A Formalization of Dialectical Bases for Defeasible Logic Programming

Marcela Capobianco Carlos I. Chesñevar
Grupo de Investigación en Inteligencia Artificial (GIIA)
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
Av. Alem 1253 - (8000) Bahía Blanca - ARGENTINA
E-mail: {mc,cic}@cs.uns.edu.ar

Abstract

Defeasible argumentation is a form of defeasible reasoning that emphasizes the notion of arguments. An argument \( \mathcal{A} \) for a conclusion \( q \) is a tentative piece of reasoning which supports \( q \). In an argumentative framework, common-sense reasoning can be modeled as a process in which we must determine whether an argument justifies its conclusion.

Defeasible logic programming (DeLP) is an extension of the current work on logic programming, which uses an argumentation formalism for deciding between contradictory goals through a dialectical analysis.

The process of finding justifications in the DeLP framework takes considerable computational effort. For this reason, it would be convenient to maintain a repository of the previously computed justifications, in order to avoid executing the query solving process several times on the same input.

Keywords: non-monotonic reasoning, defeasible argumentation, argumentative frameworks.

1 Introduction

Defeasible argumentation is a form of defeasible reasoning [10, 9] which emphasizes the notion of arguments [15]. An argument \( \mathcal{A} \) for a conclusion \( q \), denoted \( \langle \mathcal{A}, q \rangle \), is a tentative piece of reasoning that supports \( q \). In an argumentative setting, commonsense reasoning can be modeled as a process in which different arguments, supporting or rebutting a conclusion, compete for getting the system to accept its conclusion.

During the last decade several alternative frameworks for defeasible argumentation have been developed [15, 11, 13, 1, 6] (we refer the interested reader to [4] for a complete review about different argumentative frameworks). Particularly, in A Mathematical Treatment of Defeasible Reasoning [15] (MTDR for short), a clear and theoretically sound framework for defeasible argumentation was introduced, which has come to be a fairly standard approach.

In MTDR (as well as in many other argumentative frameworks) to decide the acceptability of an argument \( \langle \mathcal{A}, q \rangle \), possible counterarguments against accepting \( \langle \mathcal{A}, q \rangle \) are considered. Since counterarguments are also arguments, they have to be in turn tested for acceptability, resulting in a recursive procedure in which arguments, counterarguments, and so on, have to be taken into account. Finally, those counterarguments against \( \langle \mathcal{A}, q \rangle \) which have been
accepted are compared with $A$ using a preference relation, in order to determine if any of them can prevent $\langle A, q \rangle$ to become acceptable. If $\langle A, q \rangle$ is finally accepted, then $A$ is said to be a justification for $q$.

Defeasible logic programming (DeLP) is an extension of the traditional logic programming, that uses the MTDR argumentation formalism for deciding between contradictory goals, through a dialectical analysis. A query $q$ in the DeLP formalism will succeed if the system can build a justification for $q$.

Computing DeLP justifications demands considerable computational effort, so it would be convenient to maintain a “repository” of the justifications obtained by the system, in order to save work already done with previously solved queries. As a result, the system’s efficiency would be improved. This repository of justifications will be called dialectical base.

Intelligent agents must be able to interact in a dynamic environment since they may learn new facts about the world. Therefore, a system used to model the reasoning process of an intelligent agent must be capable of adding new information into its knowledge base (in our case, the DeLP program encoding the agent’s knowledge). As a result, some of the old justifications stored in the dialectical base may become invalid. The key problem is to decide which elements of the dialectical base are affected by this operation and to define how to accommodate these changes.

This work was inspired by the Truth Maintenance System (TMS) defined by J. Doyle in [5]. A TMS is a combination of a representation for recording justifications for program beliefs, and procedures for performing any update necessary upon the addition of new information. In consequence, this system can be used to efficiently reason about the recorded justifications.

The idea of storing arguments for speeding up inference was first addressed in [8]. This approach became out of date as the original Simari-Loui framework [15] evolved, incorporating dialectical concepts to the argumentation process [14]. In this work we introduce the concept of dialectical bases, which is a key element for defining a justification maintenance system for argumentative frameworks. The main goal is to state the basic principles of this system under the new enhanced formalism.

This work is structured as follows. In section 2 we present an overview of defeasible logic programming. Next, in section 3 we introduce the formal definition of dialectical bases. In section 4 we analyze how to keep the dialectical base properly updated after adding new facts to the knowledge base. Based on this analysis, we define in section 5 the algorithms that specify how to handle this situation. Finally, in section 6 we state the conclusions obtained from this work.

2 Defeasible Logic Programming

This section surveys the main concepts of the DeLP framework (for a more detailed description we refer the interested reader to its original definition [7]). The DeLP language is defined in terms of two disjunct sets of rules: a set of strict rules for representing sound knowledge, and a set of defeasible rules for representing tentative information. Formally:

Definition 2.1. (Strict rule) A strict rule is an ordered pair, denoted by $\text{Head} \leftarrow \text{Body}$, where $\text{Head}$ is a literal and $\text{Body}$ is a finite set of extended literals\(^\dagger\). A strict rule with head $L_0$ and body $\{L_1, \ldots, L_n\}$ can also be written as $L_0 \leftarrow L_1, \ldots, L_n$. As usual, if the body is empty, a strict rule becomes $L \leftarrow \text{true}$ (or simply $L$) and it is called a fact.

\(^{\dagger}\)A extended literal is a literal possibly preceded by a symbol of default negation “not”.


Definition 2.2. (Defeasible rule) A defeasible rule is an ordered pair, denoted by head – body, where head is a literal and body is a finite set of extended literals. A defeasible rule with head $L_0$ and body $\{L_1, \ldots, L_n\}$ can also be written as $L_0 \leftarrow L_1, \ldots, L_n$. If the body is empty, a strict rule becomes $\leftarrow \text{true}$, and it is called a presupposition.

Syntactically, the symbol “$\leftarrow$” is all that distinguishes a defeasible rule from a strong rule. Pragmatically, a defeasible rule is used to represent defeasible knowledge, i.e., tentative information that can be used if nothing is posed against it.

