Autonomous Agents and Multi-Agent Systems, 11, 127–151, 2005
© 2005 Springer Science+Business Media, Inc. Manufactured in The Netherlands.

Argumentation and the Dynamics of Warranted Beliefs in Changing Environments

MARCELA CAPOBIANCO mc@cs.uns.edu.ar
Department of Computer Science and Engineering[**]*, Artificial Intelligence Research and Development Laboratory, Universidad Nacional del Sur, Av. Alem 1253, 8000 Bahía Blanca, Argentina*

CARLOS I. CHESÑEVAR cic@eps.udl.es
Department of Computer Science and Engineering[**]*, Artificial Intelligence Research and Development Laboratory, Universidad Nacional del Sur, Av. Alem 1253, 8000 Bahía Blanca, Argentina*
Departament of Computer Science, Artificial Intelligence Research Group, Universitat de Lleida, C/Jaume II, 69, E-25001 Lleida, Spain

GUILLERMO R. SIMARI grs@cs.uns.edu.ar
Department of Computer Science and Engineering[**]*, Artificial Intelligence Research and Development Laboratory, Universidad Nacional del Sur, Av. Alem 1253, 8000 Bahía Blanca, Argentina*

Abstract. One of the most difficult problems in Multi-Agent Systems (MAS) involves representing the knowledge and beliefs of an agent which performs its tasks in a dynamic environment. New perceptions modify this agent's current knowledge about the world, and consequently its beliefs about it also change. Such a revision and update process should be performed efficiently by the agent, particularly in the context of real-time constraints. In the last decade argumentation has evolved as a successful approach to formalize defeasible, commonsense reasoning, gaining wide acceptance in the MAS community by providing tools for designing and implementing features, which characterize reasoning capabilities in rational agents. In this paper we present a new argument-based formalism specifically designed for representing knowledge and beliefs of agents in dynamic environments, called *Observation-based Defeasible Logic Programming* (ODeLP). A simple but effective perception mechanism allows an ODeLP-based agent to model new incoming perceptions, and modify the agent's knowledge about the world accordingly. In addition, in order to improve the reactive capabilities of ODeLP-based agents, the process of computing beliefs in a changing environment is made computationally attractive by integrating a "dialectical database" with the agent's program, providing pre-compiled information about previous inferences. We present algorithms for managing dialectical databases as well as examples of their use in the context of real-world problems.

Keywords: argumentation, logic programming, defeasible logic programming, multi-agent systems.

1. Introduction: the problem of perceiving and changing beliefs

Knowledge representation issues play a major role in practically all areas of Artificial Intelligence, and the area of Multi-Agent Systems (MAS) is not an exception. Well-known problems in MAS involve the need for complex reasoning capabilities, planning and acting in dynamic environments [1]. In the last years, argumentation has gained wide acceptance in the MAS community by providing tools for designing and implementing different features which characterize interaction among rational agents.

Logic programming approaches to argumentation [2, 3] have proven to be suitable formalization tools in the context of MAS, as they combine the powerful features provided by logic programming for knowledge representation together with the ability to model complex, argument-based inference procedures in unified, integrated frameworks. Most MAS approaches based on logic programming rely on Extended Logic Programming (ELP) as the underlying formalism [4]. Thus, the agent's knowledge is codified in terms of an ELP program and the semantics of the program will represent the agent's beliefs. Although ELP is expressive enough to capture different kinds of negation (strict and default negation), it has limitations for modeling incomplete and potentially contradictory information. In a MAS context it is common for agents to require such capabilities, as they interact with the environment and amongst themselves, processing new inputs, changing dynamically their beliefs and intentions, etc. Clearly, in such a setting, the argumentation formalism underlying such MAS should have the capability for incorporating new information into the knowledge base of the agent and reasoning accordingly.

In this paper we present Observation-based Defeasible Logic Programming (ODeLP), an argument-based formalism for agents reasoning in dynamic environments. The fundamental notions of ODeLP come from Defeasible Logic Programming (DeLP) [5]. As in DeLP, the ODeLP formalism uses a knowledge representation language in the style of logic programming and the inference mechanism is based on argumentation. In order to provide an agent with the ability to incorporate changes in the world and integrate them into its existing beliefs, in ODeLP we have adapted the system to handle perceptions. This is a complex problem that could be considered as a form of Belief Revision or Update [6–8].

The problem of perception in artificial agents was addressed by Pollock from a philosophical standpoint based on studies about human perception. In *Taking Perception Seriously* [9], Pollock maintains that an agent residing in a complex dynamic environment cannot be provided from its creation with all the information it needs. It is then necessary to define some mechanism to gather information perceptually. Therefore, the problems faced by an agent designer are essentially similar to those faced by philosophers studying our knowledge about the external world.

In Pollock's work, perception is defined as "the process that begins with the stimulation of sensors, and ends with beliefs about the agent's immediate surroundings". The author then identifies three main problems in the implementation of a perception mechanism. First, perception may be misleading; in some cases the world may not be as it appears. Second, perception is a form of sampling; an agent cannot continuously monitor the entire state of the world. This sampling provides the agent with images of small parts of its environment at discrete moments in time or over-short intervals. It is up to the agent cognitive faculties to make inferences from these images to a coherent model of the world. Third, the world changes; the agent must update its picture of the world when faced with new perceptual inputs. In order to do this, it should be able to reason about both persistence and change using its knowledge of causal processes.

The analysis performed in Pollock's work is based on solid foundations from epistemology and provides an excellent starting point to address perception issues. In first place, perceptual inputs, called *percepts* are distinguished from beliefs. *Percepts* are the "non-doxastic states from which beliefs are obtained". The basic inference in perceptual reasoning is from a percept to a belief about the world. In this sense, a percept with a content P is considered as a defeasible reason for the agent to believe P. Next, Pollock defines how to obtain beliefs from percepts using a mechanism that is able to overcome the problems mentioned above. The result integrates perceptual reasoning with the inference engine. We have taken a similar posture in our proposal. In our approach, we simplify some of the problems described above by assuming that the agent considers that its perceptions are correct, and that any existing belief that contradicts a new perception should be abandoned and replaced by the new one.

Real time issues also play an important role when modeling agent interaction. In an argument-based MAS setting, a timely interaction is particularly hard to achieve, as the inference process involved is complex and computationally expensive. To improve the inference response, we will introduce specialized data structures for storing pre-compiled knowledge. These structures, called *Dialectical Databases*, can be used to speed up the inference process when answering future queries.

A dialectical database keeps a close resemblance to a Truth-Maintenance System (TMS). This resemblance is not in its design but in its functionality as a structure for supporting the change in beliefs in dynamic environments. TMS were defined by Jon Doyle [10] as supporting tools for problem solvers. The function of a TMS is to record and maintain the reasons an agent has for holding its beliefs. Doyle describes a series of procedures that determine the current set of beliefs and update it in accordance with new incoming reasons. Under this view, *rational thought* is deemed as the process of finding reasons for attitudes [10]. A given attitude (such as belief, desire, etc.) is rational if it is supported by some acceptable explanation.

The TMS solve part of the belief revision problem in general problem solvers and provide a mechanism for making non-monotonic assumptions. As Doyle [10] mentions, performance is also significantly improved. Even though the overhead required to record justifications for every program belief might seem excessive, we must consider the expense of not keeping these records. When information about derivations is discarded, the same information must be continually re-derived, even when only irrelevant assumptions have changed. Similar observations could be made for dialectical databases.

Every node in the TMS has an associated set of justifications. Each justification represents a different reason for asserting it. The node is believed if and only if at least one of its justifications is *valid*.[1] In this case it is said to be *in* the set of beliefs; otherwise, the node is *out* of this set. It is important to remark that the distinction between *in* and *out* is not the same as that between *true* and *false*. The former classification refers to current possession of a valid reason for belief; while *true* and *false* evaluate inferences according to its truth value, independently of any reason.

In the TMS, each potential belief to be used as a hypothesis or a conclusion of an argument must be given its own distinct node. When uncertainty about some inference P exists, nodes for both P and its negation $\sim P$ must be provided. Either of these nodes can either have or lack well-founded arguments, leading to a four-element belief set (neither P nor $\sim P$ are believed, exactly one is believed, or both are believed). The author details the procedures needed to establish the state of every node, and to update these states in case new justifications or facts are added to the TMS. Since the appearance of TMS a large body of literature and applications have been developed [11–16]. It does not seem as though the original idea was any technical mechanism in particular, but the general concept of an independent module for belief maintenance instead [14].

Informally, dialectical databases constitute a repository of structures called *dialectical trees*, which represent a potential dialectical confrontation of arguments for and against some belief. The knowledge of an agent is represented as an ODeLP program. The complete set of potential dialectical trees corresponding to a given program is built in advance and stored in a dialectical database. As the situation changes, different dialectical trees become able to support their conclusion and in that manner the beliefs of the agent change. In the rest of this work we will formally introduce the elements necessary for the construction of a dialectical database.

The remainder of this paper is organized as follows. Section 2 summarizes the main features of the ODeLP formalism. Section 3 introduces the notion of dialectical databases, discussing its role as a tool to speed up inference in ODeLP. Section 4 presents a worked example. Finally, Section 5 summarizes the main conclusions that have been obtained.

2. ODeLP: observation-based DeLP

Defeasible argumentation [3, 17, 35] has evolved as a successful approach to formalize defeasible, commonsense reasoning. In the last few years particular attention has been given to argument-based extensions of *logic programming*, which has turned out to be a suitable paradigm, as it provides a natural vehicle for modeling argumentative inference. Argument-based applications have been developed in many areas, such as agent theory, knowledge engineering, and legal reasoning, among others [18–21].

DeLP [5] is a defeasible argumentation formalism based on logic programming that uses *defeasible argumentation* to decide between contradictory conclusions through a *dialectical analysis*. Codifying the knowledge base of the agent by means of a DeLP program provides a good trade-off between expressivity and implementability. Recently, extensions of DeLP that integrate possibilistic logic and vague knowledge along with an argument-based framework have also been proposed [22, 23].

In such applications, DeLP is intended to model the behavior of a single intelligent agent in a *static* scenario. DeLP lacks the appropriate mechanisms to represent knowledge in dynamic environments, where agents must be able to perceive the changes in the world and integrate them into their existing beliefs [9]. The ODeLP framework aims at solving this limitation by modeling perception as new facts to be added to the agent's knowledge base. Since adding such new facts may result in inconsistencies, an associated updating process is used to solve them. The definitions that follow summarize the main features of ODeLP.

