updating new beliefs
72  updateBeliefsWaitPassengers:-
73      updateBelief('beliefs.delp','have_passengers <- true.'),
74      updateTime('beliefsTime.txt'),
75      updateBelief('beliefs.delp','~rank <- true.'),
76      updateTime('beliefsTime.txt'),
77      updateBelief('beliefs.delp','~airport <- true.'),
78      updateTime('beliefsTime.txt').
79
80  % Actions to start ride until choose road
81  startRide(Robot):-
82      write('Now Starting Ride.'),nl,nl,
83      nxt_pl_turnLeftTurns(Robot, 20, 1),
84      moveSquares(Robot,2).
85
```

```
86  % Moves claw down
87  downclaw(Robot):-
88      nxt_pl_moveDegrees(Robot,'c',28,45).
89
90  % Moves claw up
91  upclaw(Robot):-
92      nxt_pl_moveDegrees(Robot,'c',-28,45).
93
94  % Actions to follown on Road 1
95  % X represents the number of squares - NT represents Not Traffic
    variable
96  moveonRoad1(Robot):-
97      X is 0,
98      NT is 0,
99      turnLeft(Robot),
100     checkTraficAndLinesRoad1(Robot,X,NT).
101
102 % Loop function that stops at the end of the road
103 checkTraficAndLinesRoad1(Robot,X,NT):-
104     X < 4,
105     nxt_pl_read(Robot,2,Result),Result > 30,
106     nxt_pl_forward(Robot,10),
107     traficlightblackvalueRoad1(Robot,X,NT).
108
109 checkTraficAndLinesRoad1(Robot,X,NT):-
110     X = 4,
111     updateRoad1NOTTraffic1(NT),
112     cls(Robot),
113     nxt_pl_write(Robot,'Ending ride'),
114     write('Ending Ride.'),nl,nl.
115
116 % Loop function to be always alert to traffic and count squares
117 traficlightblackvalueRoad1(Robot,X,NT):-
118     nxt_pl_read(Robot,2,Result),Result < 30,
119     nxt_pl_stop(Robot),
120     updateRoad1Traffic1,
121     inc0(X,A,NTF),
122     %Do not icrement X, but set NT,
123     checkTraficAndLinesRoad1(Robot,A,NTF).
124
125 traficlightblackvalueRoad1(Robot,X,NT):-
126     nxt_pl_read(Robot,1,Result),Result > 550,
127     inc(X,A),
128     checkTraficAndLinesRoad1(Robot,A,NT).
129
130 traficlightblackvalueRoad1(Robot,X,NT):-traficlightblackvalueRoad1(
```

```
    Robot,X,NT).
131
132 % Adds 1 to X
133 inc(X,Z):-
134     Z is X+1.
135
136 % Adds 0 to X and set NT (not traffic).
137 inc0(X,Y,NT):-
138     Z is X+1,
139     Y is Z-1,
140     NT is 1.
141     %write(Z),nl.
142
143 % Updating new beliefs
144 updateRoad1Traffic1:-
145     updateBelief('beliefs.delp','traffic1 <- true.'),
146     updateTime('beliefsTime.txt'),
147     updateBelief('../publicDynamicKnowledge.delp','traffic1 <-
        true.'),
148     updateTime('../publicDynamicKnowledgeTime.txt').
149
150 % Updating new beliefs
151 updateRoad1airport:-
152     updateBelief('beliefs.delp','airport <- true.'),
153     updateTime('beliefsTime.txt').
154
155 % Updating new beliefs
156 updateRoad1NOTTraffic1(NT):-
157     NT = 0,
158     updateBelief('beliefs.delp','~traffic1 <- true.'),
159     updateTime('beliefsTime.txt'),
160     updateBelief('../publicDynamicKnowledge.delp','~traffic1 <-
        true.'),
161     updateTime('../publicDynamicKnowledgeTime.txt'),
162     write('At Road 1 there is no traffic.').
163
164 % Information menssage
165 updateRoad1NOTTraffic1(NT):-
166     NT = 1,
167     write('At Road 1 there is traffic.').
168
169 % Moves 1 Square until black line
170 move1Square(Robot):-
171     nxt_pl_forward(Robot,10),
172     blackLineStop(Robot).
173
```