Definition 2.3. (Defeasible logic program) A defeasible logic program (DLP) is a finite set of strict and defeasible rules. If $\mathcal{P}$ is a DLP, we will distinguish the set $\Pi$ of strict rules in $\mathcal{P}$, (which must be non-contradictory) and the subset $\Delta$ of defeasible rules in $\mathcal{P}$. When required, we will denote $\mathcal{P}$ as $(\Pi, \Delta)$.

A defeasible query (or simply a query) is a defeasible rule with empty consequent denoted “$\neg Q_1, \ldots, Q_n$”, where each $Q_i$, (1 $\leq$ $i$ $\leq$ $n$) is a literal. Given a program $\mathcal{P}$, a defeasible derivation for a literal $h$ (abbreviated $\mathcal{P} \models h$), will be a finite set of strict and defeasible rules. A defeasible derivation is obtained like an SLD-derivation, but considering the negation symbol $\neg$ as part of the predicate name. Formally, given a DLP $\mathcal{P}$ and a defeasible query $Q$, an SLD-defeasible refutation of $\mathcal{P} \cup \{Q\}$ is a finite sequence $C_1, C_2, \ldots, C_n$ of variants of strict or defeasible rules of $\mathcal{P}$, provided there exists a sequence $Q = Q_0, Q_1, \ldots, Q_n$ of defeasible queries and a sequence $\Theta_1, \Theta_2, \ldots, \Theta_n$ of mgu’s such that each $Q_{i+1}$ is derived from $Q_i$ and $C_{i+1}$ using $\Theta_{i+1}$, and $Q_n$ is the empty rule. If there exists an SLD-defeasible refutation for $Q$, then the finite set of rules used in the refutation constitutes a defeasible derivation for $Q$.

Although the set $\Pi$ must be non-contradictory, the set $\Delta$, and hence $\mathcal{P}$ itself, may be contradictory. In consequence, in a program that contains contradictory information, two complementary literals may have both a defeasible derivation. In a case like this, the system should have a formal criterion for deciding between them. DeLP uses a defeasible argumentation formalism based on the MTDR system, in order to perform such a task. Next, we will describe the justification process capable of deciding between contradictory goals.

In DeLP, answers to queries must be supported by arguments:

Definition 2.4. (Argument - Sub-argument) Given a DLP $\mathcal{P}$, an argument $\mathcal{A}$ for a query $q$, also denoted $\langle \mathcal{A}, q \rangle$, is a subset of ground instances of the defeasible rules of $\mathcal{P}$, such that: (1) there exists a defeasible derivation for $q$ from $\Pi \cup \mathcal{A}$, (2) $\Pi \cup \mathcal{A}$ is non-contradictory, and (3) $\mathcal{A}$ is minimal with respect to set inclusion. An argument $\langle \mathcal{A}_1, q_1 \rangle$ is a sub-argument of another argument $\langle \mathcal{A}_2, q_2 \rangle$, if $\mathcal{A}_1 \subseteq \mathcal{A}_2$.

When considering a query $q$, an argument $\mathcal{A}$ for $q$ will be built, but arguments that contradict $\mathcal{A}$ (called counterarguments) could also exist.

Definition 2.5. (Counterargument) 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 an sub-argument $\langle \mathcal{A}, q \rangle$ of $\langle \mathcal{A}_2, q_2 \rangle$ such that the set $\Pi \cup \{q_1, q\}$ is contradictory.

Informally, a query $q$ will succeed if the supporting argument is not defeated; that argument becomes a justification. To establish if $\mathcal{A}$ is a non-defeated argument, counterarguments that could be defeaters for $\mathcal{A}$ are considered, i.e., counterarguments that for some criterion are preferred to $\mathcal{A}$. DeLP defines a particular criterion called specificity (see [12, 7]) which favors
an argument with greater information content and/or less use of defeasible rules. Although a preference criterion is required, the notion of defeat can be formulated independently of the criterion begin used.

**Definition 2.6. (Defeater)** An argument $\langle A_1, q_1 \rangle$ defeats $\langle A_2, q_2 \rangle$ at a literal $q$, if and only if there exists a sub-argument $\langle A, q \rangle$ of $\langle A_2, q_2 \rangle$ such that $\langle A_1, q_1 \rangle$ counter-argues $\langle A_2, q_2 \rangle$ at $q$, and either: (1) $\langle A_1, q_1 \rangle$ is “better” that $\langle A, q \rangle$ (then $\langle A_1, q_1 \rangle$ is a proper defeater of $\langle A, q \rangle$); or (2) $\langle A_1, q_1 \rangle$ is unrelated by the preference order to $\langle A, q \rangle$ (then $\langle A_1, q_1 \rangle$ is a blocking defeater of $\langle A, q \rangle$).

Since defeaters are arguments, there may exist defeaters for the defeaters and so on. That prompts for a complete dialectical analysis to determine which arguments are ultimately defeated. As result of the execution of this process, the arguments undefeated arguments will be marked as $U$-nodes, and the defeated ones as $D$-nodes. Next, we state the formal definitions required for this process:

**Definition 2.7. (Dialectical Tree)** Let $A$ be an argument for $q$. A dialectical tree for $\langle A, q \rangle$, denoted $T_{\langle A,q \rangle}$, is recursively defined as follows:

1. A single node labeled with an argument $\langle A, q \rangle$ with no defeaters (proper or blocking) is by itself the dialectical tree for $\langle A, q \rangle$.

2. Let $\langle A_1, q_1 \rangle, \langle A_2, q_2 \rangle, \ldots, \langle A_n, q_n \rangle$ be all the defeaters (proper or blocking) for $\langle A, q \rangle$. We construct the dialectical tree for $\langle A, q \rangle$, $T_{\langle A,q \rangle}$, by labeling the root node with $\langle A, q \rangle$ and by making this node the parent node of the roots of the dialectical trees for $\langle A_1, q_1 \rangle, \langle A_2, q_2 \rangle, \ldots, \langle A_n, q_n \rangle$.