2.1. *Language*

The language of ODeLP is based on the language of extended logic programming. To characterize the different elements associated with the ODeLP language we will introduce the notion of signature, as usual in logic programming.

Definition 1 (Signature). *A signature Σ is a tuple $\langle \mathscr{V}, \text{Pred}, \text{Func} \rangle$, where $\mathscr{V}$ is a countable set of variables, Pred is a finite set of predicates, and Func is a finite set of functions, such that $\mathscr{V} \cap (\text{Pred} \cup \text{Func}) = \emptyset$.*

As in Prolog standard notation, variables are denoted with identifiers starting with uppercase letters while functions and predicates start with lowercase letters. Every signature has an associated arity function that assigns a natural number to each function and predicate. As usual, constants are functions with arity 0 and propositions are predicates with a arity 0. The set of symbols that can be used in ODeLP programs is defined as follows:

Definition 2 (Alphabet). *The alphabet generated from a given signature Σ is composed by the members of Σ, the symbol "$\sim$" denoting strong negation [4] and the symbols "(", ")", "." and ",".*

Terms represent objects in the environment that belong to the agent's epistemic model. As usual, the notions of atom and literal are defined on the basis of the notion of term.

Definition 3 (Term). *Let $\Sigma = \langle \mathscr{V}, \text{Pred}, \text{Func} \rangle$ be a signature. A term of Σ is inductively defined as follows:*

1. *every variable $V \in \mathscr{V}$ is a term,*
2. *every constant $c \in \text{Func}$ is a term,*
3. *if $f \in \text{Func}, \text{arity}\ (f) = n$ and $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is also a term.*

Definition 4 (Atom). *Let $\Sigma = \langle \mathscr{V}, \text{Pred}, \text{Func} \rangle$ be a signature, $t_1, \ldots, t_n$ terms of Σ and $p \in \text{Pred}$ such that $\text{arity}(p) = n$ then $p(t_1, \ldots, t_n)$ is an atom of Σ.*

Definition 5 (Literal). *Let Σ be a signature, then every atom A of Σ is a positive literal, while every negated atom $\sim A$ is a negative literal. A literal of Σ is a positive literal or a negative literal.*

Definition 6 (Complement of a literal) . *Let L be a literal and A an atom. The complement of L, noted as $\overline{L}$, is defined as follows:*

$$\overline{L} = \begin{cases} A & \text{if}\, L = \sim A \\ \sim A & \text{if}\, L = A \end{cases}$$

Programs in ODeLP are formed by *observations* and *defeasible rules*. Observations correspond to facts in the context of logic programming, and represent the knowledge an agent has about the world. *Defeasible rules* provide a way of performing tentative reasoning as in other argumentation formalisms [19, 24].

Definition 7 (Observation) (Defeasible Rule). *An observation is a ground literal L representing some fact about the world (obtained through the agent's perception mechanism) that the agent believes to be correct. A defeasible rule has the form* $L_0 \prec L_1, L_2, \ldots, L_k$, *where* L_0 *is a literal and* $L_1, L_2, \ldots, L_k$ *is a non-empty finite set of literals.*

Intuitively a defeasible rule $L_0 \prec L_1, L_2, \ldots, L_k$ can be read as "$L_1, L_2, \ldots, L_k$ provide a tentative reasons to believe in L_0" [24].

Definition 8 (ODeLP Program). *An ODeLP program* $\mathcal{P}$ *is a pair* $\langle \Psi, \Delta \rangle$, *where* Ψ *is a finite set of observations and* Δ *is a finite set of defeasible rules. In a program* $\mathcal{P}$, *the set* Ψ *must be non-contradictory (i.e., it is not the case that* $Q \in \Psi$ *and* $\overline{Q} \in \Psi$, *for any literal* Q*).*

Example 1. Figure 1 shows an ODeLP program for performing basic email filtering. In this program observations stand for different characteristics of email messages. Thus, `virus(X)` stands for "message `X` has a virus"; `local(X)` indicates that "message `X` is from the local host"; `filters(X)` specifies that "message `X` should be filtered" redirecting it to a particular folder; `black_list(X)` indicates that "message `X` is considered dangerous" because of the server it is coming from; and `contacts(X)` indicates that "the sender of message `X` is in the contact list of the user".

The first rule expresses that if the email does not match with any user-defined filter then it usually should be moved to the "inbox" folder. The second rule indicates that unfiltered messages in the "junk" folder usually should not be moved to the inbox. According to the third rule, messages to be filtered should not be moved to the inbox. The following two rules establish that a message should be moved to the "junk" folder if it is marked as spam or it contains viruses. Finally

```
Ψ
virus(b) local(b)
local(d)
~filters(b)
~filters(c)
~filters(d)
black_list(c)
black_list(d)
contacts(d)
Δ
move_inbox(X) -< ~filters(X)
~move_inbox(X) -< ~filters(X), move_junk(X)
~move_inbox(X) -< filters(X)
move_junk(X) -< spam(X)
move_junk(X) -< virus(X)
spam(X) -< black_list(X)
~spam(X) -< black_list(X), contacts(X)
~spam(X) -< local(X)
```

Figure 1. An ODeLP program $\mathcal{P}_{\text{mail}}$ for email filtering.

there are three rules for spam classification: a message is usually labeled as spam if it comes from a server that is in the blacklist. Nevertheless, even if an email comes from a server in the blacklist it is not labeled as spam when the sender is in the contact list of the user. Besides, a message from the local host is usually not classified as spam.

2.2. *Inference mechanism*

Given an ODeLP program $\mathcal{P}$, a query posed to $\mathcal{P}$ corresponds to a ground literal Q which must be supported by an *argument* [5, 24]. Arguments are built on the basis of a *defeasible derivation* computed by backward chaining applying the usual SLD inference procedure used in logic programming [25]. Observations play the role of facts and defeasible rules can be seen as inference rules. In addition to provide a proof supporting a ground literal, such a proof must be non-contradictory and minimal for being considered an argument in ODeLP. Formally:

Definition 9 (Defeasible Derivation). *Let $\mathcal{P} = \langle \Psi, \Delta \rangle$ be an ODeLP program and let Q be a ground literal. A finite sequence of ground literals,*

$$s = Q_1, Q_2, \ldots, Q_{n-1}, Q$$

is said to be a defeasible derivation for Q from $\mathcal{P}$ (abbreviated $\mathcal{P} \mid\!\sim Q$) if for every $Q_i, 1 \leq i \leq n$

1. *the literal Q_i belongs to Ψ, or*
2. *there exists a defeasible rule $r \in \Delta$ and a ground instance t of r, $t = Q_i \prec L_1, \ldots, L_m$, where $L_1, \ldots, L_m$ are ground literals previously occurring in the sequence s.*

Definition 10 (Argument – Sub-argument). *Given an ODeLP program $\mathcal{P}$, an argument $\mathcal{A}$ for a ground literal Q, also denoted $\langle \mathcal{A}, Q \rangle$, is a subset of ground instances of the defeasible rules in $\mathcal{P}$ such that:*

1. *there exists a defeasible derivation for Q from $\Psi \cup \mathcal{A}$,*
2. *$\Psi \cup \mathcal{A}$ is non-contradictory,*
3. *There is no $\mathcal{A}' \subset \mathcal{A}$ such that $\Psi \cup \mathcal{A}' \mid\!\sim Q$.*

Given two arguments $\langle \mathcal{A}_1, Q_1 \rangle$ and $\langle \mathcal{A}_2, Q_2 \rangle$, we will say that $\langle \mathcal{A}_1, Q_1 \rangle$ is a sub-argument of $\langle \mathcal{A}_2, Q_2 \rangle$ iff $\mathcal{A}_1 \subseteq \mathcal{A}_2$.

As in most argumentation frameworks, arguments in ODeLP can attack each other. This situation is captured by the notion of *counterargument*. Defeat among arguments is defined combining the counterargument relation and a preference criterion "$\preceq$". Specificity [24, 26, 27] is the syntactic preference criterion used by default in ODeLP, although other alternative criteria could be used. Specificity favors those arguments which are *more direct* (i.e., less use of defeasible rules) or *more informed* (i.e., contain more specific information) [24].

Definition 11 (Counter-argument). *An argument $\langle \mathcal{A}_1, Q_1 \rangle$ counter-argues an argument $\langle \mathcal{A}_2, Q_2 \rangle$ at a literal Q if and only if there is a sub-argument $\langle \mathcal{A}, Q \rangle$ of $\langle \mathcal{A}_2, Q_2 \rangle$ such that Q_1 and Q are complementary literals.*

Definition 12 (Defeater). *An argument $\langle \mathcal{A}_1, Q_1 \rangle$ defeats $\langle \mathcal{A}_2, Q_2 \rangle$ at a literal Q if and only if there exists a sub-argument $\langle \mathcal{A}, Q \rangle$ of $\langle \mathcal{A}_2, Q_2 \rangle$ such that $\langle \mathcal{A}_1, Q_1 \rangle$ counter-argues $\langle \mathcal{A}_2, Q_2 \rangle$ at Q, and either:*

1. *$\langle \mathcal{A}_1, Q_1 \rangle$ is strictly preferred over $\langle \mathcal{A}, Q \rangle$ according to the preference criterion "$\preceq$" (then $\langle \mathcal{A}_1, Q_1 \rangle$ is a proper defeater of $\langle \mathcal{A}_2, Q_2 \rangle$), or*
2. *$\langle \mathcal{A}_1, Q_1 \rangle$ is unrelated to $\langle \mathcal{A}, Q \rangle$ by "$\preceq$" (then $\langle \mathcal{A}_1, Q_1 \rangle$ is a blocking defeater of $\langle \mathcal{A}_2, Q_2 \rangle$).*

Defeaters are arguments and may in turn be defeated. Thus, a complete dialectical analysis is required to determine which arguments are ultimately accepted. Naturally, such analysis results in a tree structure called *dialectical tree*, in which nodes are labeled as undefeated (`U-nodes`) or defeated (`D-nodes`) according to a marking procedure [5, 28]. Formally:

Definition 13 (Dialectical Tree). *The dialectical tree for an argument $\langle \mathcal{A}, Q \rangle$, denoted $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$, is recursively defined as follows:*

