Computing Dialectical Trees Efficiently in Possibilistic Defeasible Logic Programming

Carlos I. Chesñevar[1], Guillermo R. Simari[2], and Lluis Godo[3]

[1] Departament of Computer Science – Universitat de Lleida
C/Jaume II, 69 – 25001 Lleida, SPAIN – EMAIL: `cic@eps.udl.es`
[2] Department of Computer Science and Engineering – Universidad Nacional del Sur
Alem 1253, (8000) Bahía Blanca, ARGENTINA – EMAIL: `grs@cs.uns.edu.ar`
[3] Artificial Intelligence Research Institute (IIIA-CSIC)
Campus UAB - 08193 Bellaterra, Barcelona, SPAIN – EMAIL: `godo@iiia.csic.es`

Abstract. Possibilistic Defeasible Logic Programming (P-DeLP) is a logic programming language which combines features from argumentation theory and logic programming, incorporating as well the treatment of possibilistic uncertainty and fuzzy knowledge at object-language level. Solving a P-DeLP query Q accounts for performing an exhaustive analysis of arguments and defeaters for Q, resulting in a so-called dialectical tree, usually computed in a depth-first fashion. Computing dialectical trees efficiently in P-DeLP is an important issue, as some dialectical trees may be computationally more expensive than others which lead to equivalent results. In this paper we explore different aspects concerning how to speed up dialectical inference in P-DeLP. We introduce definitions which allow to characterize dialectical trees constructively rather than declaratively, identifying relevant features for pruning the associated search space. The resulting approach can be easily generalized to be applied in other argumentation frameworks based in logic programming.

KEY WORDS: Defeasible Argumentation, Logic Programming, Dialectical Reasoning

1 Introduction and motivations

Possibilistic Defeasible Logic Programming (P-DeLP) [1] is a logic programming language which combines features from argumentation theory and logic programming, incorporating as well the treatment of possibilistic uncertainty and fuzzy knowledge at object-language level. As in many argumentation frameworks based in logic programming, solving a P-DeLP query Q accounts for performing an exhaustive analysis of arguments and defeaters for Q, resulting in a so-called dialectical tree, usually computed in a depth-first fashion.

Computing dialectical trees efficiently in P-DeLP is an important issue, as some dialectical trees may be computationally more expensive than others which lead to equivalent results. In this paper we explore different aspects concerning how to speed up dialectical inference in P-DeLP. We introduce definitions which allow to characterize dialectical trees constructively rather than declaratively, identifying relevant features for pruning the associated search space. The resulting approach can be easily generalized to be applied in other argumentation frameworks based in logic programming.

The rest of the paper is structured as follows. Section 2 summarizes the details of P-DeLP. Section 3 discusses how computation of dialectical trees can be modelled in the context of P-DeLP, a characterization extensible to other similar frameworks. Section 4 presents a generic algorithm for computing dialectical trees in a depth-first fashion, as well as some criteria to be considered for pruning the resulting search space. Finally, Section 5 summarizes related work and the main conclusions that have been obtained.

2 The P-DeLP programming language: fundamentals

The P-DeLP language $\mathcal{L}$ is defined from a set of ground fuzzy atoms (fuzzy propositional variables) $\{p, q, \ldots\}$ together with the connectives $\{\sim, \wedge, \leftarrow\}$. The symbol $\sim$ stands for *negation.* A *literal* $L \in \mathcal{L}$ is a ground (fuzzy) atom q or a negated ground (fuzzy) atom $\sim q$, where q is a ground (fuzzy) propositional variable. A *rule* in $\mathcal{L}$ is a formula of the form $Q \leftarrow L_1 \wedge \ldots \wedge L_n$, where $Q, L_1, \ldots, L_n$ are literals in $\mathcal{L}$. When $n = 0$, the formula $Q \leftarrow$ is called a *fact* and simply written as Q. The term *goal* will be used to refer to any literal $Q \in \mathcal{L}$.[4] In the following, capital and lower case letters will denote literals and atoms in $\mathcal{L}$, respectively.

Definition 1 (P-DeLP formulas). *The set Wffs($\mathcal{L}$) of wffs in $\mathcal{L}$ are* facts, rules *and* goals *built over the literals of $\mathcal{L}$. A* certainty-weighted *clause in $\mathcal{L}$, or simply* weighted clause*, is a pair of the form (φ, α), where $\varphi \in$ Wffs($\mathcal{L}$) and $\alpha \in [0, 1]$ expresses a lower bound for the certainty of φ in terms of a necessity measure.*

The original P-DeLP language [1] is based on Possibilistic Gödel Logic or PGL [2], which is able to model both uncertainty and fuzziness and allows for a partial matching mechanism between fuzzy propositional variables. For simplicity and space reasons we will restrict ourselves to fragment of P-DeLP built on non-fuzzy propositions, and hence based on the necessity-valued classical propositional Possibilistic logic [3]. As a consequence, possibilistic models are defined by possibility distributions on the set of classical interpretations [5] and the proof method for our P-DeLP formulas, written $\vdash$, is defined based on the following generalized modus ponens rule (GMP):

$$\frac{\begin{array}{c}(L_0 \leftarrow L_1 \wedge \cdots \wedge L_k, \gamma)\\ (L_1, \beta_1), \ldots, (L_k, \beta_k)\end{array}}{(L_0, \min(\gamma, \beta_1, \ldots, \beta_k))}$$

which is a particular instance of the well-known possibilistic resolution rule, and which provides the *non-fuzzy* fragment of P-DeLP with a complete calculus

[4] Note that conjunction of literals is not a valid goal.

[5] Although the connective $\leftarrow$ in logic programming is different form the material implication, e.g. $p \leftarrow q$ is not the same as $\sim q \leftarrow \sim p$, regarding the possibilistic semantics we assume here they share the same set interpretations.

(1)	$(\sim fuel\_ok \leftarrow pump\_clog, 1)$	(9)	$(oil\_ok \leftarrow pump\_oil, 0.8)$
(2)	$(sw1, 1)$	(10)	$(engine\_ok \leftarrow fuel\_ok \wedge oil\_ok, 0.3)$
(3)	$(sw2, 1)$	(11)	$(\sim engine\_ok \leftarrow fuel\_ok \wedge oil\_ok \wedge heat, 0.95)$
(4)	$(sw3, 1)$	(12)	$(\sim oil\_ok \leftarrow heat, 0.9)$
(5)	$(heat, 1)$	(13)	$(pump\_clog \leftarrow pump\_fuel \wedge low\_speed, 0.7)$
(6)	$(pump\_fuel \leftarrow sw1, 0.6)$	(14)	$(low\_speed \leftarrow sw2, 0.8)$
(7)	$(fuel\_ok \leftarrow pump\_fuel, 0.3)$	(15)	$(\sim low\_speed \leftarrow sw2, sw3, 0.8)$
(8)	$(pump\_oil \leftarrow sw2, 0.8)$	(16)	$(fuel\_ok \leftarrow sw3, 0.9)$