**Definition 2.8. (Marking of the dialectical tree)** Let $\langle A, q \rangle$ be an argument and $T_{\langle A,q \rangle}$ its dialectical tree, then:

1. All the leaves in $T_{\langle A,q \rangle}$ are marked as $U$-nodes.

2. Let $\langle B, h \rangle$ be an inner node of $T_{\langle A,q \rangle}$. Then $\langle B, h \rangle$ will be a $U$-node iff every child of $\langle B, h \rangle$ is a $D$-node. The node $\langle B, h \rangle$ will be a $D$-node iff it has at least a child marked as $U$-node.

To avoid the occurrence of fallacious argumentation [14], it is necessary to establish certain condition over every argumentation line of the dialectical tree. Following [14] this notion is defined as follows. Let $\langle A_0, q_0 \rangle$ be an argument, and let $T_{\langle A_0,q_0 \rangle}$ be it associated dialectical tree. Every path from the root $\langle A_0, q_0 \rangle$ to a leaf $T_{\langle A_0,q_0 \rangle}$ in $T_{\langle A_0,q_0 \rangle}$, denoted $\lambda = [\langle A_0, q_0 \rangle, \langle A_1, q_1 \rangle, \ldots, \langle A_n, q_n \rangle]$, is called an argumentation line for $q_0$.

In each argumentation line $\lambda = [\langle A_0, q_0 \rangle, \langle A_1, q_1 \rangle, \ldots, \langle A_n, q_n \rangle]$, the argument $\langle A_0, q_0 \rangle$ is supporting the main query $q_0$, and every argument $\langle A_i, q_i \rangle$ defeats its predecessor $\langle A_{i-1}, q_{i-1} \rangle$. Therefore, for $k \geq 0$, $\langle A_{2k}, q_{2k} \rangle$ is a supporting argument for $q_0$ and $\langle A_{2k-1}, q_{2k-1} \rangle$ is an interfering argument for $q_0$. In other words, every argument in the line supports $q_0$ or interferes with it. In consequence, an argumentation line can be split in two disjoint sets: $\lambda_S$ of supporting arguments and $\lambda_I$ of interfering arguments.

Fallacious argumentation is avoided by requiring that all argumentation lines be acceptable. Let $\lambda = [\langle A_0, q_0 \rangle, \langle A_1, q_1 \rangle, \ldots, \langle A_n, q_n \rangle]$ be an argumentation line, we will say that $\lambda$ is an acceptable argumentation line iff: (1) The sets $\lambda_S$ and $\lambda_I$ are each one non-contradictory sets.
of arguments; (2) No argument \( \langle A_k, q_k \rangle \) in \( \lambda \) is a sub-argument of an earlier argument \( \langle A_i, q_i \rangle \) of \( \lambda (i < k) \).

With this conditions averting undesirable situations, an acceptable dialectical tree can defined as tree where every argumentation line is acceptable. Thus, the notion of justification can be defined as follows:

**Definition 2.9. (Justification)** Let \( A \) be an argument for a literal \( q \), and let \( T_{(A,q)} \) be its associated acceptable dialectical tree. The argument \( A \) for \( q \) will be a justification if and only if the root of \( T_{(A,q)} \) is a U-node.

### 3 Dialectical Bases

Next, we will define the concept of dialectical bases which will be used as a basis for extending the DeLP framework by incorporating an argument-based maintenance system to record the program justifications.

As suggested in the introduction, a dialectical base will act as a repository of the justifications computed in the past, containing the dialectical trees obtained as answers to previous queries. This will allow the system to save work when it is faced with a query that has already been solved. Formally, a dialectical base can be defined as follows:

**Definition 3.1. (Dialectical Base)** Let \( (\Pi, \Delta) \) be a defeasible logic program and let \( T_{(A_i,q_i)} \) denote an acceptable dialectical tree. We will say that the finite set \( B \) of dialectical trees

\[
B = \{ T_{(A_1,q_1)}, T_{(A_2,q_2)}, \ldots, T_{(A_n,q_n)} \}
\]

is a dialectical base supported by \( (\Pi, \Delta) \) (denoted \( B_{(\Pi,\Delta)} \)) if and only if for every \( T_{(A_i,q_i)} \in B_{(\Pi,\Delta)}, 1 \leq i \leq n \), it holds that \( T_{(A_i,q_i)} \) is an acceptable dialectical tree which can be obtained from \( (\Pi, \Delta) \) applying the DeLP inference procedure, and \( \langle A_i, q_i \rangle \) is a justification.

A reasonable strategy for constructing a dialectical base \( B_{(\Pi,\Delta)} \) consists in adding justifications to \( B_{(\Pi,\Delta)} \) as the system obtains new justifications to the queries performed by the user. Accordingly, we will assume that before any interaction with the user takes place, the dialectical base \( B_{(\Pi,\Delta)} \) will be empty. When a query \( q \) is successfully solved, obtaining its associated dialectical tree \( T_{(A,q)} \), the current dialectical base will be extended into

\[
B'_{(\Pi,\Delta)} = B_{(\Pi,\Delta)} \cup \{ T_{(A,q)} \}
\]

This process of adding dialectical trees to the dialectical base \( B_{(\Pi,\Delta)} \) will be performed every time a new successful query is solved.

**Using the dialectical base for solving queries**

When the system is given a query \( q \) that has been solved in the past, the dialectical base \( B_{(\Pi,\Delta)} \) can be used to speed up the inference process. All the system has to do is check if there is a tree in \( B_{(\Pi,\Delta)} \) that can be used to justify \( q \), in this case \( q \) will be justified. In contrast, if \( q \) cannot be justified from \( B_{(\Pi,\Delta)} \) then the usual justification procedure will be started, looking for a dialectical tree from \( (\Pi, \Delta) \) whose root be a justified argument for \( q \). This analysis leads us to define when a literal \( q \) is justified with respect to a dialectical base, \( B_{(\Pi,\Delta)} \). Formally stated:

**Definition 3.2. (Dialectical base justification)** Let \( B_{(\Pi,\Delta)} \) be a dialectical base, and let \( q \) be a ground literal. We will say that \( B_{(\Pi,\Delta)} \) justifies \( q \) if and only if there is a dialectical tree \( T_{(A,q)} \) in \( B_{(\Pi,\Delta)} \).
Algorithm 3.1. SolveQuery

input: $q$ (a query)
output: $(A,q)$ (a justified argument for $q$, if any)

if $B_{(\Pi,\Delta)}$ justifies $q$
then returns $(A,q)$
else if there exists a justified argument $(A,q)$ wrt $(\Pi,\Delta)$
then returns $(A,q)$
else the query can not be justified

Figure 1: Algorithm for the SolveQuery process

Next, we will define an extended DeLP framework, which incorporates a dialectical base to improve the efficiency the query solving process.