1. *A single node labeled with an argument $\langle \mathcal{A}, Q \rangle$ with no defeaters (proper or blocking) is by itself the dialectical tree for $\langle \mathcal{A}, Q \rangle$.*
2. *Let $\langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle$ be all the defeaters (proper or blocking) for $\langle \mathcal{A}, Q \rangle$. The dialectical tree for $\langle \mathcal{A}, Q \rangle, \mathcal{T}_{\langle \mathcal{A}, Q \rangle}$, is obtained by labeling the root node with $\langle \mathcal{A}, Q \rangle$, and making this node the parent of the root nodes for the dialectical trees of $\langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle$.*

Definition 14 (Marking of the Dialectical Tree). *Let $\langle \mathcal{A}_1, Q_1 \rangle$ be an argument and $\mathcal{T}_{\langle \mathcal{A}_1, Q_1 \rangle}$ its dialectical tree, then:*

1. *All the leaves in $\mathcal{T}_{\langle \mathcal{A}_1, Q_1 \rangle}$ are marked as a* `U-node`.
2. *Let $\langle \mathcal{A}_2, Q_2 \rangle$ be an inner node of $\mathcal{T}_{\langle \mathcal{A}_1, Q_1 \rangle}$. Then $\langle \mathcal{A}_2, Q_2 \rangle$ is marked as* `U-node` *iff every child of $\langle \mathcal{A}_2, Q_2 \rangle$ is marked as a* `D-node`. *The node $\langle \mathcal{A}_2, Q_2 \rangle$ is marked as a* `D-node` *if and only if it has at least a child marked as* `U-node`.

Dialectical analysis may in some situations give rise to *fallacious argumentation* [28]. In ODeLP dialectical trees are ensured to be free of fallacies [27] by applying additional constraints when building *argumentation lines* (the different possible paths in a dialectical tree). The notions that follow have been developed to address these issues.

Definition 15 (Argumentation line) (based on [29]). *Let $\mathscr{P} = \langle \Psi, \Delta \rangle$ be a ODeLP program and let $\langle \mathscr{A}, Q \rangle$ be an argument wrt $\mathscr{P}$. An argumentation line starting from $\langle \mathscr{A}, Q \rangle$, denoted $\lambda^{\langle \mathscr{A}, Q \rangle}$ (or simply λ), is a possibly infinite sequence of arguments*

$$\lambda^{\langle \mathscr{A}, Q \rangle} = [\langle \mathscr{A}_0, Q_0 \rangle, \langle \mathscr{A}_1, Q_1 \rangle, \ldots, \langle \mathscr{A}_n, Q_n \rangle, \ldots]$$

satisfying the following conditions:

1. *If $\langle \mathscr{A}, Q \rangle$ has no defeaters, then $\lambda^{\langle \mathscr{A}, Q \rangle} = [\langle \mathscr{A}, Q \rangle]$.*
2. *If $\langle \mathscr{A}, Q \rangle$ has a defeater $\langle \mathscr{B}, P \rangle$ in $\mathscr{P}$, then $\lambda^{\langle \mathscr{A}, Q \rangle} = \langle \mathscr{A}, Q \rangle \circ \lambda^{\langle \mathscr{B}, P \rangle}$.*

where the '$\circ$' operator stands for adding $\langle \mathscr{A}, Q \rangle$ as the first element of $\lambda^{\langle \mathscr{B}, P \rangle}$.

In each argumentation line $\lambda^{\langle \mathscr{A}, Q \rangle} = [\langle \mathscr{A}_0, Q_0 \rangle, \langle \mathscr{A}_1, Q_1 \rangle, \ldots, \langle \mathscr{A}_n, Q_n \rangle, \ldots]$ the argument $\langle \mathscr{A}_0, Q_0 \rangle$ is supporting the main query Q_0, and every argument $\langle \mathscr{A}_i, Q_i \rangle$ defeats its predecessor $\langle \mathscr{A}_{i-1}, Q_{i-1} \rangle$. Thus, for $k \geq 0, \langle \mathscr{A}_{2k}, Q_{2k} \rangle$ is a supporting argument for Q_0 and $\langle \mathscr{A}_{2k+1}, Q_{2k+1} \rangle$ is an interfering argument for Q_0. In other words, every argument in the line supports Q_0 or interferes with it. As a result, an argumentation line can be split in two disjoint sets: λ_S of supporting arguments, and λ_I of interfering arguments.

On the basis of the above notions, fallacies that could appear in argumentation lines in ODeLP programs can be classified as follows:

1. An argument $\mathscr{A}_1$ could be introduced in an argumentation line *both as an interfering and supporting argument*, producing a *contradictory* argumentation line e.g., $\lambda_1 = [\mathscr{A}_1, \mathscr{A}_2, \mathscr{A}_3, \mathscr{A}_1, \ldots]$.
2. An argument $\mathscr{A}_1$ could be reintroduced as a supporting argument *for itself*. In that case a *circular* argumentation line would result, e.g., $\lambda_2 = [\mathscr{A}_1, \mathscr{A}_2, \mathscr{A}_3, \mathscr{A}_4, \mathscr{A}_1, \ldots]$.

Argumentation lines as λ_1 and λ_2 should not be considered as acceptable, as they represent flawed reasoning processes. These fallacious situations can be generalized to cycles of any length: even cycles evidence contradictory argumentation, whereas odd cycles indicate circular argumentation. To solve these problems we introduce the following concepts.

Definition 16 Contradictory set of arguments. *A set of arguments $S = \bigcup_{i=1}^{n} \{\langle \mathscr{A}_i, Q_i \rangle\}$ is contradictory with respect to a program $\mathscr{P} = \langle \Psi, \Delta \rangle$ if and only if the set $\Psi \cup \bigcup_{i=1}^{n} \mathscr{A}_i$ allows the derivation of complementary literals.*

After the discussion above, we can introduce the notion of *acceptable argumentation line*.

Definition 17 (Acceptable argumentation line) (based on [29]). *Let $\mathscr{P} = \langle \Psi, \Delta \rangle$ be a program, and let*

$$\lambda = [\langle \mathscr{A}_0, q_0 \rangle, \langle \mathscr{A}_1, Q_1 \rangle, \ldots, \langle \mathscr{A}_n, Q_n \rangle, \ldots]$$

be an argumentation line in $\mathcal{P}$, such that

$$\lambda' = [\langle \mathcal{A}_0, Q_0 \rangle, \langle \mathcal{A}_1, Q_1 \rangle, \ldots, \langle \mathcal{A}_k, Q_k \rangle]$$

is an initial segment of λ. The sequence λ' is an acceptable argumentation line in $\mathcal{P}$ if and only if it is the longest initial segment in λ satisfying the following conditions:

1. *The sets λ'_S and λ'_I are each non-contradictory sets of arguments with respect to $\mathcal{P}$,*
2. *No argument $\langle \mathcal{A}_j, Q_j \rangle$ in λ' is a sub-argument of an argument $\langle \mathcal{A}_i, Q_i \rangle$ of $\lambda', i < j$,*
3. *In λ' there is no subsequence of arguments*

$$[\langle \mathcal{A}_{i-1}, Q_{i-1} \rangle, \langle \mathcal{A}_i, Q_i \rangle, \langle \mathcal{A}_{i+1}, Q_{i+1} \rangle]$$

such that $\langle \mathcal{A}_i, Q_i \rangle$ is a blocking defeater for $\langle \mathcal{A}_{i-1}, Q_{i-1} \rangle$, and $\langle \mathcal{A}_{i+1}, Q_{i+1} \rangle$ is a blocking defeater for $\langle \mathcal{A}_i, Q_i \rangle$.

Let us analyze the rationale for the conditions in Definition 17. Condition 1 prohibits the use of contradictory information on either side (proponent or opponent). Condition 2 eliminates circular reasoning. Finally, Condition 3 enforces the use of a stronger argument to defeat an argument which acts as a blocking defeater. The reason for this policy is justified by the following considerations. Suppose that argumentation lines containing two consecutive blocking defeaters were allowed, and consider the following scenario. An argument $\langle \mathcal{A}, L \rangle$ is blocked by $\langle \mathcal{B}, \sim L \rangle$ which in turn is blocked by $\langle \mathcal{C}, L \rangle$. If there are no more arguments to take into account, $\langle \mathcal{A}, L \rangle$ would be warranted. Nevertheless, the support for L is not better than the support for $\sim L$.

An *acceptable dialectical tree* is defined in turn as a tree where every argumentation line is *acceptable* and the notion of warrant in ODeLP is grounded on this concept. Given a query Q and an ODeLP program $\mathcal{P}$, we will say that Q is *warranted* wrt $\mathcal{P}$ iff there exists an argument $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ such that the root of its associated dialectical tree $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ is marked as a U-node.

Definition 18 (Warrant). *Let $\mathcal{A}$ be an argument for a literal Q, and let $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ be its associated dialectical tree. Argument $\mathcal{A}$ is a warrant* for *Q if and only if the root of $\mathcal{T}_{\langle \mathcal{A}, Q \rangle}$ is marked as a* `U-node`.

Solving a query Q in ODeLP accounts for trying to find a warrant for Q, as shown in the following example.

Example 2. Consider the program $\mathcal{P}_{mail}$ shown in Example 1. Let `move_inbox(d)` be a query wrt $\mathcal{P}_{mail}$. The search for a warrant for `move_inbox(d)` will result in an argument $\langle \mathcal{A}, \texttt{move\_inbox(d)} \rangle$, with

$$\mathcal{A} = \{\texttt{move\_inbox(d)} \prec \sim \texttt{filters(d)}\}$$

allowing to conclude that message d should be moved to the folder Inbox, as it has no associated filter. However, there exists a defeater for $\langle \mathcal{A}, \texttt{move\_inbox(d)} \rangle$, namely $\langle \mathcal{B}, \sim \texttt{move\_inbox(d)} \rangle$, as there are reasons to believe that message d is spam:[2]

$$\mathscr{B} = \{ \sim \texttt{move\_inbox(d)} \prec \sim \texttt{filters(d)}, \texttt{move\_junk(d)}; \\ \texttt{move\_junk(d)} \prec \texttt{spam(d)}, \texttt{spam(d)} \prec \texttt{black\_list(d)}\}$$

Using specificity as the preference criterion, $\langle \mathscr{B}, \sim \texttt{move\_inbox(d)} \rangle$ is a proper defeater for $\langle \mathscr{A}, \texttt{move\_inbox(d)} \rangle$. However, two other defeaters can be found for $\langle \mathscr{B}, \sim \texttt{move\_inbox(d)} \rangle$, since message *d* comes from the local host, and the sender is in the user's contacts list:

– $\langle \mathscr{C}, \sim \texttt{spam(d)} \rangle$, *where* $\mathscr{C} = \{\sim \texttt{spam(d)} \prec \texttt{black\_list(d)}, \texttt{contacts(d)}\}$.
– $\langle \mathscr{D}, \sim \texttt{spam(d)} \rangle$, *where* $\mathscr{D} = \{\sim \texttt{spam(d)} \prec \texttt{local(d)}\}$.