Fig. 1. P-DeLP program $\mathcal{P}_{eng}$ (example 1)

for determining the maximum degree of possibilistic entailment for weighted literals.[6]

In P-DeLP we distinguish between *certain* and *uncertain* clauses. A clause (φ, α) will be referred as certain if $\alpha = 1$ and uncertain, otherwise. Moreover, a set of clauses Γ will be deemed as *contradictory*, denoted $\Gamma \vdash \bot$, if $\Gamma \vdash (q, \alpha)$ and $\Gamma \vdash (\sim q, \beta)$, with $\alpha > 0$ and $\beta > 0$, for some atom q in $\mathcal{L}$.[7] A P-DeLP program is a set of weighted rules and facts in $\mathcal{L}$ in which we distinguish certain from uncertain information. As additional requirement, certain knowledge is required to be non-contradictory. Formally:

Definition 2 (Program). *A P-DeLP program $\mathcal{P}$ (or just program $\mathcal{P}$) is a pair (Π, Δ), where Π is a non-contradictory finite set of certain clauses, and Δ is a finite set of uncertain clauses.*

Example 1. Consider an intelligent agent controlling an engine with three switches $sw1$, $sw2$ and $sw3$. These switches regulate different features of the engine, such as pumping system, speed, etc. The knowledge of such an agent can be modelled by the program $\mathcal{P}_{eng}$ shown in Fig. 1. Note that uncertainty is assessed in terms of different necessity measures. This agent may have the following certain and uncertain knowledge about how this engine works, *e.g.* *"if the pump is clogged, then the engine gets no fuel with necessity measure of 1"* (rule 1) or *"When there is heat, then oil is usually not ok with necessity measure of 0.9"* (rule 12). Suppose also that the agent knows that $sw1$, $sw2$ and $sw3$ are on, and there is *heat* (rules 1-5). The agent wants to determine if the engine is ok on the basis of this program $\mathcal{P}_{eng}$.

Definition 3 (Argument. Subargument). *Given a program $\mathcal{P} = (\Pi, \Delta)$, a set $\mathcal{A} \subseteq \Delta$ of uncertain clauses is an* argument *for a goal Q with necessity degree $\alpha > 0$, denoted $\langle \mathcal{A}, Q, \alpha \rangle$, iff: (1) $\Pi \cup \mathcal{A} \vdash (Q, \alpha)$; (2) $\Pi \cup \mathcal{A}$ is non contradictory; and (3) There is no $\mathcal{A}_1 \subset \mathcal{A}$ such that $\Pi \cup \mathcal{A}_1 \vdash (Q, \beta)$, $\beta > 0$. Let $\langle \mathcal{A}, Q, \alpha \rangle$ and $\langle \mathcal{S}, R, \beta \rangle$ be two arguments. We will say that $\langle \mathcal{S}, R, \beta \rangle$ is a* subargument *of $\langle \mathcal{A}, Q, \alpha \rangle$ iff $\mathcal{S} \subseteq \mathcal{A}$. Notice that the goal R may be a subgoal associated with the goal Q in the argument $\mathcal{A}$.*

Note that from the definition of argument, it follows that on the basis of a P-DeLP program $\mathcal{P}$ there may exist different arguments $\langle \mathcal{A}_1, Q, \alpha_1 \rangle$, $\langle \mathcal{A}_2, Q, \alpha_2 \rangle$,

[6] From now on we will simply use P-DeLP to actually refer to the non-fuzzy fragment of the original P-DeLP.

[7] For a given goal Q, we write $\sim Q$ as an abbreviation to denote "$\sim q$" if $Q \equiv q$ and "q" if $Q \equiv \sim q$.

..., $\langle \mathcal{A}_k, Q, \alpha_k \rangle$ supporting a given goal Q, with (possibly) different necessity degrees $\alpha_1, \alpha_2, \ldots, \alpha_k$. Arguments are built by backward chaining on the basis of the P-DeLP program P. The necessity degree of the conclusion of an argument involving clauses $(C_1, \beta_1), \ldots (C_k, \beta_k)$ is defined as $min(\beta_1, \ldots, \beta_k)$. Consequently, if $\langle \mathcal{S}, R, \beta \rangle$ is a subargument of an argument $\langle \mathcal{A}, Q, \alpha \rangle$, then $\beta \geq \alpha$.

Example 2. Consider theprogram $\mathcal{P}_{eng}$ in Ex 1. The argument $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$ can be obtained, with
$\mathcal{A}_1 = \{(\mathit{engine\_ok} \leftarrow \mathit{fuel\_ok} \wedge \mathit{oil\_ok}, 0.3), (\mathit{pump\_fuel} \leftarrow \mathit{sw1}, 0.6);$
$(\mathit{fuel\_ok} \leftarrow \mathit{pump\_fuel}, 0.3), \{(\mathit{pump\_oil} \leftarrow \mathit{sw2}, 0.8);(\mathit{oil\_ok} \leftarrow \mathit{pump\_oil}, 0.8)\}.$
In particular, the argument $\langle \mathcal{B}, \mathit{fuel\_ok}, 0.3 \rangle$, with $\mathcal{B} = \{(\mathit{pump\_fuel} \leftarrow \mathit{sw1}, 0.6);$ $(\mathit{fuel\_ok} \leftarrow \mathit{pump\_fuel}, 0.3)\}$, is a subargument of $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$.

Conflict among arguments will be formalized by the notions of counterargument and defeat presented next.

Definition 4 (Counterargument). *Let $\mathcal{P}$ be a program, and let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments wrt $\mathcal{P}$. We will say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ counterargues $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ iff there exists a subargument (called* disagreement subargument*) $\langle \mathcal{S}, Q, \beta \rangle$ of $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ such that $\Pi \cup \{(Q_1, \alpha_1), (Q, \beta)\}$ is contradictory. The literal (Q, β) will be called* disagreement literal.

Example 3. Consider the program from Ex 1. Another argument $\langle \mathcal{A}_2, \sim \mathit{fuel\_ok}, 0.6 \rangle$ can be found, with

$$\mathcal{A}_2 = \{ (\sim\mathit{fuel\_ok} \leftarrow \mathit{sw1}, 0.6), (\mathit{low\_speed} \leftarrow \mathit{sw2}, 0.8), (\mathit{pump\_clog} \leftarrow \mathit{pump\_fuel} \wedge \mathit{low\_speed}, 0.7)\}$$

Argument $\langle \mathcal{A}_2, \sim\mathit{fuel\_ok}, 0.6 \rangle$ is a counterargument for $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$ as there exists a subargument $\langle \mathcal{B}, \mathit{fuel\_ok}, 0.3 \rangle$ in $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$ (see Example 2) such that $\Pi \cup \{(\mathit{fuel\_ok}, 0.3), (\sim\mathit{fuel\_ok}, 0.6)\}$ is contradictory.