Extending the DeLP framework with a dialectical base

The proposed extension to the DeLP system can be defined as follows:

Definition 3.3. (Extended DeLP framework) Let $(\Pi,\Delta)$ be a defeasible logic program. An extended DeLP framework is composed by:

- An extended knowledge base $((\Pi,\Delta), B_{(\Pi,\Delta)})$ where $B_{(\Pi,\Delta)}$ corresponds to a dialectical base supported by $(\Pi,\Delta)$.
- An extended query solving process which acts according to the algorithm 3.1.

The DeLP extended system recently defined has the ability to record the justifications computed in the past. This will speed up the system’s response time when faced with a previously solved query, but it will also spring some new problems that must be taken into account. We will analyze this situations in the next section.

4 Revision in Dialectical Bases

It is desirable that a system modeling the behavior of an intelligent agent be capable of acquiring new information in a dynamic way. Therefore, we should take into account a mechanism to add a new fact $h$ to the strict knowledge set $\Pi$, resulting in a new knowledge base $(\Pi',\Delta)$. Since in a DeLP program the set of strong knowledge must be consistent, the set $\Pi \cup \{h\}$, resulting of adding the new fact, must be also consistent\(^2\).

Nevertheless, when a new fact $h$ is added to the non-defeasible knowledge $\Pi$, the justifications obtained in the past stored in $B_{(\Pi,\Delta)}$ may no longer be valid with respect to $\Pi \cup \{h\}$.

\(^2\)From now on we will assume that adding a new fact $h$ to $\Pi$ results in a set $\Pi \cup \{h\}$ such that $\Pi \cup \{h\} \not\models \bot$. The situation when the fact to add is inconsistent with the information stored in the knowledge base forces us to take into account subtler questions and, for the time being, stills under investigation.
To overcome this problem a revision process should be performed every time a new fact is added, in order to keep the dialectical base properly updated.

To this purpose, we will define an update procedure, based on actualizing each dialectical tree present in the dialectical base with respect to \( h \). To state the update procedure, we must be aware of the changes that can arise from the adding of a new fact.

Given a dialectical tree \( \mathcal{T}_{(A,q)} \) in \( B_{(\Pi,\Delta)} \) and a new fact \( h \) to be introduced into the knowledge base \( \Pi \), we will distinguish two kinds of problematic situations that can result in the invalidation of \( \mathcal{T}_{(A,q)} \):

- Argument invalidation
- Dialectical tree invalidation

The aforementioned update procedure must be aware of both of these issues. In what follows we will discuss each one in a more detailed way.

### 4.1 Argument Invalidation

#### Consistency

When we add a new fact \( h \) to \( \Pi \), some of the arguments supported by the old knowledge base may not be consistent with respect to the new knowledge base, \( \Pi \cup \{ h \} \), and therefore they do not constitute arguments with respect to \( \Pi \cup \{ h \} \).

Thus, after we add a new fact \( h \) to the knowledge base \( \Pi \) we should verify the consistency of every argument \( \langle A_i, q_i \rangle \) belonging to every \( \mathcal{T}_{(A,q)} \) in \( B_{(\Pi,\Delta)} \), with respect to the new knowledge base. With this object, we will define a function \( \text{Consistent} \), based on the procedure proposed in [8], which will accomplish this task.

**Definition 4.1.** Let \( (\Pi, \Delta) \) be a knowledge base, let \( A \) be an argument for \( q \) built from \( (\Pi, \Delta) \) and let \( h \) be a new fact added to \( \Pi \), such that \( \Pi \cup \{ h \} \not\vdash \bot \).

We define the function \( \text{Consistent}(A, h) : \varphi(\Delta_{grounded}) \times \text{facts}(\mathcal{L}) \mapsto \{\text{true, false}\} \) as follows:

\[
\text{Consistent}(A, h) = \begin{cases} 
\text{true} & \text{if } \Pi \cup \{ h \} \cup \text{Heads}(A) \not\vdash \bot \\
\text{false} & \text{otherwise}
\end{cases}
\]

where \( \text{Heads}(A) \) denotes the consequents of the rules in \( A \), and \( \text{facts}(\mathcal{L}) \) denotes all the possible facts in the underlying knowledge representation language, \( \mathcal{L} \).

Note that, if there is an argument \( \langle A_i, q_i \rangle \) in some dialectical tree \( \mathcal{T}_{(A,q)} \) which is not consistent with respect to the new knowledge base we must eliminate not only \( \langle A_i, q_i \rangle \) from \( \mathcal{T}_{(A,q)} \), but also the sub-tree rooted in \( \langle A_i, q_i \rangle \). It must also be remarked that when a sub-tree is eliminated we need to perform some operation in order to insure the correctness of the resulting dialectical tree. To illustrate this fact, let us consider the following example.

**Example 4.1.** Suppose that \( B_{(\Pi,\Delta)} \) is a dialectical base supported by \( (\Pi, \Delta) \), where:

- \( \Pi = \{ d \leftarrow \text{true}, c \leftarrow h \} \)
- \( \Delta = \{ c \leftarrow \text{true}, b \leftarrow \text{true}, a \leftarrow b, \neg b \leftarrow c, \neg c \leftarrow d \} \)

\[3\text{For further details about how to implement this function, we refer the reader to [8].}\]
and suppose that $\mathcal{T}_{(A_1,a)}$ is a dialectical tree stored in $\mathcal{B}_{(\Pi,\Delta)}$ composed by the arguments:

- $\langle A_1, a \rangle$, where $A_1 = \{b \leftarrow \text{true}, a \leftarrow b\}$,
- $\langle A_2, \neg b \rangle$, where $A_2 = \{c \leftarrow \text{true}, \neg b \leftarrow c\}$
- $\langle A_3, \neg c \rangle$, where $A_3 = \{\neg c \leftarrow d\}$.