```
174  % Moves K Squares
175  moveSquares(Robot,K):-
176      K = 0,
177      nxt_pl_stop(Robot),
178      write('o move squares chegou ao fim').
179
180  moveSquares(Robot,K):-
181      K > 0,
182      write('Está no quadrado:'),
183      write(K),
184      nxt_pl_read(Robot,2,Result),Result > 30,
185      nxt_pl_forward(Robot,10),
186      blackLineAndTraffic(Robot,K).
187
188  moveSquares(Robot,K):-moveSquares(Robot,K).
189
190  % Loop function to check black line and traffic
191  blackLineAndTraffic(Robot,K):-
192      nxt_pl_read(Robot,2,Result),Result < 30,
193      nxt_pl_stop(Robot),
194      moveSquares(Robot,K).
195
196  blackLineAndTraffic(Robot,K):-
197      nxt_pl_read(Robot,1,Result),Result > 550,
198      K1 is K - 1,
199      moveSquares(Robot,K1).
200
201  blackLineAndTraffic(Robot,K):-blackLineAndTraffic(Robot,K).
202
203  % Loop function to check black line
204  blackLineStop(Robot):-
205      nxt_pl_read(Robot,1,Result),Result > 550,
206      nxt_pl_stop(Robot).
207
208  blackLineStop(Robot):-blackLineStop(Robot).
209
210  % Function to turn left
211  turnLeft(Robot):-
212      nxt_pl_forwardDegrees(Robot,10,180),
213      nxt_pl_turnLeftTurns(Robot, 20, 1).
214
215  % Actions to follown on Road 1
216  % X represents the number of squares - NT represents Not Traffic
     variable
217  moveonRoad2(Robot):-
218      X is 0,
```

```
219         NT is 0,
220         moveSquares(Robot,1),
221         turnLeft(Robot),
222         checkTraficAndLinesRoad2(Robot,X,NT).
223
224     % Loop function that stops at the end of the road
225     checkTraficAndLinesRoad2(Robot,X,NT):-
226         X < 4,
227         nxt_pl_read(Robot,2,Result),Result > 30,
228         nxt_pl_forward(Robot,10),
229         traficlightblackvalueRoad2(Robot,X,NT).
230
231     checkTraficAndLinesRoad2(Robot,X,NT):-
232         X = 4,
233         updateRoad2NOTTraffic2(NT),
234         cls(Robot),
235         nxt_pl_write(Robot,'Ending ride'),
236         write('Ending Ride.'),nl,nl.
237
238     % Loop function to be always alert to traffic and count squares
239     traficlightblackvalueRoad2(Robot,X,NT):-
240         nxt_pl_read(Robot,2,Result),Result < 30,
241         nxt_pl_stop(Robot),
242         updateRoad2Traffic2,
243         inc0(X,A,NTF),
244         %Do not icrement X, but set NT,
245         checkTraficAndLinesRoad2(Robot,A,NTF).
246
247     traficlightblackvalueRoad2(Robot,X,NT):-
248         nxt_pl_read(Robot,1,Result),Result > 550,
249         inc(X,A),
250         checkTraficAndLinesRoad2(Robot,A,NT).
251
252     traficlightblackvalueRoad2(Robot,X,NT):-traficlightblackvalueRoad2(
        Robot,X,NT).
253
254     % Updating new beliefs
255     updateRoad2airport:-
256         updateBelief('beliefs.delp','airport <- true.'),
257         updateTime('beliefsTime.txt').
258
259     % Updating new beliefs
260     updateRoad2Traffic2:-
261         updateBelief('beliefs.delp','traffic2 <- true.'),
262         updateTime('beliefsTime.txt'),
263         updateBelief('../publicDynamicKnowledge.delp','traffic2 <-
```

```
        true.'),
264     updateTime('../publicDynamicKnowledgeTime.txt').
265
266 % Updating new beliefs
267 updateRoad2NOTTraffic2(NT):-
268     NT = 0,
269     updateBelief('beliefs.delp','~traffic2 <- true.'),
270     updateTime('beliefsTime.txt'),
271     updateBelief('../publicDynamicKnowledge.delp','~traffic2 <-
        true.'),
272     updateTime('../publicDynamicKnowledgeTime.txt'),
273     write('At Road 2 there is no traffic.').
274
275 % Information menssage
276 updateRoad2NOTTraffic2(NT):-
277     NT = 1,
278     write('At Road 2 there is traffic.').
279
280 % Actions to follown on Road 1
281 % X represents the number of squares - NT represents Not Traffic
    variable
282 moveonRoad3(Robot):-
283     X is 0,
284     NT is 0,
285     moveSquares(Robot,2),
286     turnLeft(Robot),
287     checkTraficAndLinesRoad3(Robot,X,NT).
288
289 % Loop function that stops at the end of the road
290 checkTraficAndLinesRoad3(Robot,X,NT):-
291     X < 4,
292     nxt_pl_read(Robot,2,Result),Result > 30,
293     nxt_pl_forward(Robot,10),
294     traficlightblackvalueRoad3(Robot,X,NT).
295
296 checkTraficAndLinesRoad3(Robot,X,NT):-
297     X = 4,
298     updateRoad3NOTTraffic3(NT),
299     cls(Robot),
300     nxt_pl_write(Robot,'Ending ride'),
301     write('Ending Ride.'),nl,nl.
302
303 % Loop function to be always alert to traffic and count squares
304 traficlightblackvalueRoad3(Robot,X,NT):-
305     nxt_pl_read(Robot,2,Result),Result < 30,
306     nxt_pl_stop(Robot),
```