Defeat among arguments involves a *preference criterion* on conflicting arguments, defined on the basis of necessity measures associated with arguments.

Definition 5 (Defeat). *Let $\mathcal{P}$ be a program, and let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments in $\mathcal{P}$. We will say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is a* defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ iff $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ counterargues argument $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ with disagreement subargument $\langle \mathcal{A}, Q, \alpha \rangle$, with $\alpha_1 \geq \alpha$. If $\alpha_1 > \alpha$ then $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is called a* proper defeater*, otherwise ($\alpha_1 = \alpha$) it is called a* blocking defeater.

Example 4. Consider $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$ and $\langle \mathcal{A}_2, \sim \mathit{fuel\_ok}, 0.6 \rangle$ in Ex. 3. Then $\langle \mathcal{A}_2, \sim \mathit{fuel\_ok}, 0.6 \rangle$ is a proper defeater for $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$, as $\langle \mathcal{A}_2, \sim \mathit{fuel\_ok}, 0.6 \rangle$ counterargues $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$ with disagreement subargument $\langle \mathcal{B}, \mathit{fuel\_ok}, 0.3 \rangle$, and $0.6 > 0.3$.

Definition 6 (Argumentation line). *An* argumentation line λ *starting in an argument $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ is a finite sequence of arguments $[\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle, \langle \mathcal{A}_1, Q_1, \alpha_1 \rangle, \ldots, \langle \mathcal{A}_n, Q_n, \alpha_n \rangle, \ldots]$ such that every $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ defeats $\langle \mathcal{A}_{i-1}, Q_{i-1}, \alpha_{i-1} \rangle$, for $0 < i \leq n$, satisfying certain* dialectical constraints *(see below). Every argument $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ in λ has* level i. *We will distinguish the sets*

$$S^k_\lambda = \bigcup_{i=0,2,\ldots,2\lfloor k/2 \rfloor} \{\langle \mathcal{A}_i, Q_i, \alpha_i \rangle \in \lambda\} \text{ and } I^k_\lambda = \bigcup_{i=1,3,\ldots,2\lfloor k/2 \rfloor+1} \{\langle \mathcal{A}_i, Q_i, \alpha_i \rangle \in \lambda\}$$

associated with even-level (resp. odd-level) arguments in λ up to the k-th level ($k \leq n$).

An argumentation line can be thought of as an exchange of arguments between two parties, a *proponent* (evenly-indexed arguments) and an *opponent* (oddly-indexed arguments). In order to avoid *fallacious* reasoning, argumentation theory imposes additional constraints on such an argument exchange to be considered rationally acceptable wrt a P-DeLP program $\mathcal{P}$, namely:[8]

1. **Non-contradiction:** given an argumentation line λ of length n the set S_λ^n associated with the proponent (resp. I_λ^n for the opponent) should be *non-contradictory* wrt $\mathcal{P}$.[9]
2. **No circular argumentation:** no argument $\langle \mathcal{A}_j, Q_j, \alpha_j \rangle$ in λ is a sub-argument of an argument $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ in λ, $i < j$.
3. **Progressive argumentation:** every blocking defeater $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ in λ is defeated by a proper defeater $\langle \mathcal{A}_{i+1}, Q_{i+1}, \alpha_{i+1} \rangle$ in λ.

An argumentation line that cannot be further extended on the basis of a given program $\mathcal{P}$ will be called *exhaustive*, otherwise it will be *partial.* Formally:

Definition 7 (Partial/exhaustive argumentation line). *Given two argumentation lines λ and λ', we will say that λ'* extends *λ iff λ is an initial subsequence of λ'. An argumentation line λ will be called* exhaustive *iff there is no argumentation line λ' that extends λ; otherwise λ will be called* partial.

As most argumentation systems [5, 6], in order to determine whether a given argument is ultimately undefeated (or *warranted*) wrt a program $\mathcal{P}$, the P-DeLP framework relies on an *exhaustive* dialectical analysis. Such analysis is modelled in terms of a *dialectical tree*,[10] where every path can be seen as an exhaustive argumentation line.

Definition 8 (Dialectical tree). *Let $\mathcal{P}$ be a program, and let $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ be an argument wrt $\mathcal{P}$. A* dialectical tree *for $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$, denoted $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$, is a tree structure defined as follows:*

1. *The root node of $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$ is $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$.*
2. *$\langle \mathcal{B}', H', \beta' \rangle$ is an immediate child of $\langle \mathcal{B}, H, \beta \rangle$ iff there exists an* ***exhaustive*** *argumentation line $\lambda = [\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle, \langle \mathcal{A}_1, Q_1, \alpha_1 \rangle, \ldots, \langle \mathcal{A}_n, Q_n, \alpha_n \rangle, \ldots]$ such that there are two elements $\langle \mathcal{A}_{i+1}, Q_{i+1}, \alpha_{i+1} \rangle = \langle \mathcal{B}', H', \beta' \rangle$ and $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle = \langle \mathcal{B}, H, \beta \rangle$, for some $i = 0 \ldots n-1$.*

[8] These constraints may vary from one particular argumentation framework to another. In particular, parametrizing dialectical trees with constraints on argumentation lines may give rise to characterizations of different logic programming semantics, as shown in [4].

[9] Non-contradiction for a set of arguments is defined as a generalization of Def. 3: a set $S = \bigcup_{i=1}^{n} \{\langle \mathcal{A}_i, Q_i, \alpha_i \rangle\}$ of arguments is *contradictory* wrt $\mathcal{P}$ iff $\Pi \cup \bigcup_{i=1}^{n} \mathcal{A}_i$ is contradictory.

[10] In some frameworks other names are used for denoting tree-like structures of arguments, *e.g.* 'argument tree' or 'dialogue tree' [7, 8].

Example 5. Consider Ex. 1. To compute the dialectical tree for $\langle \mathcal{A}_1, engine\_ok, 0.3\rangle$, the P-DeLP inference engine computes argumentation lines by depth-first search. A defeater for $\langle \mathcal{A}_1, engine\_ok, 0.3\rangle$ is found, namely $\langle \mathcal{A}_2, \sim fuel\_ok, 0.6\rangle$ (Ex. 4). This defeater can on its turn be defeated by a third defeater $\langle \mathcal{A}_3, \sim low\_speed, 0.8\rangle$, with disagreement subargument $\langle \mathcal{A}_2{}', low\_speed, 0.8\rangle$. Note that argument $\langle \mathcal{A}_4, fuel\_ok, 0.9\rangle$, with $\mathcal{A}_4 = \{ (fuel\_ok \leftarrow sw3, 0.9) \}$ would be also a defeater for $\langle \mathcal{A}_2, \sim fuel\_ok, 0.6\rangle$. This completes the analysis of defeaters for $\langle \mathcal{A}_2, \sim fuel\_ok, 0.6\rangle$. Backtracking to argument $\langle \mathcal{A}_1, engine\_ok, 0.3\rangle$, another defeater is found, namely $\langle \mathcal{A}_5, \sim engine\_ok, 0.3\rangle$, with

$$\mathcal{A}_5 = \{ (\sim engine\_ok \leftarrow fuel\_ok \wedge oil\_ok \wedge heat, 0.95) ; (pump\_fuel \leftarrow sw1, 0.6); \\ (fuel\_ok \leftarrow pump\_fuel, 0.3), (pump\_oil \leftarrow sw2, 0.8); (oil\_ok \leftarrow pump\_oil, 0.8)\}.$$

