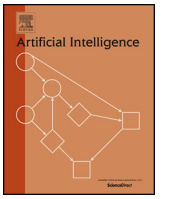
Artificial Intelligence 300 (2021) 103553

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Incremental computation for structured argumentation over dynamic DeLP knowledge bases ☆

Gianvincenzo Alfano [a], Sergio Greco [a], Francesco Parisi [a], Gerardo I. Simari [b,*], Guillermo R. Simari [b]

[a] *Department of Informatics, Modeling, Electronics and System Engineering, University of Calabria, Italy*
[b] *Departamento de Ciencias e Ingenieria de la Computacion, Universidad Nacional del Sur (UNS) & Instituto de Ciencias e Ingenieria de la Computacion (ICIC UNS-CONICET), Argentina*

ARTICLE INFO

Article history:
Received 19 March 2020
Received in revised form 3 May 2021
Accepted 28 June 2021
Available online 5 July 2021

Keywords:
Structured argumentation
Defeasible logic programming
Dynamic DeLP argumentation

ABSTRACT

Structured argumentation systems, and their implementation, represent an important research subject in the area of Knowledge Representation and Reasoning. Structured argumentation advances over abstract argumentation frameworks by providing the internal construction of the arguments that are usually defined by a set of (strict and defeasible) rules. By considering the structure of arguments, it becomes possible to analyze reasons for and against a conclusion, and the *warrant status* of such a claim in the context of a knowledge base represents the main output of a dialectical process. Computing such statuses is a costly process, and any update to the knowledge base could potentially have a huge impact if done naively. In this work, we investigate the case of updates consisting of both additions and removals of pieces of knowledge in the Defeasible Logic Programming (DeLP) framework, first analyzing the complexity of the problem and then identifying conditions under which we can avoid unnecessary computations—central to this is the development of structures (*e.g.* graphs) to keep track of which results can potentially be affected by a given update. We introduce a technique for the incremental computation of the warrant statuses of conclusions in DeLP knowledge bases that evolve due to the application of (sets of) updates. We present the results of a thorough experimental evaluation showing that our incremental approach yields significantly faster running times in practice, as well as overall fewer recomputations, even in the case of sets of updates performed simultaneously.

1. Introduction

Reasoning in argumentation-based systems is primarily carried out by weighing arguments for and against a conclusion or specific query. In structured argumentation, arguments are comprised of derivations that can make use of different kinds of rules and statements. In this paper, we adopt the Defeasible Logic Programming (DeLP) language [2], a logic programming-based approach in which knowledge bases, or programs, are composed of facts and rules, which can be either strict or defeasible.

☆ This paper is a substantially revised and expanded version of [1].

* Corresponding author.
E-mail address: gis@cs.uns.edu.ar (G.I. Simari).

https://doi.org/10.1016/j.artint.2021.103553
0004-3702/

Reasoning in DeLP is based on building a set of so-called *dialectical trees* and then marking their nodes based on their status in the dialogue—the main goal of this process is to arrive at a *warrant status* for a given literal. In general, one can be interested in keeping track of the warrant status of all literals involved in the program, allowing query answering to be optimized; in this scenario, an *update* to the DeLP program can have far-reaching effects, or perhaps none at all. Thus, a naive approach based on total recomputation of the warrant status of all literals after each update can lead to a significant amount of wasted effort. We are interested in tackling the problem of minimizing such wasted effort.

Structured argumentation frameworks. In [3], four frameworks that consider the structure of arguments are presented; two of them—ASPIC$^+$ [4] and ABA [5,6]—build the set of all possible arguments from the knowledge base and then rely on using one of the possible Dung semantics to decide on the acceptance of arguments; the other two—Logic-Based Deductive Argumentation [7] and DeLP [8]—only build the arguments involved in answering the query.

Given that our primary focus is on the changes in the structure of the arguments that are used to answer a query, we have focused on the DeLP language; however, the ideas developed here can be used to inspire similar techniques for other structured argumentation frameworks such as ASPIC$^+$ and ABA, as will be discussed towards the end of Section 6. The last two frameworks mentioned above exhibit several differences [3]—among them is the base logic used as a knowledge representation language: [7] relies on propositional logic, requiring a theorem prover to solve queries, while DeLP [8] adopts an extension of logic programming, which is on the other hand a computational framework. For a better understanding of the differences among the above-mentioned frameworks, we refer the interested reader to [9], where a variant of DeLP using the grounded semantics is also discussed.