In this situation, $\langle A_1, a \rangle$ is the root of the tree, $\langle A_2, \neg b \rangle$ is a defeater for $\langle A_1, a \rangle$ and $\langle A_3, \neg c \rangle$ is a defeater for $\langle A_2, \neg b \rangle$.

The dialectical tree $\mathcal{T}_{(A_1,a)}$ is justified because $\langle A_2, \neg b \rangle$ (the defeater for $\langle A_1, a \rangle$), is in turn defeated by $\langle A_3, \neg c \rangle$ and as a result $\langle A_1, a \rangle$ is justified. However, if we add a new fact $h$ to $\Pi$, then $\langle A_3, \neg c \rangle$ will not be an argument with respect to $(\Pi \cup \{h\}, \Delta)$, because $\Pi \cup \{h\} \cup \text{Heads}(A_3) \vdash \bot$. Therefore, $\langle A_2, \neg b \rangle$ is now undefeated, and hence $\langle A_1, a \rangle$ is defeated.

The previous example shows that eliminating the arguments which are not consistent with respect to the new knowledge base, involves also updating the marking of the resulting dialectical tree. It goes without saying that the procedure for updating each tree must be aware of this fact.

\section*{Minimality}

An argument must be minimal with respect to the set of rules used in its construction [15]. However, when we add a new fact $h$ to the knowledge base, the arguments stored in $\mathcal{B}_{(\Pi,\Delta)}$ may no longer satisfy this constrain with respect to $(\Pi \cup \{h\}, \Delta)$. In fact, it may happen that for some argument $\langle A, q \rangle$ in $\mathcal{B}_{(\Pi,\Delta)}$, we could obtain a new argument $\langle A', q \rangle$ with respect to $(\Pi \cup \{h\}, \Delta)$, such that $A' \subset A$ and $\Pi \cup \{h\} \cup A' \models q$. In this case, the argument $\langle A', q \rangle$ based on $\Pi \cup \{h\}$ will be called a \textit{reduction} of the argument $\langle A, q \rangle$ based on $\Pi$. Formally:

\begin{definition}
Reduction. Proper reduction\end{definition}

An argument $\langle A_1, q \rangle$ based on the set of strong knowledge $\Pi \cup \{h\}$ will be called a \textit{reduction} of the argument $\langle A_2, q \rangle$ based on $\Pi$, if $A_1 \subseteq A_2$. If $A_1 \subset A_2$, then $\langle A_1, q \rangle$ will be a \textit{proper reduction} of $\langle A_2, q \rangle$.

Consequently, after we add a new fact to $\Pi$ we must check the minimality of the previously computed arguments. We can do this in the following way: we find a reduction $A'$ for every argument $A$ present in $\mathcal{B}_{(\Pi,\Delta)}$ with respect to the new knowledge base. If $A' = A$ then $A$ is still minimal. Otherwise, $A$ is clearly non-minimal and therefore it must be replaced by its reduction. For this reason, we need to define a function which maps every argument into its corresponding reduction with respect to a fact $h$. Formally stated:

\begin{definition}
Let $\langle A, q \rangle$ be an argument supported by $(\Pi, \Delta)$, and let $(\Pi \cup \{h\}, \Delta)$ be the new knowledge base. We will define the function

\begin{equation*}
\text{FindReduction}((A, q), h) : \varphi(\text{Args}(\Pi, \Delta)) \times \text{facts}(\mathcal{L}) \mapsto \varphi(\text{Args}(\Pi, \Delta))
\end{equation*}

which given an argument $\langle A, q \rangle$ and an added fact $h$ returns the reduction of $\langle A, q \rangle$ with respect to $h$.
\end{definition}

\footnote{We will overload the term “justified” by saying that a dialectical tree is justified iff its root is a justification [14].}
The function \textit{FindReduction} can be implemented eliminating the defeasible rules in \(A\) which are no longer needed to conclude \(h\) [8]. Note that \(\text{FindReduction}(\langle A, q \rangle, h)\) always exists, but in some cases (for instance when the new added fact is \(q\)) it can result in the empty argument.

Then, after adding a new fact to \(\Pi\), we must apply the \textit{FindReduction} function to every argument \(\langle A_i, q_i \rangle\) present in \(B_{(\Pi, \Delta)}\). If we find that for some argument \(\langle A_i, q_i \rangle\) in some \(T_{(A,q)}\), supported by the old knowledge base \((\Pi, \Delta)\), \(\text{FindReduction}(\langle A_i, q_i \rangle, h) \neq \langle A_i, q_i \rangle\), then we must replace \(\langle A_i, q_i \rangle\) with its corresponding reduction with respect to the new added fact \(h\), that is \(\text{FindReduction}(\langle A_i, q_i \rangle, h)\).

It is important to mention that when we replace a stored argument \(\langle A_i, q_i \rangle\) from a dialectical tree \(T_{(A,q)}\), with an argument \(\langle A'_i, q_i \rangle\), where \(\text{FindReduction}(\langle A_i, q_i \rangle, h) = \langle A'_i, q_i \rangle\), we should apply some procedure on \(T_{(A,q)}\) in order to insure the correctness of the resulting dialectical tree. The following example illustrates this fact.

**Example 4.2.** Let \(T_{(A_1, \neg a)}\) be a dialectical tree stored in \(B_{(\Pi, \Delta)}\), where:

- \(\Pi = \{ d \leftarrow \text{true}, e \leftarrow \text{true} \}\)
- \(\Delta = \{ \neg a \leftarrow d, a \leftarrow h, h \leftarrow c, e \leftarrow \text{true}, \neg e \leftarrow e \}\)

and \(T_{(A_1, \neg a)}\) is composed by the following arguments:

- \(\langle A_1, \neg a \rangle\), where \(A_1 = \{ \neg a \leftarrow d \}\)
- \(\langle A_2, a \rangle\), where \(A_2 = \{ a \leftarrow h, h \leftarrow c, c \leftarrow \text{true} \}\)
- \(\langle A_3, \neg c \rangle\), where \(A_3 = \{ \neg c \leftarrow e \}\)

In \(T_{(A_1, \neg a)}\), the argument \(\langle A_1, \neg a \rangle\) is the root of the tree, \(\langle A_2, a \rangle\) is a defeater for \(\langle A_1, \neg a \rangle\) and \(\langle A_3, \neg c \rangle\) is a defeater for \(\langle A_2, a \rangle\). Clearly \(\langle A_1, \neg a \rangle\) is a justification with respect to \((\Pi, \Delta)\).