Note that no further arguments can be found in the dialectical analysis. Although $\langle \mathcal{A}_6, \sim oil\_ok, 0.9\rangle$, with $\mathcal{A}_6 = \{ (\sim oil\_ok \leftarrow heat, 0.9) \}$ seems a possible defeater for $\langle \mathcal{A}_5, \sim engine\_ok, 0.3\rangle$, such argument would be *fallacious* as $\mathcal{A}_1$ supports *oil_ok*, and there would be even-level arguments (namely $\mathcal{A}_1$ and $\mathcal{A}_5$) supporting contradictory conclusions. Therefore three different exhaustive argumentation lines can be computed, namely rooted in $\langle \mathcal{A}_1, engine\_ok, 0.3\rangle$, namely:

- $[\langle \mathcal{A}_1, engine\_ok, 0.3\rangle, \langle \mathcal{A}_2, \sim fuel\_ok, 0.6\rangle, \langle \mathcal{A}_3, \sim low\_speed, 0.8\rangle]$
- $[\langle \mathcal{A}_1, engine\_ok, 0.3\rangle, \langle \mathcal{A}_2, \sim fuel\_ok, 0.6\rangle, \langle \mathcal{A}_4, fuel\_ok, 0.9\rangle]$
- $[\langle \mathcal{A}_1, engine\_ok, 0.3\rangle, \langle \mathcal{A}_5, \sim engine\_ok, 0.3\rangle]$

Fig. 2(b) shows the corresponding dialectical tree $\mathcal{T}_{\langle \mathcal{A}_1, engine\_ok, 0.3\rangle}$.

Nodes in a dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0\rangle}$ can be marked as *undefeated* and *defeated* nodes (U-nodes and D-nodes, resp.). A dialectical tree will be marked as an AND-OR tree: all leaves in $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0\rangle}$ will be marked U-nodes (as they have no defeaters), and every inner node is to be marked as *D-node* iff it has at least one U-node as a child, and as *U-node* otherwise. Note that $\alpha - \beta$ pruning (see Fig. 2(a)) can be applied, so not every node in the tree needs to be generated. We will write $Mark(\mathcal{T}_i) = U$ (resp.$Mark(\mathcal{T}_i) = D$) to denote that the root node of $\mathcal{T}_i$ is marked as U-node (resp. D-node).

Definition 9 (Warrant). *An argument $\langle \mathcal{A}_0, Q_0, \alpha_0\rangle$ is ultimately accepted as valid (or* warranted*) with a necessity degree α_0 wrt a program $\mathcal{P}$ iff the root of the tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0\rangle}$ is marked as* U-node *(i.e., $Mark(\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0\rangle}) = U$).*

Example 6. Consider $\mathcal{T}_{\langle \mathcal{A}_1, engine\_ok, 0.3\rangle}$ in Ex. 5. Fig. 2(b) shows the result of computing $Mark(\mathcal{T}_{\langle \mathcal{A}_1, engine\_ok, 0.3\rangle}) = D$ with $\alpha - \beta$ pruning. From Def. 9 it holds that $\langle \mathcal{A}_1, engine\_ok, 0.3\rangle$ is not warranted.

3 Modelling the Computation of Dialectical Trees

P-DeLP –as well as other implemented logic programming approaches to argumentation– relies on *depth-first search* to generate dialectical trees. As discussed before, such search can be improved by applying $\alpha - \beta$ pruning, so that not every node (argument) is computed. A well-known fact in depth-first search is that the *order* in which branches are generated is important. Fig. 2(b) shows a pruned

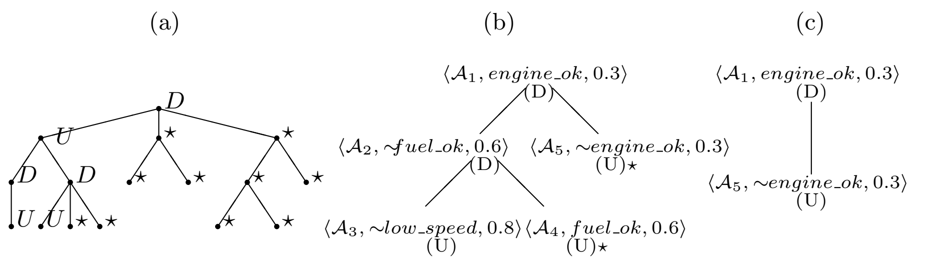

Fig. 2. (a) Dialectical tree, where $\star$'s denote arguments that do not need to be generated because of $\alpha - \beta$ pruning; (b) Dialectical Tree $\mathcal{T}_{\langle \mathcal{A}_1,\, engine\_ok,\, 0.3 \rangle}$ with exhaustive argumentation lines (ex. 5) marked with $\alpha - \beta$ pruning; (c) Optimally settled dialectical tree $\mathcal{T}'$

dialectical tree, where only three arguments were actually computed to deem the root node as defeated. Fig. 2(c) shows that there is an alternative analysis which renders the search space even smaller, by considering first the argument $\langle \mathcal{A}_5, \sim engine\_ok, 0.3 \rangle$ instead of $\langle \mathcal{A}_2, \sim fuel\_ok, 0.6 \rangle$. Such evaluation order for generating argumentation lines is an issue not taken into account in existing formalizations of argumentation frameworks which mostly rely on dialectical trees computed exhaustively. On the other hand, the actual branching factor of a the dialectical tree is clearly restricted by dialectical constraints as discussed in Def. 6. In order to take into account such features we will introduce some new definitions required to characterize dialectical trees *constructively* rather than *declaratively* as follows.