There are no more arguments to consider, and the resulting dialectical tree is shown in Figure 2. The marking procedure determines that the root node $\langle \mathscr{A}, \texttt{move\_inbox(d)} \rangle$ is an `U-node` and hence the original query is warranted.

2.3. *Modeling beliefs and perceptions in ODeLP*

ODeLP models the beliefs of an agent in a simple way: given a program $\mathscr{P}$ representing an agent's knowledge, a literal Q is believed by the agent iff Q is warranted. In particular, different doxastic attitudes are distinguished wrt a given literal Q:

– believe that Q is *true* whenever Q is warranted;
– believe that Q is *false* (i.e., believe $\overline{Q}$) whenever $\overline{Q}$ is warranted; and
– believe that Q is *undecided* whenever none of the above cases apply.

Consistency is a basic property for agent's beliefs, in the sense that it is not possible to believe simultaneously in a literal Q and its complement $\overline{Q}$ [30]. It can be proven that agents using ODeLP naturally satisfy this requirement.[3]

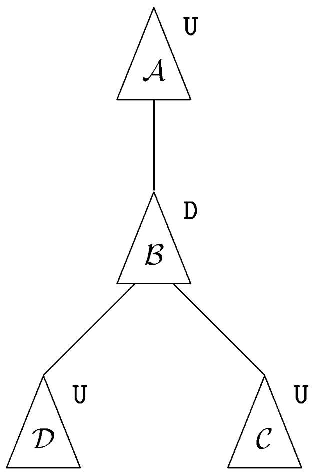

Figure 2. Dialectical tree from Example 2.

In ODeLP the mechanism for updating the knowledge base of an agent is simple but effective. We assume that perception is carried out by devices that detect changes in the world and report them as new facts (literals). The actual devices used will depend on the particular application domain, and their characterization is outside the scope of this paper. We also make the assumption that the perception mechanism is flawless, and new perceptions always supersede old ones. Any perception will be reported as a new fact Q to be added to the set of observations Ψ. If this new perception Q is contradictory with Ψ, then necessarily $\overline{Q} \in \Psi$. In such a case, we use a simple update function [7] that computes a new observation set Ψ' as $\Psi \backslash \{\overline{Q}\} \cup \{Q\}$. Thus new perceptions are always preferred: with a flawless perception mechanism, the source of the conflict must be a change in the state of world.

Consistency of the set of beliefs is a fundamental property of ODeLP. It can be shown that the set of warranted arguments obtained from a given program does not contain a pair of conflicting arguments. To show this property we will first demonstrate some auxiliary results.

Proposition 1. *Let $\mathcal{P}$ be a program and $\langle \mathcal{A}, Q\rangle, \langle \mathcal{B}, R\rangle$ two arguments built from $\mathcal{P}$ such that $\langle \mathcal{A}, Q\rangle$ is a defeater of $\langle \mathcal{B}, R\rangle$ and $\langle \mathcal{A}, Q\rangle$ is warranted wrt $\mathcal{P}$. Let $\langle \mathcal{A}_s, Q_s\rangle$ be a supporting argument of $\mathcal{T}_{\langle \mathcal{A}, Q\rangle}$ such that $\langle \mathcal{A}_s, Q_s\rangle$ is a* `U node`. *If $\langle \mathcal{A}_s, Q_s\rangle$ appears in $\mathcal{T}_{\langle \mathcal{B}, R\rangle}$ it must be also labeled as* `U node` *in $\mathcal{T}_{\langle \mathcal{B}, R\rangle}$.*

Proof. To show this property we use structural induction on the subtree rooted in $\langle \mathcal{A}_s, Q_s\rangle$.

Base case. Let us consider the fact that $\langle \mathcal{A}_s, Q_s\rangle$ does not have defeaters in $\mathcal{T}_{\langle \mathcal{A}_1, Q_1\rangle}$. If $\mathcal{A}_s$ belongs to $\mathcal{T}_{\langle \mathcal{A}_2, Q_2\rangle}$ then it must be marked as an `U node`, given that no new defeaters can be built.

Inductive step. Let us assume that $\langle \mathcal{A}_s, Q_s\rangle$ has k defeaters in $\mathcal{T}_{\langle \mathcal{A}, Q\rangle}$, noted as $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$, and $\langle \mathcal{A}_s, Q_s\rangle$ is present in $\mathcal{T}_{\langle \mathcal{B}, R\rangle}$. Since $\mathcal{A}_s$ is a `U node` in $\mathcal{T}_{\langle \mathcal{A}, Q\rangle}$ every defeater of $\mathcal{B}_i$ must be a `D node` in $\mathcal{T}_{\langle \mathcal{A}, Q\rangle}$.

Next we will show that if $\mathcal{B}_i$ is present in $\mathcal{T}_{\langle \mathcal{B}, P\rangle}$ then it must be labeled as `D node` in this tree. Given that $\mathcal{B}_i$ is a `D node` in $\mathcal{T}_{\langle \mathcal{A}, Q\rangle}$ there must exist a non-defeated supporting argument $\mathcal{C}$ in the subtree rooted in $\mathcal{B}_i$.

If we use the inductive hypothesis on this subtree it follows that if $\mathcal{C}$ is present in $\mathcal{T}_{\langle \mathcal{B}, R\rangle}$ it must be labeled as a *U node* and then $\mathcal{B}_i$ must be a `D node` in $\mathcal{T}_{\langle \mathcal{B}, R\rangle}$. It remains to check, what happens when $\mathcal{C}$ cannot be introduced as a defeater of $\mathcal{B}_i$ in $\mathcal{T}_{\langle \mathcal{B}, R\rangle}$. This may follow from one of these reasons:

1. $\mathcal{C}$ is contradictory with an interfering argument in the argumentation line in which $\mathcal{C}$ should be added.
2. $\mathcal{C}$ is a sub-argument of a previous argument in this argumentation line.

The first scenario is not possible, given that in this case $\mathcal{C}$ could not appear in $\mathcal{T}_{\langle \mathcal{A}, Q\rangle}$ since it would be contradictory with a supporting argument in the argumentation line. In the second case, $\mathscr{C}$ must be a sub-argument of $\mathscr{B}$ (otherwise it could not belong to $\mathscr{T}_{\langle\mathscr{A},Q\rangle}$). Nevertheless, since $\mathscr{C}$ is a defeater of $\mathscr{B}_i$, the set $\mathscr{C} \cup \mathscr{B}_i \cup \Psi$ allows the derivation of complementary literals. Then $\mathscr{B} \cup \mathscr{B}_i \cup \Psi$ also allows the derivation of complementary literals since $\mathscr{C} \subseteq \mathscr{B}$. In this situation it is not possible that $\mathscr{B}_i$ be in $\mathscr{T}_{\langle\mathscr{B},R\rangle}$, given that it contradicts a supporting argument in its argumentative line. Thus, if $\mathscr{B}_i$ is in $\mathscr{T}_{\langle\mathscr{B},R\rangle}$ it must be labeled as a `U node` in this tree. □

The following statement derives from the previous proposition.

Corollary 1. *Let $\mathscr{P}$ be a program and $\langle\mathscr{A},Q\rangle$, $\langle\mathscr{B},R\rangle$ two arguments built from $\mathscr{P}$. If $\langle\mathscr{A},Q\rangle$ is a defeater of $\langle\mathscr{B},R\rangle$ and $\langle\mathscr{A},Q\rangle$ is a warranted argument wrt $\mathscr{P}$ then $\langle\mathscr{A},Q\rangle$ must be labeled as* `U node` *in the dialectical tree rooted in $\langle\mathscr{B},R\rangle$.*

Proof. By hypothesis, argument $\langle\mathscr{A},Q\rangle$ is a supporting undefeated argument in $\mathscr{T}_{\langle\mathscr{A},Q\rangle}$ that is also present in $\mathscr{T}_{\langle\mathscr{B},R\rangle}$. From proposition 1 we can infer that $\langle\mathscr{A},Q\rangle$ must be a `U node` in the dialectical tree for $\langle\mathscr{B},R\rangle$. □

Finally, the following lemma proves that the set of beliefs in an ODeLP program is consistent.

Lemma 1. *Let $\mathscr{P} = \langle\Psi,\Delta\rangle$ be an ODeLP program, and let Warr($\mathscr{P}$) be the set of warranted arguments in $\mathscr{P}$. For any pair of arguments $\langle\mathscr{A},Q\rangle$, $\langle\mathscr{B},R\rangle$, such that $\langle\mathscr{A},Q\rangle \in$ Warr($\mathscr{P}$) and $\langle\mathscr{B},R\rangle \in$* Warr($\mathscr{P}$) *it holds that $\langle\mathscr{A},Q\rangle$ is not a counterargument for $\langle\mathscr{B},R\rangle$.*

Proof. Suppose by contradiction that there exists a pair of arguments $\langle\mathscr{A},Q\rangle$, $\langle\mathscr{B},R\rangle$ such that $\langle\mathscr{A},Q\rangle$ counter-argues $\langle\mathscr{B},R\rangle$ and both belong to Warr($\mathscr{P}$). We can assume without any loss of generality that $\langle\mathscr{A},Q\rangle$ defeats $\langle\mathscr{B},R\rangle$ and $\langle\mathscr{B},R\rangle$ also counter-argues $\langle\mathscr{A},Q\rangle$, that is to say that the relation of counterargument is reciprocal (otherwise there would exist an argument $\langle\mathscr{B}',R'\rangle$ sub-argument of $\langle\mathscr{B},R\rangle$, such that $\langle\mathscr{B}',\mathscr{R}'\rangle$ counter-argues $\langle\mathscr{A},Q\rangle$ and the relation of counter-argumentation among $\langle\mathscr{B}',\mathscr{R}'\rangle$ and $\langle\mathscr{A},Q\rangle$ is reciprocal).