If the fact \(h\) is added to \(\Pi\), the argument \(\langle A_2, a \rangle\) will no longer be minimal and as a consequence, we must replace it with its corresponding reduction with respect to \(h\), \(\langle A'_2, a \rangle = \{ a \leftarrow h \}\). Note that when we replace \(\langle A_2, a \rangle\) with \(\langle A'_2, a \rangle\) in \(T_{(A_1, \neg a)}\), \(\langle A_3, \neg c \rangle\) (which was a defeater for \(\langle A_2, a \rangle\)) is not a defeater for \(\langle A'_2, a \rangle\), and as a result \(\langle A'_2, a \rangle\) is now undefeated with respect to \((\Pi \cup \{ h \}, \Delta)\). Thus \(\langle A_1, \neg a \rangle\) becomes a defeated argument.

### 4.2 Dialectical Tree Invalidation

After we add a new fact \(h\) to \(\Pi\), a dialectical tree \(T_{(A,q)}\) stored in a dialectical base \(B_{(\Pi, \Delta)}\) may no longer remain valid, even if the arguments in \(T_{(A,q)}\) remain consistent and minimal. This situation obeys to one of the following causes: invalidation of some of the argumentation lines in \(T_{(A,q)}\), or generation of conflicting arguments using the new fact. In what follows we analyze and propose solutions to these situations, that will be taken into account when designing the \textit{Update} procedure.
**Argumentative line invalidation**

For a dialectical tree to be acceptable, it must be composed by valid arguments, and all its argumentation lines have to be acceptable [14]. Recall that for an argumentation line to be acceptable, it must, among other things, be composed by concordant arguments (as stated in section ??). When we add a new fact to $\Pi$, some of the acceptable argumentation lines may become unacceptable, because some of its arguments may not remain concordant among themselves.

**Example 4.3.** Consider the following situation: let $T_{(A_1, \neg a)}$ be a dialectical tree stored in a dialectical base $B_{(\Pi, \Delta)}$, where:

- $\Pi = \{d \leftarrow \text{true}, a \leftarrow \neg c, h\}$
- $\Delta = \{b \leftarrow \text{true}, c \leftarrow \text{true}, \neg a \leftarrow b, \neg b \leftarrow c, \neg c \leftarrow d\}$

and $T_{(A_1, \neg a)}$ is composed by three arguments:

- $\langle A_1, \neg a \rangle$, where $A_1 = \{b \leftarrow \text{true}, \neg a \leftarrow b\}$,
- $\langle A_2, \neg b \rangle$, where $A_2 = \{c \leftarrow \text{true}, \neg b \leftarrow c\}$
- $\langle A_3, \neg c \rangle$, where $A_3 = \{\neg c \leftarrow d\}$.

In this example, the argument $\langle A_3, \neg c \rangle$ defeats $\langle A_2, \neg b \rangle$ which in turn defeats the root, $\langle A_1, \neg a \rangle$. Note that in this situation $\langle A_3, \neg c \rangle$ is undefeated, and thus $\langle A_2, \neg b \rangle$ is defeated and $\langle A_1, \neg a \rangle$ is undefeated (justified).

In this setting, if we add a new fact $h$ to $\Pi$, the argumentation line $\lambda = \{\langle A_1, \neg a \rangle, \langle A_2, \neg b \rangle, \langle A_3, \neg c \rangle\}$ in $T_{(A_1, \neg a)}$ becomes unacceptable, since the supporting arguments in $\lambda$, $\langle A_1, \neg a \rangle$ and $\langle A_3, \neg c \rangle$, are not concordant (i.e., $\Pi \cup A_1 \cup A_3 \not\models \bot$). As a result $T_{(A_1, \neg a)}$ is no longer an acceptable dialectical tree.

To overcome this problem, when a new fact is introduced each argumentation line in a dialectical tree we will have to be tested for acceptability. To accomplish this task, every dialectical tree, $T_{(A, q)}$, in $B_{(\Pi, \Delta)}$ must be traversed using a top-down approach. For each argument $\langle A_i, q_i \rangle$ in $T_{(A, q)}$, its consistency should be checked with respect to the sequence $\lambda = \{\langle A, q \rangle, \langle A_1, q_1 \rangle, \ldots, \langle A_r, q_r \rangle\}$ (composed by the arguments that form the path from the root to $\langle A_i, q_i \rangle$)\(^5\), in the following way:

- if $\langle A_i, q_i \rangle$ is a supporting argument, then it should hold that $\Pi \cup \{h\} \cup \lambda_S \not\models \bot$, where $\lambda_S$ denotes the set of supporting arguments present in $\lambda$.
- if $\langle A_i, q_i \rangle$ is an interference argument, then it should hold that $\Pi \cup \{h\} \cup \lambda_I \not\models \bot$, where $\lambda_I$ denotes the set of interference arguments present in $\lambda$.

If we find that $\langle A_i, q_i \rangle$ is not consistent with its argumentation line, then we must eliminate the sub-tree rooted in $\langle A_i, q_i \rangle$ from $T_{(A, q)}$. Note that, in this case the situation is analogous to the consistency case. Then, when a sub-tree is eliminated we need to perform some operation in order to insure the correctness of the resulting dialectical tree.

\(^5\)Note that $\lambda$ represents the path from the root to $\langle A_i, q_i \rangle$ and it is present in all the argumentation lines which include $\langle A_i, q_i \rangle$. 

New conflicting arguments

The addition of a new fact $h$ into $(\Pi, \Delta)$ can spring new, previously unavailable arguments which may be defeaters for some of the arguments present in a dialectical tree $T_{(A,q)}$ stored in $B_{(\Pi,\Delta)}$. This situation may lead to changes into $T_{(A,q)}$, and as matter of fact, it may no longer be a justified dialectical tree.