Definition 10 (Dialectical tree (revisited)). *Consider the definition of dialectical tree (as in Def. 8)* without *the restriction of argumentation lines being exhaustive. A dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$ will be called* exhaustive *iff each of its argumentation lines is exhaustive, otherwise $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$ will be called* partial. *We will write $\mathfrak{Tree}_{\mathcal{P}}$ (or just $\mathfrak{Tree}$) to denote the set of all possible dialectical trees based on $\mathcal{P}$.*

In this new setting the process of building a dialectical tree can be thought of as a *computation* starting from an initial tree (consisting of a single node), evolving into more complex trees by adding stepwise new arguments (nodes). This will be formalized by means of a precedence relationship "$\sqsubset$" among trees:

Definition 11 (Precedence relationship $\sqsubset$). *Let $\mathcal{P}$ be a program, and let $\mathcal{T}$, $\mathcal{T}'$ be dialectical trees in $\mathcal{P}$. We define a relationship $\sqsubset \subseteq \mathfrak{Tree} \times \mathfrak{Tree}$, where $\mathcal{T} \sqsubset \mathcal{T}'$ (expressed as $\mathcal{T}'$* evolves from *$\mathcal{T}$) whenever $\mathcal{T}'$ can be obtained from $\mathcal{T}$ by extending some argumentation line in $\mathcal{T}$. We will also write $\mathcal{T} \sqsubset_* \mathcal{T}'$ iff there exists a (possibly empty) sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ such that $\mathcal{T} = \mathcal{T}_1 \sqsubset \ldots \sqsubset \mathcal{T}_k = \mathcal{T}'$.*

Clearly from Defs. 7 and 10 the notion of exhaustive dialectical tree can be recast as follows: A dialectical tree $\mathcal{T}_i$ is exhaustive iff there is no $\mathcal{T}_j \neq \mathcal{T}_i$ such that $\mathcal{T}_i \sqsubset_* \mathcal{T}_j$. In fact, every dialectical tree $\mathcal{T}_i$ can be seen as a 'snapshot' of the status of a disputation between two parties (proponent and opponent), and the relationship "$\sqsubset$" allows to capture the evolution of such disputation. As discussed before, pruning strategies could be applied (*e.g.* $\alpha - \beta$ pruning), allowing to determine whether the marking of the root of a partial tree without computing its associated exhaustive tree. We formalize this situation as follows:

Definition 12 (Settled dialectical tree). *Let $\mathcal{T}_i$ be a dialectical tree, such that for every $\mathcal{T}_j$ evolving from $\mathcal{T}_i$ (i.e., $\mathcal{T}_i \sqsubset_* \mathcal{T}_j$) it holds that $Mark(\mathcal{T}_i) = Mark(\mathcal{T}_j)$. Then $\mathcal{T}_i$ is a* settled dialectical tree. *A $\mathcal{T}_i$ is an* optimally settled dialectical tree *iff there is no $\mathcal{T}_i{}' \sqsubset_* \mathcal{T}_i$ such that $\mathcal{T}_i{}'$ is a settled dialectical tree.*

Example 7. Consider the dialectical trees shown in Fig. 2(b) and (c). Then it holds that $\mathcal{T}\,' \sqsubset_* \mathcal{T}_{\langle \mathcal{A}_1,\, engine\_ok,\, 0.3\rangle}$. Note also that both $\mathcal{T}_{\langle \mathcal{A}_1,\, engine\_ok,\, 0.3\rangle}$ and $\mathcal{T}\,'$ are settled dialectical trees. In particular, $\mathcal{T}\,'$ is an optimally settled dialectical tree.

Note that from the above definition argumentation lines in a settled dialectical tree are not necessarily exhaustive. It is also clear that every exhaustive dialectical tree will be settled, although not necessarily optimally settled. Optimally settled dialectical trees are those involving *the least number of arguments* needed to determine whether the root of the tree is ultimately defeated or not according to the marking procedure.

Proposition 1. *Let $\mathcal{P}$ be a program, and $\langle \mathcal{A}_0, Q_0, \alpha_0\rangle$ an argument in $\mathcal{P}$. Then $\langle \mathcal{A}_0, Q_0, \alpha_0\rangle$ is warranted with necessity degree α_0 iff $Mark(\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0\rangle}) = U$, where $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0\rangle}$ is a settled dialectical tree.*

Next we will analyze how to characterize the computation of dialectical trees in depth-first fashion, modelling informed search oriented towards computing optimally settled dialectical trees. Consider a leaf (argument) $\langle \mathcal{B}, H, \beta\rangle$ in a given argumentation line λ in a partial dialectical tree $\mathcal{T}$ which is not settled, so that further computation will be needed (possibly expanding λ). Clearly, the dialectical constraints given in Def. 6 make that not every defeater as defined in Def. 5 can be used to extend λ. Defeaters satisfying dialectical constraints will be called *feasible defeaters.* Formally:

Definition 13 (Feasible defeaters). *Let $\mathcal{T}_1$ be a partial dialectical tree and let $\langle \mathcal{B}, H, \beta\rangle$ be a leaf node in $\mathcal{T}_1$ at level k in an argumentation line λ. Let $\mathcal{T}$ be the exhaustive dialectical tree associated with $\mathcal{T}_1$ and let $\{\lambda_1, \ldots \lambda_m\}$ be the set of all possible argumentation lines in $\mathcal{T}$ of length $> k+1$ that extend λ, i.e. each λ_i has the form $[\langle \mathcal{A}_0, Q_0, \alpha_0\rangle, \ldots, \langle \mathcal{B}, H, \beta\rangle, \langle \mathcal{B}_i, H_i, \beta_i\rangle, \ldots]$, for $i = 1 \ldots m$. We define the set of* feasible defeaters *for $\langle \mathcal{B}, H, \beta\rangle$ wrt λ as $\mathsf{FDefeat}(\langle \mathcal{B}, H, \beta\rangle, \lambda) = \bigcup_{i=1}^{m}\{\langle \mathcal{B}_i, H_i, \beta_i\rangle\}$.*