Since $\langle\mathscr{A},Q\rangle$ and $\langle\mathscr{B},R\rangle$ are warranted we can assume that there exists a dialectical tree with root in $\langle\mathscr{A},Q\rangle$, noted $\mathscr{T}_{\langle\mathscr{A},Q\rangle}$, that warrants Q and a dialectical tree $\mathscr{T}_{\langle\mathscr{B},R\rangle}$ that warrants $\langle\mathscr{B},R\rangle$. Argument $\langle\mathscr{A},Q\rangle$ defeats $\langle\mathscr{B},R\rangle$ and thus $\langle\mathscr{A},Q\rangle$ must be marked as a `D node` in the dialectical tree $\mathscr{T}_{\langle\mathscr{B},R\rangle}$, since otherwise $\langle\mathscr{B},R\rangle$ could not be a warranted argument. However, applying corollary 1 argument $\langle\mathscr{A},Q\rangle$ must be labeled as `U node` in the dialectical tree for $\langle\mathscr{B},R\rangle$ (since $\langle\mathscr{A},Q\rangle$ is warranted and $\langle\mathscr{B},R\rangle$ defeats $\langle\mathscr{A},Q\rangle$). This contradiction arises from assuming the existence of a pair of arguments $\langle\mathscr{A},Q\rangle$, $\langle\mathscr{B},R\rangle$ under the conditions previously stated. □

3. Pre-compiled argumentation: dialectical databases

Based on previous work in TMS and argumentation [31, 32], our goal here is to integrate pre-compiled argumentation into an agent framework based on ODeLP in order to address real-time constraints in a MAS setting. To do so, we want an

ODeLP-based agent to be able to answer queries efficiently, by avoiding re-computing arguments which were already computed before.

There are different options for this task starting from ODeLP program $\mathscr{P}$. A simple approach would be to record every argument that has been computed so far. However, a large number of arguments can be obtained from a relatively small program, thus resulting in a large database. On the other hand, many arguments are obtained using *different* instances of the *same* defeasible rules. Recording every generated argument could result in storing many arguments which are structurally identical, only differing on the constants being used to build the corresponding derivations.

Another important problem arises with perceptions. Note that the set of arguments that can be built from a program $\mathscr{P} = \langle \Psi, \Delta \rangle$ also depends on the observation set Ψ. When Ψ is updated with new perceptions, arguments which were previously derivable from $\mathscr{P}$ may no longer be so. If pre-compiled knowledge depends on Ψ, it should be updated as new perceptions appear. Such an alternative is not suitable, as new perceptions are frequent in dynamic environments. As a consequence, pre-compiled knowledge should be managed independently from the set of observations Ψ.

Based on the previous analysis we will define a database structure called *dialectical database*, which will keep a record of all possible *potential arguments* in an ODeLP program $\mathscr{P}$ as well as their defeat relations among them. Potential arguments are formed by non-grounded defeasible rules, depending thus only on the set of rules Δ in $\mathscr{P}$. As we will discuss later, attack relations among potential arguments can be also captured. Potential arguments and the defeat relations among them will be stored in the dialectical database. This will be analyzed Section 3.1.

3.1. Computing potential arguments in ODeLP

Definition 19 (Instance for a set of defeasible rules). *Let* A *be a set of defeasible rules, and consider any set* $\mathscr{A}$ *formed by ground instances of the defeasible rules in* A. *The set* $\mathscr{A}$ *is said to be an instance of* A *iff every rule in* $\mathscr{A}$ *results from an instance of a defeasible rule in* A.

Example 3. If $\mathsf{A} = \{\mathtt{s(X)} \prec \sim \mathtt{r(X)}; \sim \mathtt{r(X)} \prec \mathtt{p(X)}$ then $\mathscr{A} = \mathtt{s(t)} \prec \sim \mathtt{r(t)} \sim \mathtt{r(a)};$ $\prec \mathtt{p(a)}\}$ is an instance of A

Definition 20 (Potential argument). *Let* Δ *be a set of defeasible rules. A subset* A *of* Δ *is a potential argument for a literal* Q, *noted as* $\langle\langle \mathsf{A}, Q \rangle\rangle$ *if there exist a non-contradictory set of literals* Φ *and an instance* $\mathscr{A}$ *of* A *such that* $\langle \mathscr{A}, Q \rangle$ *is an argument wrt* $\langle \Phi, \Delta \rangle$.

In the definition above the set Φ stands for a state of the world (set of observations). Note that the set Φ must necessarily be non-contradictory to model a coherent scenario.

The first step to obtain the pre-compiled knowledge component of a given program $\mathscr{P} = \langle \Psi, \Delta \rangle$ is to record every potential argument that can be obtained from Δ. Unfortunately, Definition 20 does not provide an effective procedure for this task.

Besides, not every subset of the rules in Δ is a potential argument. As the following example shows, some additional constraints must be satisfied.

Example 4. Consider the program in Example 1. Then the set of defeasible rules

$$\mathsf{A} = \{\texttt{move\_junk(X)} \prec \texttt{virus(X)}; \sim \texttt{spam(X)} \prec \texttt{local(X)}\}$$

is not a potential argument with respect to Δ. Given any instance $\mathscr{A}$ of A it is easy to see that $\mathscr{A}$ does not satisfy the minimality restriction in Definition 10.

In order to find the set of potential arguments of a given program in an efficient manner, we will introduce a constructive definition. First, we will define two auxiliary concepts.

Definition 21 (Heads-Bodies-Literals [24]). *Let $\mathscr{A}$ be an argument for Q, then heads($\mathscr{A}$) is the set of all literals that appear as heads of rules in $\mathscr{A}$. Similarly, bodies($\mathscr{A}$) is the set of all literals that appear in the bodies of rules in $\mathscr{A}$. The set of all literals appearing in $\mathscr{A}$, denoted literals($\mathscr{A}$) is the set heads($\mathscr{A}$)$\cup$ bodies($\mathscr{A}$).*

Definition 22 (Argument Support). *Let $\mathscr{A}$ be an argument for Q, we will say that the set*

$$\mathscr{S}(\mathscr{A}) = \textit{bodies}(\mathscr{A}) - \textit{heads}(\mathscr{A})$$

is the support of $\mathscr{A}$.

Example 5. For argument $\langle \mathscr{B}, \sim \texttt{move\_inbox(d)}\rangle$ in Example 2,

$$\mathscr{B} = \{\ \sim \texttt{move\_inbox(d)} \prec \sim \texttt{filters(d)},\texttt{move\_junk(d)};$$
$$\texttt{move\_junk(d)} \prec \texttt{spam(d)}; \texttt{spam(d)} \prec \texttt{black\_list(d)}\}$$

the corresponding sets are:

$$\begin{aligned}
\textit{heads}(\mathscr{B}) &= \{\sim\texttt{move\_inbox(d)}, \texttt{move\_junk(d)}, \texttt{spam(d)}\}\\
\textit{bodies}(\mathscr{B}) &= \{\tilde{}\,\texttt{filters(d)}, \texttt{move\_junk(d)}, \texttt{spam(d)}, \texttt{black\_list(d)}\}\\
\textit{literals}(\mathscr{B}) &= \{\tilde{}\,\texttt{move\_inbox(d)}, \sim \texttt{filters(d)},\texttt{move\_junk(d)},\\
&\qquad \texttt{spam(d)}, \texttt{black\_list(d)}\}
\end{aligned}$$

Definition 23 (Potential argument – constructive version). *Let Δ be a set of defeasible rules. A subset A of Δ is a potential argument for a literal Q, denoted as $\langle\!\langle \mathsf{A}, Q\rangle\!\rangle$, if and only if there exists an instance $\mathscr{A}$ of A such that*:

1. *The set literals($\mathscr{A}$) of literals in $\mathscr{A}$ is a non-contradictory set,*
2. $\mathscr{S}(\mathscr{A}) \cup \mathscr{A} \mid\!\sim P$ (*where P is a ground instance of Q*), *and*
3. *There is no $\mathscr{A}' \subset \mathscr{A}$ such that $\mathscr{S}(\mathscr{A}') \cup \mathscr{A}' \mid\!\sim P$.*

Both definitions are equivalent, as we will show next:

Proposition 2. *Definition 23 is equivalent to Definition 20.*

Proof. (1) $\Rightarrow$ (2): Let A be a set of defeasible rules that satisfies the conditions in Definition 23 and let $\mathscr{A}$ be an instance of A. To satisfy the existence condition in Definition 20 it is enough to consider $\mathscr{A}$ as the required argument with $\Phi = \mathscr{S}(A)$. It remains to show that $\mathscr{A}$ is an argument with respect to $\langle \Phi, \Delta \rangle$. As the set of rules in $\mathscr{A}$ is non-contradictory, $\mathscr{S}(A)$ must also be non-contradictory. In this case it is easy to prove that $\mathscr{A}$ is an argument for P with respect to $\langle \Phi, \Delta \rangle$. Let us consider the conditions in Definition 2: clearly $\mathscr{A} \cup \Phi \mid\!\sim Q$, given that $\Phi = \mathscr{S}(\mathscr{A})$ and $\mathscr{S}(\mathscr{A}) \cup \mathscr{A} \mid\!\sim Q$. Next, it should be the case that $\mathscr{A}$ is non-contradictory with respect to Φ. This is easy to show, since *literals*(A) is non-contradictory. Finally, $\mathscr{A}$ also fulfills the minimality condition, as it is required explicitly by the last condition in Definition 23.

(2) $\Rightarrow$ (1): trivial by definition. □

Pre-compiled knowledge associated with an ODeLP program $\mathscr{P} = \langle \Psi, \Delta \rangle$ will involve the set of *all* potential arguments that can be built from $\mathscr{P}$ as well as the defeat relations among them. From Proposition 1 it follows that potential arguments can be found by analyzing the subsets of Δ that satisfy with Definition 23.