Example 4.4. Let us consider the following situation. Suppose that $T_{(A_1,\neg a)}$ is a dialectical tree stored in $B_{(\Pi,\Delta)}$, where:

- $\Pi = \{\}$
- $\Delta = \{b \prec \text{true}, c \prec \text{true}, d \prec \text{true}, \neg a \prec b, \neg b \prec c, \neg c \prec d, \neg d \prec h\}$

and $T_{(A_1,\neg a)}$ is composed by the following three arguments:

- $\langle A_1, \neg a \rangle$, where $A_1 = \{b \prec \text{true, } \neg a \prec b\}$
- $\langle A_2, \neg b \rangle$, where $A_2 = \{c \prec \text{true, } \neg b \prec c\}$
- $\langle A_3, \neg c \rangle$, where $A_3 = \{d \prec \text{true, } \neg c \prec d\}$

In $T_{(A_1,\neg a)}$ the argument $\langle A_3, \neg c \rangle$ defeats $\langle A_2, \neg b \rangle$ which in turn defeats the root, $\langle A_1, \neg a \rangle$. It goes without saying that $\langle A_3, \neg c \rangle$ and $\langle A_1, \neg a \rangle$ are undefeated and $\langle A_2, \neg b \rangle$ is defeated by $\langle A_3, \neg c \rangle$.

Suppose that the fact $h$ is added into $\Pi$. Consequently, a new argument $\langle A_4, \neg d \rangle = \{\neg d \prec h\}$ can now be generated, and $\langle A_4, \neg d \rangle$ defeats $\langle A_3, \neg c \rangle$. Accordingly, in the dialectical tree constructed with respect to $(\Pi \cup \{h\}, \Delta)$, $\langle A_2, \neg b \rangle$ is now undefeated and defeats the root of the tree, $\langle A_1, \neg a \rangle$. Therefore $T_{(A_1,\neg a)}$ is no longer a justified dialectical tree.

From the previous example, it is clear that the procedure for updating each dialectical tree stored in $B_{(\Pi,\Delta)}$ should be able to deal with the generation of new defeaters, whose introduction may change the marking of the tree.

5 The Update Procedure

Having outlined the main issues to take into account, we can specify the procedure for updating the dialectical base after adding a new fact into $\Pi$. As we stated before, this procedure will be based on updating each dialectical tree in $B_{(\Pi,\Delta)}$ with respect to the introduced fact $h$. Then, we will also define an UpdateTree procedure, that will update a dialectical tree $T_{(A_i,q_i)}$ with respect to an added fact $h$.

In practice, the Update procedure can be optimized by checking whether the fact $h$ can affect the dialectical base $B_{(\Pi,\Delta)}$ (this should be done before executing the UpdateTree procedure on each dialectical tree $T_{(A_i,q_i)}$ in $B_{(\Pi,\Delta)}$). If $h$ does not affect $B_{(\Pi,\Delta)}$, then the dialectical base will remain the same, i.e., $\text{Update}(B_{(\Pi,\Delta)}, h) = B_{(\Pi,\Delta)}$. On the contrary, if $h$ does affect $B_{(\Pi,\Delta)}$, the UpdateTree procedure will be started on each one of the dialectical trees stored in $B_{(\Pi,\Delta)}$, in order to update them with respect to $h$.

In consequence, the Update procedure should be able to determine when a fact $h$ does not affect a dialectical base $B_{(\Pi,\Delta)}$. There are two special cases where $h$ can not affect $B_{(\Pi,\Delta)}$ which can be easily verified. Namely:
Algorithm 5.1. Update

input: $h$, $B_{(\Pi,\Delta)}$ (a dialectical base)
output: $B'_{(\Pi,\Delta)}$ (the update of $B_{(\Pi,\Delta)}$ wrt $h$)

if there is no rule $r \in \Pi \cup \Delta$ such that $\text{body}(r)$ unifies with $h$ or $\Pi \vdash h$.

$B'_{(\Pi,\Delta)} \leftarrow B_{(\Pi,\Delta)}$

else

$B'_{(\Pi,\Delta)} \leftarrow \emptyset$

for every $T_{(A,\alpha)}$ in $B_{(\Pi,\Delta)}$

$B'_{(\Pi,\Delta)} \leftarrow B'_{(\Pi,\Delta)} \cup \text{UpdateTree}(T_{(A,\alpha)}, h)$

Figure 2: Algorithm for the Update procedure

1. There is no rule $r$ in $\Pi \cup \Delta$ such that the body of $r$ unifies with $h$.

2. The set of strong knowledge deduces the fact to be added, i.e., $\Pi \not\vdash h$.

According to this, the first thing the Update procedure does is to check if one of these two conditions holds. In this case, there will not be any change in the dialectical base. In spite of this, if none of the conditions is verified, the procedure will update every dialectical tree in $B_{(\Pi,\Delta)}$ by means of the UpdateTree procedure (whose full description is stated below). The complete algorithm for the Update procedure is shown in figure 2.

5.1 The UpdateTree Procedure

In the previous section, we presented the algorithm for the Update procedure, which uses the UpdateTree procedure. In the present section we will state the issues that the UpdateTree procedure has to take into account for correctly updating a dialectical tree. Having in mind these issues, we will define the UpdateTree procedure and discuss its main aspects.

To fulfill its purpose, the UpdateTree procedure has to deal with all the issues discussed in the section 4, namely:

- Argument Invalidation: this involves eliminating or replacing the old arguments which are not consistent or minimal with respect to the knowledge base resulting after adding the new fact.

- Dialectical Tree Invalidation: this involves eliminating the arguments which are not concordant with respect to its argumentation line and adding the defeaters generated using the new fact.

Besides, the procedure must also update the marking of the dialectical tree, according to the changes introduced in it.

The algorithm for the UpdateTree procedure is shown in figure 3. It uses the UpdateArg procedure, which calls recursively to itself and updates each argument in $T_{(A,\alpha)}$ (and its marking) with respect to the added fact.