Our depth-first approach can thus be improved by restricting search to feasible defeaters. Note that in depth-first search there will be always one current path associated with the last argument introduced. We call that path *current argumentation line.* Clearly, if $\langle \mathcal{B}, H, \beta\rangle$ is a leaf in the current argumentation line λ associated with the computation of a settled dialectical tree $\mathcal{T}$, *any* element in $\mathsf{FDefeat}(\langle \mathcal{B}, H, \beta\rangle, \lambda)$ is a possible candidate for expanding λ. The marking of the tree $\mathcal{T}$ induces an order "$\prec_{eval}$" in $\mathsf{FDefeat}(\langle \mathcal{B}, H, \beta\rangle, \lambda)$: for any two arguments $\langle \mathcal{B}_i, H_i, \beta_i\rangle$, $\langle \mathcal{B}_j, H_j, \beta_j\rangle$ in $\mathsf{FDefeat}(\langle \mathcal{B}, H, \beta\rangle, \lambda)$, we will say that

```
ALGORITHM 1 BuildDialecticalTree
INPUT: $\langle \mathcal{A}, Q, \alpha \rangle$, $\lambda = [\langle \mathcal{A}, Q, \alpha \rangle]$    OUTPUT: $\mathcal{T}_{\langle \mathcal{A}, Q, \alpha \rangle}$, $Mark$ (Marking)
{uses $\alpha$-$\beta$ pruning and evaluation ordering $\preceq_{eval}$}
Global variable: $\mathcal{T}_{\langle \mathcal{A}, Q, \alpha \rangle}$    Local variables: $MarkAux$, ParentDefeated, $\lambda$
{$\lambda$ is the current argumentation line, initially $\lambda = [\langle \mathcal{A}, Q, \alpha \rangle]$}

Put $\langle \mathcal{A}, Q, \alpha \rangle$ as root node of $\mathcal{T}_{\langle \mathcal{A}, Q, \alpha \rangle}$
Compute FDefeat($\langle \mathcal{A}, Q, \alpha \rangle, \lambda$) = $\{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle, \ldots, \langle \mathcal{A}_k, Q_k, \alpha_0 \rangle\}$
{FDefeat($\langle \mathcal{A}, Q, \alpha \rangle, \lambda$)= feasible defeaters for $\langle \mathcal{A}, Q, \alpha \rangle$ wrt $\lambda$}
If $S \neq \emptyset$
  Then
    ParentDefeated := false
    While (ParentDefeated=false) and ($S \neq \emptyset$) do
      Choose some $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle \in S$ minimal wrt $\prec_{eval}$
      $S := S \setminus \{\langle \mathcal{A}_i, Q_i, \alpha_i \rangle\}$
      $\lambda := \lambda \circ \langle \mathcal{A}_i, Q_i, \alpha_i \rangle$ {expand $\lambda$ adding new argument $\langle \mathcal{A}_i, Q_i, \alpha_i \rangle$}
      BuildDialecticalTree($\langle \mathcal{A}_i, Q_i, \alpha_i \rangle, \mathcal{T}_{\langle \mathcal{A}_i, Q_i, \alpha_i \rangle}, MarkAux$)
      Add $\mathcal{T}_{\langle \mathcal{A}_i, Q_i, \alpha_i \rangle}$ as immediate subtree of $\langle \mathcal{A}, Q, \alpha \rangle$.
      If $MarkAux$=$U$ then ParentDefeated := true
    end while
    If ParentDefeated=false          {$S = \emptyset$, all defeaters were defeated}
     then $Mark$ := U {mark $\mathcal{T}_{\langle \mathcal{A}, Q, \alpha \rangle}$ as $U$}
     else $Mark$ := D {mark $\mathcal{T}_{\langle \mathcal{A}, Q, \alpha \rangle}$ as $D$}
  else {$S = \emptyset$, hence $\langle \mathcal{A}, Q, \alpha \rangle$ has no defeaters}
    $Mark$ := U {mark $\mathcal{T}_{\langle \mathcal{A}, Q, \alpha \rangle}$ as $U$}
Return $\mathcal{T}_{\langle \mathcal{A}, Q, \alpha \rangle}$, $Mark$
```

Fig. 3. Algorithm for building and labelling settled dialectical trees in a depth-first fashion taking into account feasible defeaters and evaluation order $\prec_{eval}$

$\langle \mathcal{B}_i, H_i, \beta_i \rangle \prec_{eval} \langle \mathcal{B}_j, H_j, \beta_j \rangle$ if the subtree rooted in $\langle \mathcal{B}_i, H_i, \beta_i \rangle$ is marked before than the subtree rooted in $\langle \mathcal{B}_j, H_j, \beta_j \rangle$. Fig. 3 illustrates how a dialectical tree can be built in a depth-first fashion using $\alpha - \beta$ pruning and the evaluation order $\prec_{eval}$. In order to speed up the construction of a settled dialectical tree, our approach will be twofold: on the one hand, we will identify which literals can be deemed as candidates for computing feasible defeaters. On the other hand, we will provide a definition of $\prec_{eval}$ which prunes the search space using dialectical constraints.

4 Pruning Dialectical Trees in P-DeLP

Given an argument $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$, building a dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$ involves computing defeaters in a recursive way. According to Def. 4, to automate the computation of such defeaters it is necessary to detect the set of *disagreement literals* $\{ (L_1, \phi_1), \ldots, (L_k, \phi_k) \}$ that can be source of conflict with counter-arguments $\langle \mathcal{B}_1, H_1, \beta_1 \rangle, \ldots, \langle \mathcal{B}_k, H_k, \beta_k \rangle$ that defeat $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$. Fortunately, in the context of P-DeLP this can be done on the basis of the *consequents* of uncertain clauses associated with subarguments in $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$.

Definition 14 (Set of consequents Co). *Let $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ be an argument. The set* $\mathbf{Co}(\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle) = \{ (Q, \alpha) \mid \exists\, (Q \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k, \gamma) \in \mathcal{A}_0$ *such that* $\langle \mathcal{A}, Q, \alpha \rangle$ *is a subargument of* $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle \}$. *We generalize this to a set S of arguments, $S = \bigcup_{i=1\ldots k} \langle \mathcal{A}_i, Q_i, \alpha_i \rangle$, defining* $\mathbf{Co}(S) = \bigcup_{i=1\ldots k} \mathbf{Co}(\langle \mathcal{A}_i, Q_i, \alpha_i \rangle)$.[11]

Lemma 1 (Goal-driven defeat [9]). *Let $\langle \mathcal{A}, Q, \alpha \rangle$ be an argument, and let $\langle \mathcal{B}, H, \beta \rangle$ be a defeater for $\langle \mathcal{A}, Q, \alpha \rangle$. Then there exists an argument $\langle \mathcal{B}, H', \beta \rangle$, such that H' is the complement of a literal in* $\mathbf{Co}(\langle \mathcal{A}, Q, \alpha \rangle)$, *(where complement of (L, γ) is defined as $(\sim L, \gamma)$).*

Lemma 1 allows to search for defeaters automatically by backward chaining, on the basis of the consequents of uncertain clauses in an argument. Thus if $(L, \gamma) \in \mathbf{Co}(\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle)$, a search for a defeater with disagreement literal (L, ϕ) will involve finding an argument for concluding $(\sim L, \gamma')$, with $\gamma' \geq \gamma$.

Example 8. Consider $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$ in Ex. 2. Then $\mathbf{Co}(\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle) = \{ (\mathit{engine\_ok}, 0.3), (\mathit{pump\_fuel}, 0.6), (\mathit{fuel\_ok}, 0.3), (\mathit{oil\_ok}, 0.3), (\mathit{pump\_oil}, 0.8) \}$. Defeaters for $\langle \mathcal{A}_1, \mathit{engine\_ok}, 0.3 \rangle$ can be found by backward chaining from the complement of each weighted literal (L, γ), searching for arguments for $(\sim L, \gamma')$, with $\gamma' \geq \gamma$.