– **Computing potential arguments**: to obtain and record every potential argument of $\mathscr{P}$ we use Definition 23. Potential arguments will save time in computing arguments when solving queries. Instead of computing a query for a given ground literal Q, the ODeLP interpreter will search for a potential argument A for Q such that a particular instance $\mathscr{B}$ of A is an argument for Q wrt $\mathscr{P}$.
– **Defeat relations among potential arguments**: Recording information about defeat relations among potential arguments is also useful as it helps to speed up the construction of dialectical trees when solving queries, as we will see later. To do this, we extend the concepts of counterargument and defeat for potential arguments. A potential argument $\langle\!\langle \mathsf{A}_1, Q \rangle\!\rangle$ *counter-argues* $\langle\!\langle \mathsf{A}_2, Q \rangle\!\rangle$ at a literal Q if and only if there is a non-empty potential sub-argument $\langle\!\langle \mathsf{A}, Q \rangle\!\rangle$ of $\langle\!\langle \mathsf{A}_2, Q \rangle\!\rangle$ such that Q_1 and Q are contradictory literals.[4] Note that potential counter-arguments may or may not result in a real conflict between the instances (arguments) associated with the corresponding potential arguments. In some cases instances of these arguments cannot co-exist in any scenario (e.g., consider two potential arguments based on contradictory observations).
  The notion of defeat is also extended to potential arguments. Since specificity is a syntactic-based criterion, a particular version of specificity [27] is applicable to potential arguments, determining when a potential argument is more informed or more direct than another.

To search for an argument for a given query Q, we will use the algorithm shown in Figure 3, that uses backward chaining on the set of defeasible rules performing the required substitutions. Therefore, for completeness, previous to the presentation of that algorithm, we introduce some auxiliary definitions related to the concept of substitution [25].

```
Algorithm 1 Find Instance
Input:  ⟨⟨A, Q⟩⟩, Q_1 {Q_1 is a ground instance of Q},
        Ψ { Observation set}
Output: ⟨𝒜, Q_1⟩ {Such that ⟨𝒜, Q_1⟩ is an instance of ⟨⟨A, Q⟩⟩ if one exists}
CreateStack(S)
push(Q_1,S)
θ:= {}
While S is not empty
    goal := pop(S)
    If there exists a rule r in A and a sustitution σ
       such that head(r)σ = goal
    then
        new_body := apply σ to the body of the rule r
        θ := compose θ and σ
        push(new_body,S)
    else
        r := pop(S)
        If there exists a substitution σ and an observation α in Ψ
           such that rσ = α
        then θ := compose θ and σ
        else fail {It is not possible to find an instance}
𝒜 := apply θ to every rule in A
Return(⟨𝒜, Q_1⟩)
```

Figure 3. Algorithm to find an instance of a potential argument.

Definition 24 (Substitution). *A substitution θ is a finite set $\{v_1/t_1, \ldots, v_n/t_n\}$ where every v_i is a variable, no two variables v_i, v_j are syntactically equal, and for every i and every j, v_i does not appear in t_j. If all of the terms t_i, $1 \leq i \leq n$, are ground, then θ is a ground substitution.*

As usual we will write $\{v_1, \ldots, v_n\}/\theta$ to denote the result of applying a substitution θ to every member of the set $\{v_1, \ldots, v_n\}$. If r is a defeasible rule we will write r/θ to denote the result of applying the substitution θ to the body and head of r.

Definition 25 (Instance of a literal by a substitution). *Let $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ be a substitution and L a literal. Then the instance of L by θ is obtained by simultaneously replacing every occurrence of v_i in L for the term t_i.*

Definition 26 (Composition of substitutions). *Let $\theta = \{u_1/s_1, \ldots, u_n/s_m\}$ and $\sigma = \{v_1/t_1, \ldots, v_n/t_n\}$ be two substitutions. The composition $\theta\sigma$ of θ and σ is the substitution that is given by the set*

$$\{(u_1/s_1)\sigma, \ldots, (u_n/s_m)\sigma, v_1/t_1, \ldots, v_n/t_n\}$$

removing every $(u_i/s_i)\sigma$ such that $u_i = s_i\sigma$ and any v_j/t_j such that $v_j \in \{u_1, \ldots, u_m\}$.

Note that the algorithm in Figure 3 requires defeasible rules in the potential argument to be standarized apart so that they do not contain common variables.

That is, for any pair of rules r_1, r_2 in A it must hold that the intersection between the set of variables in r_1 and the set of variables in r_2 is empty.

Using potential arguments and their associated defeat relations we can formally define the notion of *Dialectical Database* associated with a given ODeLP program $\mathscr{P}$.

Definition 27 (Dialectical Database). *Let $\mathscr{P} = \langle \Psi, \Delta \rangle$ be an ODeLP program. The dialectical database of Δ, denoted as $\mathbb{D}_\Delta$, is a 3-tuple $(PotArg(\Delta), D_p, D_b)$ such that:*

1. *$PotArg(\Delta)$ is the set $\{\langle\langle \mathsf{A}_1, Q\rangle\rangle, \ldots, \langle\langle \mathsf{A}_k, Q\rangle\rangle\}$ of all the potential arguments that can be built from Δ.*
2. *D_p and D_b are relations over the elements of $PotArg(\Delta)$ such that for every pair $(\langle\langle \mathsf{A}_1, Q\rangle\rangle, \langle\langle \mathsf{A}_2 Q\rangle\rangle)$ in D_p (respectively D_b) it holds that $\langle\langle \mathsf{A}_2, Q\rangle\rangle$ is a proper (respectively blocking) defeater of $\langle\langle \mathsf{A}_1, Q\rangle\rangle$.*

Example 6. Consider the ODeLP program given in Example 1. The dialectical database of $\mathscr{P}$ is composed by the potential arguments shown in Figure 4 and the defeat relations recorded in Figure 5, where the fact that A_1 properly defeats A_2 is

$\langle\langle \mathsf{A}_1, \mathtt{\sim move\_inbox(X)}\rangle\rangle$,
where $\mathsf{A}_1 = \{\mathtt{\sim move\_inbox(X)} \prec \mathtt{\sim filters(X),\ move\_junk(X)}\}$

$\langle\langle \mathsf{A}_2, \mathtt{\sim move\_inbox(X)}\rangle\rangle$,
where $\mathsf{A}_2 = \{\mathtt{\sim move\_inbox(X)} \prec \mathtt{\sim filters(X),\ move\_junk(X)};$
$\mathtt{move\_junk(X)} \prec \mathtt{virus(X)}\}$

$\langle\langle \mathsf{A}_3, \mathtt{\sim move\_inbox(X)}\rangle\rangle$,
where $\mathsf{A}_3 = \{\mathtt{\sim move\_inbox(X)} \prec \mathtt{\sim filters(X),\ move\_junk(X)};$
$\mathtt{move\_junk(X)} \prec \mathtt{spam(X)}\}$

$\langle\langle \mathsf{A}_4, \mathtt{\sim move\_inbox(X)}\rangle\rangle$,
where $\mathsf{A}_4 = \{\mathtt{\sim move\_inbox(X)} \prec \mathtt{\sim filters(X),\ move\_junk(X)};$
$\mathtt{move\_junk(X)} \prec \mathtt{spam(X)}, \mathtt{spam(X)} \prec \mathtt{black\_list(X)}\}$

$\langle\langle \mathsf{A}_5, \mathtt{move\_inbox(X)}\rangle\rangle$,
where $\mathsf{A}_5 = \{\mathtt{move\_inbox(X)} \prec \mathtt{\sim filters(X)}\}$

$\langle\langle \mathsf{A}_6, \mathtt{\sim move\_inbox(X)}\rangle\rangle$,
where $\mathsf{A}_6 = \{\mathtt{\sim move\_inbox(X)} \prec \mathtt{filters(X)}\}$

$\langle\langle \mathsf{B}_1, \mathtt{move\_junk(X)}\rangle\rangle$,
where $\mathsf{B}_1 = \{\mathtt{move\_junk(X)} \prec \mathtt{virus(X)}\}$

$\langle\langle \mathsf{B}_2, \mathtt{move\_junk(X)}\rangle\rangle$,
where $\mathsf{B}_2 = \{\mathtt{move\_junk(X)} \prec \mathtt{spam(X)}\}$

$\langle\langle \mathsf{B}_3, \mathtt{move\_junk(X)}\rangle\rangle$,
where $\mathsf{B}_3 = \{\mathtt{move\_junk(X)} \prec \mathtt{spam(X)}; \mathtt{spam(X)} \prec \mathtt{black\_list(X)}\}$

$\langle\langle \mathsf{C}_1, \mathtt{spam(X)}\rangle\rangle$,
where $\mathsf{C}_1 = \{\mathtt{spam(X)} \prec \mathtt{black\_list(X)}\}$

$\langle\langle \mathsf{C}_2, \mathtt{\sim spam(X)}\rangle\rangle$,
where $\mathsf{C}_2 = \{\mathtt{\sim spam(X)} \prec \mathtt{black\_list(X),contacts(X)}\}$

$\langle\langle \mathsf{C}_3, \mathtt{\sim spam(X)}\rangle\rangle$,
where $\mathsf{C}_1 = \{\mathtt{\sim spam(X)} \prec \mathtt{local(X)}\}$

Figure 4. Potential arguments for Example 6.

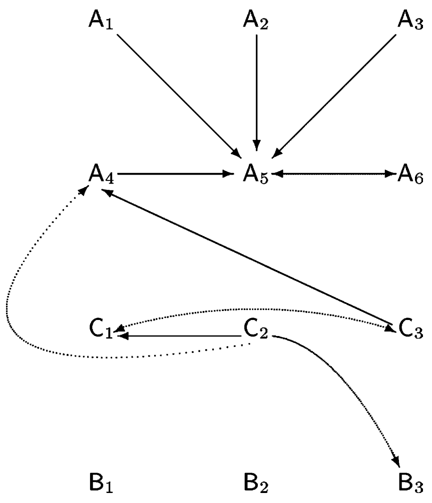

Figure 5. Dialectical database of Example 6.

indicated with an arrow from A_1 to A_2. Proper defeaters are indicated using a solid arrow and blocking defeaters are distinguished with a dotted arrow.

3.2. Dialectical databases: a structure for speeding up inference

Given a ODeLP program $\mathcal{P}$, its dialectical database $\mathbb{D}_\Delta$ can be understood as a graph from which *all* possible dialectical trees computable from $\mathcal{P}$ can be obtained. In the original ODeLP framework (as detailed in Section 2), solving a query Q wrt a given program $\mathcal{P} = \langle\Psi, \Delta\rangle$ accounted for obtaining a warranted argument $\langle\mathcal{A}, Q\rangle$. As already discussed, computing warrant involves many intermediate steps which are computationally expensive (computing arguments, detecting defeaters, building a dialectical tree, etc.).