But since the UpdateArg procedure needs to check the concordance of the argumentation lines in the dialectical tree, $T_{(A,\alpha)}$, it needs to know the arguments presents in the path from
Algorithm 5.2. UpdateTree

input: \( h, T(A,q) \) (a dialectical tree)
output: \( S \) (a singleton containing the updating of \( T(A,q) \) or \( \emptyset \))

\[
\begin{align*}
SL & \leftarrow \emptyset \\
IL & \leftarrow \emptyset \\
T(A',q) & \leftarrow \text{UpdateArg}(h, SL, IL, \langle A, q \rangle) \\
\text{if } T(A',q) & \text{ is undefeated} \\
S & \leftarrow \{T(A',q)\} \\
\text{else} \\
S & \leftarrow \emptyset
\end{align*}
\]

Figure 3: Algorithm for the UpdateTree procedure

the root of the tree to the argument being considered (as specified in section 4). Then, it will be able to check if the argument is concordant with its argumentation line. For this reason, SL and IL (which represent the supporting and interference arguments that are present in the path from the root to \( \langle A, q \rangle \)) must be provided to the UpdateArg procedure.

Note that, as mentioned before, the labeling of a dialectical tree \( T(A,q) \) may change after updating it with respect to an added fact, and as a result it can no longer be a justified tree. In this case, the UpdateTree procedure eliminates \( T(A',q) \) (the update of \( T(A,q) \)) from the dialectical base, and returns the empty set\(^6\).

5.2 The UpdateArg Procedure

In this section, we will analyze the UpdateArg procedure, which is the core of the Update procedure. Briefly stated, the UpdateArg procedure has to accomplish the following tasks:

- Eliminate the old arguments which are not consistent wrt \( (\Pi \cup \{h\}, \Delta) \).
- Replace each argument which is non-minimal wrt \( (\Pi \cup \{h\}, \Delta) \).
- Eliminate the arguments which are not concordant wrt its argumentation line.
- Add to every argument its new defeaters, generated using the added fact.
- Update the marking of each argument.

According to this goals, we designed the UpdateArg algorithm, as depicted in figure 4. In the first place the UpdateArg procedure checks if the argument \( \langle A, q \rangle \) is consistent, using the function defined in section 4. If \( \langle A, q \rangle \) is not consistent, \( (i.e., \text{if } \text{Consistent}(A,h) = \text{false}) \) then the procedure eliminates not only the argument \( \langle A, q \rangle \), but also the subtree rooted in \( \langle A, q \rangle \). On the contrary, if \( \langle A, q \rangle \) is consistent, it must check that \( \langle A, q \rangle \) is concordant with respect to its argumentation line. If it is not, then the procedure eliminates the argument and its associated subtree from the dialectical tree being updated.

\(^6\)Note that UpdateTree returns either a singleton or the empty set. We use set notation just to simplify the specification of the related algorithms
Algorithm 5.3. UpdateArg

\textbf{input:} \(h, SL, IL, \langle A, q \rangle\) (an argument \textit{wrt} to \((\Pi, \Delta)\))

\textbf{output:} \(\langle A', q \rangle\) (the updating of \(A\) \textit{wrt} \(h\))

\begin{itemize}
  \item if (not Consistent \((A, q)\))
  \item or (\((A, q)\) is an interference argument and \(A \cup \Pi \cup IL \not\models \bot\))
  \item or (\((A, q)\) is a supporting argument and \(A \cup \Pi \cup SL \not\models \bot\))
    \begin{itemize}
      \item eliminate the subtree rooted in \(A\)
    \end{itemize}
  \item else
    \begin{itemize}
      \item replace \((A, q)\) with \((A', q) = \text{FindReduction}(\langle A, q \rangle, h)\)
      \item \(DS \leftarrow \text{Defeaters}(\langle A, q \rangle)\)
      \item if \(A' \neq A\)
        \begin{itemize}
          \item for every \((D, r)\) in \(DS\)
            \begin{itemize}
              \item If \((D, r)\) does not defeat \((A', q)\)
                \begin{itemize}
                  \item eliminate the tree rooted in \((D, r)\)
                \end{itemize}
                \(DS \leftarrow DS \setminus \{(D, r)\}\)
              \end{itemize}
        \end{itemize}
      \item for every \((D', r)\) in \(DS\)
        \begin{itemize}
          \item if \((A', q)\) is a supporting argument
            \begin{itemize}
              \item UpdateArg\((\langle D, r \rangle, SL \cup \{A'\}, IL)\)
            \end{itemize}
          \item else
            \begin{itemize}
              \item UpdateArg\((\langle D, r \rangle, SL, IL \cup \{A'\})\)
            \end{itemize}
          \end{itemize}
        \end{itemize}
      \item Build the new defeaters for \((A', q)\) and its associated subtrees
      \item Update the marking of \((A', q)\), according its defeaters’ marking
    \end{itemize}
\end{itemize}

Figure 4: Algorithm for the UpdateArg procedure

If the argument is both consistent and concordant with respect to its argumentation line, the procedure must check \((A, q)\) for minimality. If \((A, q)\) is not minimal it has to be replaced with its corresponding reduction \((A', q)\). In this case the defeaters of \((A, q)\) must be considered one by one, to determine if they are also defeaters for \((A', q)\). If the procedure finds some argument \((D, r)\), which is a defeater for \((A, q)\), but is not a defeater for \((A', q)\), it eliminates the subtree rooted in \((D, r)\) from the dialectical tree. Next, UpdateArg calls recursively to itself for updating the old defeaters and their corresponding subtrees. Once these are updated, the procedure will focus on the generation of new defeaters for \((A', q)\) (and its associated subtrees) using the added fact. To accomplish this task, it uses the procedure specified in [7] for finding the defeaters of a given argument. Finally, it updates the marking of \((A', q)\) according to its defeaters’ marking.

6 Conclusions

We claim that the notion of dialectical bases can improve the performance of existing argument-based frameworks. In this work we have outlined the main issues to consider in order to incorporate a dialectical base into the DeLP framework and proposed solutions to the main problems.

The complete definition of an argumentative setting extended with a justification maintenance system demands taking into account several interacting features, which were discussed
in this work. We have also defined an update procedure which considers all of these features, in order to properly solve the problem of defining a revision operation for dialectical bases.

As a result of the analysis in this work, we can affirm that the formalization of the notion of dialectical bases is the first step in direction towards the definition of a justification maintenance system for argumentative frameworks.

References