From the above considerations we can establish the following inclusionship for detecting candidate disagreement literals in an argument $\langle \mathcal{A}, Q, \alpha \rangle$ appearing as a leaf in an argumentation line λ:

$$\mathbf{Optimal}(\langle \mathcal{A}, Q, \alpha \rangle, \lambda) \subseteq \mathbf{Feasible}(\langle \mathcal{A}, Q, \alpha \rangle, \lambda) \subseteq \mathbf{Co}(\langle \mathcal{A}, Q, \alpha \rangle)$$

Here $\mathbf{Feasible}(\langle \mathcal{A}, Q, \alpha \rangle, \lambda)$ denotes the set of weighted literals (ϕ, α) which are possible disagreement literals for $\langle \mathcal{A}, Q, \alpha \rangle$ for some *feasible* defeater, whereas $\mathbf{Optimal}(\langle \mathcal{A}, Q, \alpha \rangle, \lambda)$ denotes the set of weighted literals (ϕ, α) which are possible disagreement literals for feasible defeaters leading to *the shortest argumentation lines.*[12] Clearly $\mathbf{Optimal}(\langle \mathcal{A}, Q, \alpha \rangle, \lambda)$ is in a sense an *ideal* set of disagreement literals, for which we can find different approximations. One possibility is to consider the set $\mathbf{Feasible}(\langle \mathcal{A}, Q, \alpha \rangle, \lambda)$. However, determining feasible defeaters is computationally also quite a difficult task, as it involves checking different dialectical constraints (see Def. 6). As discussed before, a more tractable way to detect candidate defeaters is to consider the set $\mathbf{Co}(\langle \mathcal{A}, Q, \alpha \rangle)$.

A better approximation than $\mathbf{Co}(\langle \mathcal{A}, Q, \alpha \rangle)$ can be stated taking into account the following intuition: let us assume that the current argumentation line λ has been computed up to level k. From the 'non-contradiction' constraint, even-level as well as odd-level arguments in λ should not be contradictory. This accounts to saying also that literals which are common to *both* even-level and odd-level arguments cannot be disagreement literals within any extension of λ. To formalize this notion, we will suitably extend the definitions for set intersection and difference for weighted literals as follows: given two sets of weighted literals S_1 and S_2, we define intersection among S_1 and S_2 as $S_1 \sqcap S_2 =_{def} \{(Q, \alpha) \mid$

[11] Note that the set $\mathbf{Co}(\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle)$ can be easily computed along the derivation process of the argument itself.

[12] Note that this is a set, as there may be different argumentation lines extending λ, all of them having the same length.

$(Q, \alpha_1) \in S_1$ and $(Q, \alpha_2) \in S_2$, with $\alpha = min(\alpha_1, \alpha_2)$ }. Similarly, we define difference among S_1 and S_2 as follows: $S_1 \setminus S_2 =_{def} \{ (Q, \alpha) \mid (Q, \alpha) \in S_1$ and $\nexists (Q, \beta) \in S_2$, for $\beta > 0 \}$

Definition 15 (Set $\mathsf{SharedLit}$). *Let λ be an argumentation line. $\mathsf{SharedLit}(\lambda, k)$ is the set of weighted literals common to even-level and odd-level arguments up to level k, i.e.* $\mathsf{SharedLit}(\lambda, k) =_{def} \mathbf{Co}(S^k_\lambda) \sqcap \mathbf{Co}(I^k_\lambda)$.

Proposition 2. *Let λ be an argumentation line in a partial dialectical tree $\mathcal{T}$, and let $\langle \mathcal{A}, Q, \alpha \rangle$ an argument which is a leaf in λ at level k. Let $(L, \gamma) \in \mathsf{SharedLit}(\lambda, k), k > 0$. Then $(L, \gamma') \notin \mathbf{Feasible}(\langle \mathcal{A}, Q, \alpha \rangle, \lambda)$, for any $\gamma' \geq \gamma$.*

Proposition 2 allows to further refine the inclusion relationship given before as follows:

$$\mathbf{Feasible}(\langle \mathcal{A}, Q, \alpha \rangle, \lambda) \subseteq \mathbf{Co}(\langle \mathcal{A}, Q, \alpha \rangle) \setminus \mathsf{SharedLit}(\lambda, k) \subseteq \mathbf{Co}(\langle \mathcal{A}, Q, \alpha \rangle)$$

We can now come back to the original question: how to choose which defeater belongs to the (on the average) shorter argumentation line, *i.e.* the one more prone to settle the disputation as soon as possible. From our preceding results we can suggest the following definition for $\prec_{eval}$:

Definition 16 (Evaluation order based on $\mathsf{SharedLit}$). *Let λ be an argumentation line, and let $\langle \mathcal{A}, Q, \alpha \rangle$ be a leaf at level k. Let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two candidate defeaters for $\langle \mathcal{A}, Q, \alpha \rangle$, such that λ can be extended to λ_1 (using $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$) or λ_2 (using $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$). Then $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle \prec_{eval} \langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ iff* $\mathbf{Co}(\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle) \setminus \mathsf{SharedLit}(\lambda_1, k+1) \subseteq \mathbf{Co}(\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle) \setminus \mathsf{SharedLit}(\lambda_2, k+1)$.[13]

Example 9. Consider the argument $\langle \mathcal{A}_1, engine\_ok, 0.3 \rangle$ as in Ex. 2 and 8, and assume that the current argumentation line is $\lambda = [\langle \mathcal{A}_1, engine\_ok, 0.3 \rangle]$. In such a case the set $\mathsf{FDefeat}(\langle \mathcal{A}_1, engine\_ok, 0.3 \rangle, \lambda)$ has two defeaters $\{ \langle \mathcal{A}_2, \sim fuel\_ok, 0.6 \rangle$, $\langle \mathcal{A}_5, \sim engine\_ok, 0.3 \rangle \}$, computed in Ex. 3 and 5. Argumentation line λ can therefore be extended in two different ways, $\lambda_1 = \lambda \circ \langle \mathcal{A}_2, \sim fuel\_ok, 0.6 \rangle$ and $\lambda_2 = \lambda \circ \langle \mathcal{A}_5, \sim engine\_ok, 0.3 \rangle$. Let us compute the set of consequents for these arguments:

$$\begin{aligned}
\mathbf{Co}(\langle \mathcal{A}_1, engine\_ok, 0.3 \rangle) &= \{ (engine\_ok, 0.3), (pump\_fuel, 0.6), \\
&\quad (fuel\_ok, 0.3), (oil\_ok, 0.3), (pump\_oil, 0.8) \}. \\
\mathbf{Co}(\langle \mathcal{A}_5, \sim engine\_ok, 0.3 \rangle) &= \{ (\sim engine\_ok, 0.3), (pump\_fuel, 0.6), (fuel\_ok, 0.3), \\
&\quad (oil\_ok, 0.3), (pump\_oil, 0.8) \}. \\
\mathbf{Co}(\langle \mathcal{A}_2, \sim fuel\_ok, 0.6 \rangle) &= \{ (\sim fuel\_ok, 0.6), (low\_speed, 0.8), (pump\_clog, 0.7) \}
\end{aligned}$$

From the above sets we have then $\mathsf{SharedLit}(\lambda_1, 1) = \emptyset$ and $\mathsf{SharedLit}(\lambda_2, 1) = \{ (engine\_ok, 0.3), (pump\_fuel, 0.6), (fuel\_ok, 0.3), (oil\_ok, 0.3), (pump\_oil, 0.8) \}$. Consequently, it holds that

$$\mathbf{Co}(\langle \mathcal{A}_1, engine\_ok, 0.3 \rangle) \setminus \mathsf{SharedLit}(\lambda_2, 1) \subset \mathbf{Co}(\langle \mathcal{A}_1, engine\_ok, 0.3 \rangle) \setminus \mathsf{SharedLit}(\lambda_1, 1)$$

Thus the defeater $\langle \mathcal{A}_5, \sim engine\_ok, 0.3 \rangle$ should be evaluated before the defeater $\langle \mathcal{A}_2, \sim fuel\_ok, 0.6 \rangle$ in the depth-first computation of the dialectical tree using the algorithm in Fig. 3.