Using the dialectical database the inference process can be improved by keeping track of all possible potential arguments and the defeat relationships among them. Generating this structure is computationally expensive, and therefore it is important for the dialectical database not to be recalculated every time a new perception is incorporated. Note that, if new rules could also be perceived, the dialectical database should have to be built from scratch every time a new rule is introduced. It is for this reason that only new facts can be perceived.

Given a query Q, the extended framework (i.e., including the dialectical database) will select first a potential argument $\langle\langle\mathsf{A}, L\rangle\rangle$ (such that Q is a ground instance of L) that can be instantiated into $\langle\mathcal{A}, Q\rangle$, supporting Q. From the D_p and D_b relations in $\mathbb{D}_\Delta$ the potential defeaters for $\langle\mathsf{A}, Q\rangle$ can be identified, and also instantiated: traversing the graph we recover the dialectical tree for $\langle\mathcal{A}, Q\rangle$ in a top-down fashion.

The algorithm given in Figure 6 illustrates how the inference process is assisted by dialectical databases. when computing a warrant for a query Q from a program $\mathcal{P} = \langle\Psi, \Delta\rangle$. To do this, the algorithm considers first the potential arguments $\langle\langle\mathsf{A}, L\rangle\rangle$

```
Algorithm 2 Inference Process
Input: $\mathcal{P} = \langle \Psi, \Delta \rangle$, $Q$, $PotArg(\Delta)$
Output: $\langle \mathcal{A}, Q \rangle$ {A warrant for Q, if any}

        {Looks for instances of the potential arguments that support Q}
For every $\langle\langle \mathsf{A}, L \rangle\rangle$ in $PotArg(\Delta)$ such that
        $\langle \mathcal{A}, Q \rangle$ is an instance of $\langle\langle \mathsf{A}, L \rangle\rangle$ and Acceptable($\langle \mathcal{A}, Q \rangle$,$\mathcal{P}$)
        {and then determines the state of their defeaters}
        Defeat-state := undefeated
        For every $\langle\langle \mathsf{B}, X \rangle\rangle$ in $PotArg(\Delta)$ such that $(\mathsf{B}, \mathsf{A}) \in D_p$ or $(\mathsf{B}, \mathsf{A}) \in D_p$
                For every instance $\langle \mathcal{B}, R \rangle$ of $\langle\langle \mathsf{B}, X \rangle\rangle$ such that
                $\langle \mathcal{B}, R \rangle$ defeats $\langle \mathcal{A}, Q \rangle$ and Acceptable($\langle \mathcal{B}, R \rangle$,$\mathcal{P}$)
                        {Sets the state of the main argument according to its defeaters}
                        If state($\langle \mathcal{B}, R \rangle$, $\mathcal{P}$, $\emptyset$, $\{\langle \mathcal{A}, Q \rangle\}$) = undefeated
                            then Defeat-state := defeated
        {If any of the instances remains undefeated it is a warrant}
If Defeat-state = undefeated
        then Return($\langle \mathcal{A}, Q \rangle$)
```

Figure 6. Algorithm for modeling inference in ODeLP assisted by a dialectical database.

such that Q is an instance of L, trying to find an instance $\langle \mathscr{A}, Q \rangle$ of $\langle\langle \mathsf{A}, L \rangle\rangle$ that is also an argument with respect to $\mathscr{P}$, according to Definition 19. This is done in function `Find Instance` which in case such an instance exists returns it as $\langle \mathscr{A}, Q \rangle$. Next, argument $\langle \mathscr{A}, Q \rangle$ is analyzed to see whether it is a warrant for Q. To do this, the relations D_p and D_b are used to find all possible defeaters for $\langle \mathscr{A}, Q \rangle$. Once the algorithm finds an instance of the potential defeaters that is in conflict with $\langle \mathscr{A}, Q \rangle$, the function `Acceptable` checks if they are well-formed arguments with respect to the program $\mathscr{P}$. Then the `State` function (see algorithm in Figure 7) determines the marking of these defeaters (i.e., if they are labeled as `U-nodes` or `D-nodes`) and finally this information is used to compute the state of $\langle \mathscr{A}, Q \rangle$.

The `State` algorithm used in the inference process takes as input an ODeLP program $\mathscr{P}$, an argument $\langle \mathscr{A}, Q \rangle$ based on it, and the *interference* and *support* argumentation lines up to this point, denoted as `IL` and `SL`, resp. Simply put, `IL` represents the set of arguments with an even level in the current path of the tree under construction, and `SL` the arguments with an odd level. Then the `State` algorithm works like the algorithm in Figure 6, analyzing the defeaters for $\mathscr{A}$ to define its state. However, one extra condition must be met: defeaters must also comply with the rules established for avoiding fallacies. This test is performed by the function `Valid`.

Figure 8 summarizes the main elements of the ODeLP-based agent architecture. The agent's knowledge is represented by an ODeLP program $\mathscr{P}$. Perceptions from the environment result in changes in the set of observations in $\mathscr{P}$, handled by an appropriate updating mechanism as discussed previously. In order to solve queries from other agents, the agent relies on the ODeLP inference engine. Queries are speeded up by first searching on the potential arguments stored in the dialectical database, applying the algorithms discussed before. The final answer to a given query Q will be *yes*, *no* or *undecided*, according to the warrant status of Q with respect to $\mathscr{P}$.

```
Algorithm 3 State
Input: $\langle\mathcal{A},Q\rangle$, $\mathcal{P}$, IL, SL
Output: State

State := undefeated
{Uses the stored defeat relation to find the defeaters for $\langle\mathcal{B},Q\rangle$}
For every pair (A, B) $\in D_p \cup D_b$ such that $\langle\mathcal{A},Q\rangle$ is an instance of A
      For every instance $\langle\mathcal{B},S\rangle$ of B such that
            Acceptable($\langle\mathcal{B},S\rangle$,$\mathcal{P}$) and Valid($\langle\mathcal{B},S\rangle$,IL,SL)
      {Then checks for every defeater whether it gives raise to fallacies}
      If $\langle\mathcal{A},Q\rangle$ is in SL and State($\langle\mathcal{B},S\rangle$,$\mathcal{P}$,IL,SL $\cup$ {$\mathcal{A}$}) = undefeated
                  then State := defeated
      If $\langle\mathcal{A},Q\rangle$ is in IL and State($\langle\mathcal{B},S\rangle$,$\mathcal{P}$,IL $\cup$ {$\mathcal{A}$},SL) = undefeated
                  then State := defeated
      {The recursive call does the same for the defeaters of $\langle\mathcal{B},S\rangle$}
Return(State)
```

Figure 7. Algorithm state.

4. A case study

In this section we present a toy example to illustrate the use of a dialectical database to speed up inference in ODeLP. Let us consider the ODeLP program $\mathcal{P}_{mail} = \langle\Psi,\Delta\rangle$ in Example 1 as the knowledge base of an email filtering agent. The associated dialectical database $\mathbb{D}_\Delta$ is shown in Example 6.

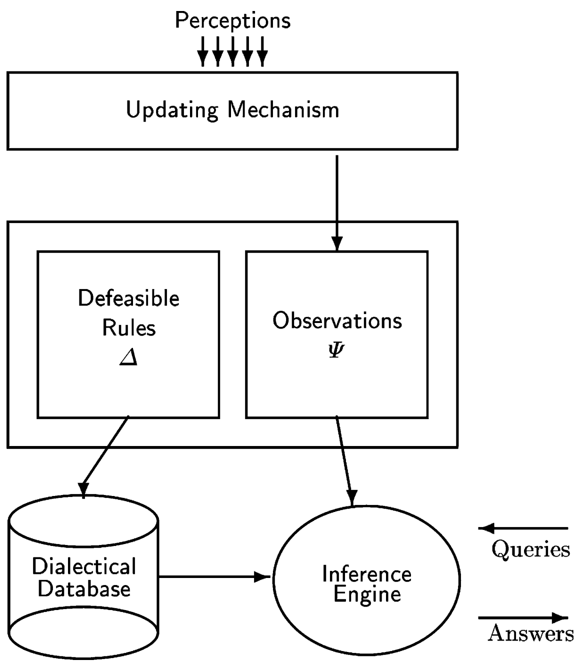

Figure 8. Agent architecture using ODeLP as underlying framework.

Suppose that the agent has to decide where to place a given message *d*. To see if *d* must be placed in the inbox, the agent tries to solve the query `move_inbox(d)`. As shown in Example 2, solving this query wrt $\mathscr{P}$ involves a dialectical tree with four arguments (see Figure 2). Let us analyze how the agent would proceed to perform the same inference using the dialectical database $\mathbb{D}_\Delta$. Following the algorithm in Figure 6, the potential argument $\langle\langle \mathsf{A}_5, \texttt{move\_inbox(X)}\rangle\rangle$ will be instantiated resulting in the argument $\langle \mathscr{A}, \texttt{move\_inbox(d)}\rangle$, with $\mathscr{A} = \{\texttt{move\_inbox(d)} \prec \sim \texttt{filters}(d)\}$.

From the dialectical database $\mathbb{D}_\Delta$ it follows that $\langle\langle \mathsf{A}_5, \texttt{move\_index(X)}\rangle\rangle$ has defeaters $\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3, \mathsf{A}_4, \mathsf{A}_6$ (see the figure in Example 6), but only A_4 is active according to the current set of observations. This argument is instanciated to $\langle \mathscr{B}, \sim \texttt{move\_inbox(d)}\rangle$, where $\mathscr{B} = \{\sim \texttt{move\_inbox(d)} \prec \sim \texttt{filters(d)}, \texttt{move\_junk(d)}; \texttt{move\_junk(d)} \prec \texttt{spam(d)}; \texttt{spam(d)} \prec \texttt{black\_list(d)}\}$.
From the graph associated with the dialectical database, two new defeaters are found from A_4, namely A_2 and A_3. These are, respectively, instanciated to $\langle \mathscr{C}, \sim \texttt{spam(d)}\rangle$, where $\mathscr{C} = \{\sim \texttt{spam(d)} \prec \}\texttt{black\_list(d)}, \texttt{contacts(d)}\}$ and $\langle \mathscr{D}, \sim \texttt{spam(d)}\rangle$, where $\mathscr{D} = \{\sim \texttt{spam(d)} \prec \}\texttt{local(d)}\}$. Note that from the information in $\mathbb{D}_\Delta$ associated with C_3 and C_2 there are no more links in the graph to new defeaters for these potential arguments that can be instanciated to defeat $\langle \mathscr{C}, \sim \texttt{spam(d)}\rangle$ or $\langle \mathscr{D}, \sim \texttt{spam(d)}\rangle$. As a consequence, a dialectical tree identical to the one shown in Figure 2 has been computed on the basis of the potential arguments present in the dialectical database and their associated defeat relations.