```
307       updateRoad3Traffic3,
308       inc0(X,A,NTF),
309       %Do not icrement X, but set NT,
310       checkTraficAndLinesRoad3(Robot,A,NTF).
311
312   traficlightblackvalueRoad3(Robot,X,NT):-
313       nxt_pl_read(Robot,1,Result),Result > 550,
314       %nxt_pl_stop(Robot),
315       inc(X,A),
316       checkTraficAndLinesRoad3(Robot,A,NT).
317
318   traficlightblackvalueRoad3(Robot,X,NT):-traficlightblackvalueRoad3(
      Robot,X,NT).
319
320   % Updating new beliefs
321   updateRoad3airport:-
322       updateBelief('beliefs.delp','airport <- true.'),
323       updateTime('beliefsTime.txt').
324
325   % Updating new beliefs
326   updateRoad3Traffic3:-
327       updateBelief('beliefs.delp','traffic3 <- true.'),
328       updateTime('beliefsTime.txt'),
329       updateBelief('../publicDynamicKnowledge.delp','traffic3 <-
          true.'),
330       updateTime('../publicDynamicKnowledgeTime.txt').
331
332   % Updating new beliefs
333   updateRoad3NOTTraffic3(NT):-
334       NT = 0,
335       updateBelief('beliefs.delp','~traffic3 <- true.'),
336       updateTime('beliefsTime.txt'),
337       updateBelief('../publicDynamicKnowledge.delp','~traffic3 <-
          true.'),
338       updateTime('../publicDynamicKnowledgeTime.txt'),
339       write('At Road 3 there is no traffic.').
340
341   % Information menssage
342   updateRoad3NOTTraffic3(NT):-
343       NT = 1,
344       write('At Road 3 there is traffic.').
345
346   % Updating new beliefs
347   updatePassengersOff:-
348       updateBelief('beliefs.delp','~have_passengers <- true.'),
349       updateTime('beliefsTime.txt').
```

```

% Updating new beliefs
updaterank:-
    updateBelief('beliefs.delp','~airport <- true.'),
    updateTime('beliefsTime.txt'),
    updateBelief('beliefs.delp','rank <- true.'),
    updateTime('beliefsTime.txt').

% Clears NXT Screen
cls(Robot):-
    nxt_pl_clear(Robot).

% Given a file, this function will insert a new belief
updateBelief(File,Belief):-
    open(File, append, Stream),
    write(Stream, (Belief)),
    nl(Stream),
    close(Stream).

% Given a file, this function will insert the actual time
updateTime(File):-
    get_time(T),
    stamp_date_time(T, date(_, _, _, H, M, S, _, _, _), 'UTC'),
    open(File, append, Stream),
    write(Stream, (H)),
    write(Stream, (',')),
    write(Stream, (M)),
    write(Stream, (',')),
    write(Stream, (S)),
    nl(Stream),
    close(Stream).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

12.5. HUB

The content of the file “hub.pl” is presented on the following page and has been printed directly from the used IDE.

```
1   :-[actionPlans].
2
3   % This function is a switch that associate the desires (queries) to
     the respective intentions
4   checkQuery(Query, Robot):-
5           Query = 'wait_passengers',
6           wait_passengers(Robot),!
7           ;
8           Query = 'choose_road1',
9           choose_road1(Robot),!
10          ;
11          Query = 'choose_road2',
12          choose_road2(Robot),!
13          ;
14          Query = 'choose_road3',
15          choose_road3(Robot),!
16          ;
17          Query = 'drop_passengers',
18          drop_passengers(Robot),!
19          ;
20          Query = 'back_to_rank',
21          back_to_rank(Robot).
```