Although Example 9 is rather naïve, it is intended to show one possible way of characterizing the evaluation order $\prec_{eval}$, reducing the average branching factor of the dialectical tree when in a depth-first fashion.

[13] Note that this partial order can be refined by considering those arguments with higher necessity, *i.e.* given two defeaters $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ equally preferred wrt $\prec_{eval}$, the one having $max(\beta_i, \beta_j)$ as necessity degree is preferred.

5 Related work. Conclusions

In this paper we have presented a novel approach to characterize dialectical reasoning in the context of Possibilistic Defeasible Logic Programming, aiming at speeding up the underlying inference procedure. The contribution of this paper is twofold: on the one hand, we have formalized the notion of dialectical trees constructively, taking into account salient features in modelling the depth-first construction of such trees. On the other hand, we have analyzed the role of dialectical constraints as an additional element for pruning the resulting search space. Although our characterization is based in P-DeLP, it can be generalized to be applied in other argumentation frameworks based on logic programming. It must be remarked that P-DeLP is an extension of Defeasible Logic Programming [9], which has been successfully integrated in a number of real-world applications (e.g. clustering [10], and recommender systems [11]).

Our work complements previous research concerning the dynamics of argumentation, notably [12] and [13]. In particular, Prakken [12] has analyzed the exchange of arguments in the context of dynamic disputes. Our approach can also be understood in the light of his characterization of dialectical proof theories. However, Prakken focuses on a comprehensive but rather general framework, in which important computational issues (*e.g.* detecting disagreement literals, search space considerations, etc.) are not taken into account. Hunter [14] analyzes the search space associated with dialectical trees taking into account novel features such as the *resonance* of arguments. His interesting formalization combines a number of features that allow to assess the impact of dialectical trees, contrasting shallow vs. deep trees. However, computational aspects as the ones analyzed in this paper are outside the scope of his work. In [4] a throughout analysis of various argumentation semantics for logic programming is presented on the basis of parametric variations of derivation trees. In contrast with that approach, our aim in this paper was not to characterize different emerging semantics, but rather to focus on an efficient construction of dialectical trees for speeding up inference. On the other hand, in [4] the authors concentrate in normal logic programming, whereas our approach deals with extended logic programming enriched with necessity degrees. Recently semantical aspects of P-DeLP have been analyzed in the context of specialized inference operators [15].

It must be remarked that our approach can also be improved by considering the non-circularity constraint (see Def. 6) for argumentation lines: as the current argumentation line λ is computed, the set of feasible defeaters associated to the last argument in λ at level k is also restricted by arguments which already appeared earlier at any level $k' < k$. Part of our current research work involves how to extend our algorithm to include such non-circularity constraints in our analysis, in order to develop a full-fledged implementation of the algorithm presented in this paper including such features. Our experiments so far have been performed only on a "proof of concept" prototype, as we have not been able yet to carry out thorough evaluations in the context of a real-world application. The results obtained, however, have been satisfactory and as stated before can be generalized to most argumentation frameworks. The development of such generalization is part of our future work.

Acknowledgements This research was partially supported by Projects TIC2003-00950, TIN2004-07933-C03-01/03, by Ramón y Cajal Program (Ministerio de Ciencia y Tecnología, Spain), by CONICET (Argentina), by the Secretaría General de Ciencia y Tecnología de la Universidad Nacional del Sur and by Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 No. 13096).

References

1. Chesñevar, C.I., Simari, G., Alsinet, T., Godo, L.: A Logic Programming Framework for Possibilistic Argumentation with Vague Knowledge. In: Proc. of the Intl. Conf. in Uncertainty in Art. Intelligence. (UAI 2004). Banff, Canada. (2004) 76–84
2. Alsinet, T., Godo, L.: A complete calculus for possibilistic logic programming with fuzzy propositional variables. In: Proc. of the UAI-2000 Conference. (2000) 1–10
3. Dubois, D., Lang, J., Prade, H.: Possibilistic logic. In D.Gabbay, C.Hogger, J.Robinson, eds.: Handbook of Logic in Art. Int. and Logic Prog. (Nonmonotonic Reasoning and Uncertain Reasoning). Oxford Univ. Press (1994) 439–513
4. Kakas, A., Toni, F.: Computing argumentation in logic programming. Journal of Logic Programming **9** (1999) 515:562
5. Chesñevar, C.I., Maguitman, A., Loui, R.: Logical Models of Argument. ACM Computing Surveys **32** (2000) 337–383
6. Prakken, H., Vreeswijk, G.: Logical Systems for Defeasible Argumentation. In Gabbay, D., F.Guenther, eds.: Handbook of Philosophical Logic. Kluwer Academic Publishers (2002) 219–318
7. Besnard, P., Hunter, A.: A logic-based theory of deductive arguments. Artificial Intelligence **1:2** (2001) 203–235
8. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of Applied Non-classical Logics **7** (1997) 25–75
9. García, A., Simari, G.: Defeasible Logic Programming: An Argumentative Approach. Theory and Practice of Logic Programming **4** (2004) 95–138
10. Gómez, S., Chesñevar, C.: A Hybrid Approach to Pattern Classification Using Neural Networks and Defeasible Argumentation. In: Proc. of 17th Intl. FLAIRS Conf. Miami, Florida, USA, AAAI Press (2004) 393–398
11. Chesñevar, C., Maguitman, A.: An Argumentative Approach to Assessing Natural Language Usage based on the Web Corpus. In: Proc. of the ECAI-2004 Conference. Valencia, Spain. (2004) 581–585
12. Prakken, H.: Relating protocols for dynamic dispute with logics for defeasible argumentation. Synthese (special issue on New Perspectives in Dialogical Logic) **127** (2001) 187:219
13. Brewka, G.: Dynamic argument systems: A formal model of argumentation processes based on situation calculus. J. of Logic and Computation **11** (2001) 257–282
14. Hunter, A.: Towards Higher Impact Argumentation. In: Proc. of the 19th American National Conf. on Artificial Intelligence (AAAI'2004), MIT Press (2004) 275–280
15. Chesñevar, C., Simari, G., Godo, L., Alsinet, T.: Argument-based expansion operators in possibilistic defeasible logic programming: Characterization and logical properties. In: 8th European Conf. on Symbolic and Qualitative Aspects of Reasoning Under Uncertainty (ECSQARU 2005), Barcelona, Spain (to appear). (2005)