Consider now a different situation for the same sample program $\mathscr{P}$. Suppose that the fact `spam(d)` is added to the set of observations given that the user now specifically defines this message as spam. Solving the query `move_inbox(d)` begin as before with potential argument $\langle\langle \mathsf{A}_5, \texttt{move\_index(X)}\rangle\rangle$ instantiated to $\langle \mathscr{A}, \texttt{move\_inbox(d)}\rangle$, but now the only active defeater for A_5 is A_3, that instanciates to $\langle \mathscr{E}, \sim \texttt{move\_inbox(d)}\rangle$, where

$$\mathscr{E} = \{\sim \texttt{move\_inbox(d)} \prec \sim \texttt{filters(d)}, \texttt{move\_junk(d)}; \texttt{move\_junk(d)} \prec \texttt{spam(d)}\}$$

This concludes the analysis since A_3 has no defeaters and therefore there is no warrant for `move_inbox(d)`. Note that new perceptions do not provoke changes in the dialectical base, but nevertheless the set of beliefs of the agent changes and the fact `move_inbox(d)` is no longer believed.

5. Conclusions and future work

Solid theoretical foundations for agent design should be based on proper formalisms for knowledge representation and reasoning [33]. Thus, we have defined a framework for representing knowledge and beliefs of agents in dynamic environments, where new perceptions can modify the agent's view about its surroundings. To comply with real time issues when modeling agent interaction in a MAS setting we have proposed the notion of dialectical databases.

We have discussed the main issues in the integration of this component into ODeLP, such as building the dialectical database, adapting the specificity criterion for potential arguments and modifying the inference process to take advantage of the new component. Based on this, we can affirm that the use of pre-compiled knowledge can improve the performance of argument-based systems in the same way TMS assist general problem solvers. We believe that the approach presented in this paper can also be applied to other frameworks, paving the way for the development of more efficient argument-based applications.

Part of our current work involves extending the analysis of ODeLP properties presented in [27] to the context of MAS. We are also working on a complexity analysis of ODeLP for the algorithms used in the construction and utilization of dialectical databases.

Acknowledgements

This research was partially supported by Projects TIC2001-1577-C03-01, TIN 2004-07933-C03-03 and TIC2003-00950, by Ramón y Cajal Program (*Ministerio de Ciencia y Tecnología*, Spain), by CONICET (Argentina), by CIC (Argentina), by the *Secretaría General de Ciencia y Tecnología de la Universidad Nacional del Sur* and by *SEPCyT–FONCyT–Agencia Nacional de Promoción Científica y Tecnológica* (PICT 2002 No. 13096, PICT 2003 No. 15043).

Notes

** Extended version of "An Argument-Based Framework to Model An Agent's Beliefs in a Dynamic Environment," ArgMAS 2004, LNAI 3366, pp. 96–111, 2005, I. Rahwan et al. (Eds.)
1. See [10] for a precise definition.
2. For the sake of clarity, we use semicolons to separate elements in an argument $\mathcal{A} = \{e_1; e_2; \ldots; e_k\}$.
3. For a full discussion of ODeLP properties and their proof the interested reader is referred to [27].
4. Note that $P(X)$ and $\sim P(X)$ are contradictory literals although they are non-grounded. The same idea is applied to identify contradiction in potential arguments.

References

1. M. J. Wooldridge, *Introduction to Multiagent Systems*. Wiley, 2002.
2. C. I. Chesñevar, A. Maguitman, and R. Loui, "Logical models of argument," *ACM Comput. Surv.*, Vol. 32, pp. 337–383, 2000.
3. H. Prakken and G. Vreeswijk, "Logical systems for defeasible argumentation," in D. Gabbay, (ed.), *Handbook of Philosophical Logic*, Vol. 4, Kluwer Academic Publisher: Dordrecht 2002, pp. 219–318.
4. M. Gelfond and V. Lifschitz, " Classical negation in logic programs and disjunctive databases," *New Generation Comput.*, pp. 365–385, 1991.
5. A. García and G. Simari,"Defeasible logic programming: an argumentative approach," *Theory and Practice of Logic Programming* vol. 4, pp. 95–138, 2004.
6. C. Alchourrón, P. Gärdenfors and D. Makinson, " On the logic of theory change: Partial meet contraction and revision functions," *J. Symbolic Logic* vol. 50, pp. 510–530, 1985.
7. H. Katsuno and A. Mendelzon, " On the difference between updating a knowledge base and revising it," in P. Gardenfors (ed.), *Belief Revision*, Cambridge University Press, 1992, pp. 183–203.

8. P. Gäardenfors, *Knowledge in Flux: Modelling the Dynamics of Epistemic States*. The MIT Press, Bradford Books, Cambridge, MA, 1988.
9. J. L. Pollock, "Taking perception seriously," in: *Proceedings of the 1st International Conference on Autonomous Agent*, 1997, pp. 526–527.
10. J. Doyle, "A truth maintenance system," *Artif. Intell.*, vol. 12, pp. 231–272, 1979.
11. J. de Kleer, "An assumption-based TMS," *Artif. Intell.*, vol. 28, pp. 127–162, 1986.
12. J. de Kleer,"A comparison of ATMS and CSP techniques," in N. S. Sridharan (ed.), *Proceedings of the 11th IJCAI, Workshop on Practical Reasoning and Rationality*, Detroit, USA, 1989, pp. 290–296.
13. C. Elkan, "A rational reconstruction of nonmonotonic truth maintenance systems," Artif. Intell., vol. 43, pp. 219–234, 1990.
14. D. McAllester, "Truth maintenance," in R. Smith and T. Mitchell (eds.), *Proceedings of the 8th National Conference on Artificial Intelligence*. vol. 2., American Association for Artificial Intelligence, AAAI Press, 1990, pp. 1109–1116.
15. K. Forbus and J. de Kleer, *Building Problem Solvers*, MIT Press, Cambridge, MA, 1993.
16. A. L. Brown,"Modal propositional semantics for reason maintenance systems," in: *Proceedings of International Joint Conference on Artificial Intelligence*, 1985, pp. 178–183.
17. C. Chesñevar, A. Maguitman, and R. Loui, "Logical models of argument," *ACM Comput. Surv.*, vol. 32, pp. 337–383, 2000.
18. C. Reed and T. E. Norman, *Argumentation Machines – New Frontiers in Argument and Computation*, vol. 9 of Series Argumentation Library, Springer, 2005.
19. H. Prakken and G. Sartor, "Argument-based extended logic programming with defeasible priorities," *Appl. Non-classical Logics*, vol. 7, pp. 25–752, 1997.
20. I. Rahwan, S. Ramchurn, N. Jennings, P. McBurney, S. Parsons, and L. Sonenberg, " Argumentation-based negotiation," *Knowl. Eng. Rev.*, vol. 18, pp. 343–375, 2003.
21. D. Carbogim, D. Robertson, and J. Lee,"Argument-based applications to knowledge engineering", *Knowl. Eng. Rev.*, vol. 15, pp. 119–149, 2000.
22. C. I. Chesñevar, G. Simari, T. Alsinet and L. Godo,"A logic programming framework for possibilistic argumentation with vague knowledge," in: *Proceedings of the International Conference in Uncertainty in Artificial Intelligence (UAI 2004)*, Canada, 2004, pp. 76–84.
23. C. I. Chesñevar, G. Simari, T. Alsinet, and L. Godo,"Modelling agent reasoning in a logic programming framework for possibilistic argumentation," in: *Proceedings of the 2nd European Workshop on Multiagent Systems*. Barcelona, Spain, 2004, pp. 135–142.
24. G. R. Simari and R. P. Loui, "A mathematical treatment of defeasible reasoning and its implementation," *Artif. Intell.*, vol. 53, pp. 125–157, 1992.
25. J. W. Lloyd, *Foundations of Logic Programming*. Springer-Verlag, 1987.
26. D. L. Poole, "On the comparison of theories: preferring the most specific explanation," in: *Proceedings of the 9th IJCAI*, 1985, pp. 144–147.
27. M. Capobianco, "Argumentación rebatible en entornos dinámicos," Ph.D. thesis, Universidad Nacional del Sur, Bahía Blanca, Argentina, 2003.
28. G. R. Simari, C. I. Chesñevar, and A. J. García, "The role of dialectics in defeasible argumentation," in: *Proceedings of the XIV Conferencia Internacional de la Sociedad Chilena para Ciencias de la Computación*, 1994, pp. 111–121.
29. C. Chesñevar, J. Dix, F. Stolzenburg, and G. Simari, "Relating defeasible and normal logic programming through transformation properties," *Theor. Comput. Sci.*, vol. 290, pp. 499–529, 2003.
30. J. J. Alferes and L. M. Pereira, "On logic program semantics with two kinds of negation," in: *Proceedings of the Joint International Conference and Symposium on Logic Programming*, USA, 1992, pp. 574–588.
31. C. Chesñevar and G. R. Simari, "Distinguishing ground from nonground information in defeasible argumentation," in: *Proceedings del I Congreso Argentino en Ciencias de la Computación*, Universidad Nacional del Sur, Argentina, 1995, pp. 527–538.
32. A. J. García, C. I. Chesñevar, and G. R. Simari," Making argumentative systems computationally attractive," in: *XIII International Conference of the Chilenean Computer Science Society*, 1993, pp. 335–344.
33. C. Baral and M. Gelfond, "Reasoning agents in dynamic domains," in: J. Minker, (ed.), *Workshop on Logic-Based Artificial Intelligence*, College Park, Maryland, Computer Science Department, University of Maryland, 1999.

34. L. Amgoud and C. Cayrol,"On the use of an atms for handling conflicting desires," in: *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, 2004, pp. 194–202.
35. Phan Minh Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artif. Intell.*, vol. 77, pp. 321–358, 1995.