An important distinction between DeLP and the other three frameworks, which significantly affects the resolution of a query, rests on how attacks between arguments are described. As we will present below, DeLP considers two forms of defeat: *proper* and *blocking*; the former is akin to Dung's form of defeat, called *attack* in [10], whereas the latter behaves differently since the two arguments that are part of the blocking defeat relation, attacker and attackee, are defeated. Of course, this could be modeled in Dung's graphs as a mutual attack, but the DeLP mechanism forbids the use of two blocking defeaters successively because in a properly formed dialogue the introduction of another blocking defeater is unnecessary, since the first two are already defeated. Moreover, to find the answers required by the query, other considerations of dialogical nature are taken into account, which strengthens the reasoning process by forbidding common dialogical fallacies; these characteristics have been reflected in the development of a game-based semantics [11]. These considerations led us to the choice of DeLP as the basis of our research. It is interesting to note that the type of changes that could affect the knowledge base we study here from the algorithmic point of view seeking computational efficiency have been studied from the Belief Revision perspective in [12].

Contributions. With the aim of minimizing wasted effort in the computation of the warrant status of the literals of a DeLP program after performing an update, in this paper we make the following contributions:

- We investigate the complexity of the problems of deciding the existence of an argument as well as deciding its status, showing that both problems are computationally hard—this motivates the investigation of incremental and efficient techniques.
- We identify several cases in which single updates or sets of updates to be performed simultaneously are guaranteed to have no effect, *i.e.,* these updates are *irrelevant*, in the sense that the status of the literals does not change after performing such kinds of updates.
- We propose an approach based on the construction of a reachability hypergraph in order to keep track of literals that are potentially affected by a given update (namely, *influenced* and *core* literals), which leads to an incremental computation algorithm that focuses only on these literals and avoids the computation of the status of *inferable* and *preserved* literals.
- We empirically evaluate our approach, showing that it can lead to significant savings in running time for single updates consisting of rule addition/removal, as well as for multiple updates applied simultaneously.

Given the number of real-world applications of argumentation systems that are being implemented, the area concerned with the computational approaches to argumentation is requiring the attention of the research community. There is a significant effort to develop abstract argumentation solvers that show considerable success [13]; the immediate effect of these accomplishments can be seen in the systems that use this approach. Several lines of research have already examined the dynamics of argumentation following belief revision techniques over abstract argumentation,[1] which are not looking at the structure of arguments. Our work is based on a system that directly implements a semantics for computing answers, *i.e.,* it is concerned with deciding the status of a literal after a particular change in the knowledge base (program), looking to improve performance.

Organization. After briefly reviewing Defeasible Logic Programming (DeLP) in Section 2, where the notion of update is also presented, we introduce the concept of (labeled) hypergraph associated with a DeLP program, and investigate the complexity of problems related to the computation of the status of arguments in Section 3. Next, in Section 4 we present our

[1] See [14] for a discussion of the issues involved in the relation between Belief Revision and Argumentation.

incremental technique that deals with sets of updates applied simultaneously. The benchmark generator and the experimental evaluation covering both efficiency and effectiveness is presented in Section 5. Related work is discussed in Section 6, and the paper is concluded in Section 7, where future work is outlined. To facilitate readability, proofs of the results stated in the paper are reported in Appendix A.

2. Defeasible logic programming and updates

We first briefly review the syntax and semantics of Defeasible Logic Programming (DeLP)—the interested reader can find a thorough description of this formalism in [2]. After this we introduce the notion of updates for DeLP programs.

We assume the existence of a set **At** of atoms from which DeLP programs can be built. A literal is a ground atom $\alpha \in \mathbf{At}$ or its negation $\sim\alpha$, where symbol "~" represents strong negation; the formula $\sim\sim\alpha$ can be used for denoting α. We use *Lit* to denote the set of literals that can be obtained from the atoms in **At**, that is $Lit = \mathbf{At} \cup \{\sim\alpha \mid \alpha \in \mathbf{At}\}$. In the following, we assume that literals used to define programs and their updates are taken from the set *Lit*.

A DeLP program $\mathcal{P}$ consists of *strict* and *defeasible rules* defined using elements of **At** as follows. A strict rule is of the form $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n$, where $\alpha_0, \alpha_1, \ldots, \alpha_n$ (with $n \geq 0$) are literals. A defeasible rule is of the form $\alpha_0 \prec\!\!- \alpha_1, \ldots, \alpha_n$, where α_0, α_1, …, α_n ($n > 0$) are literals. Given a strict or defeasible rule r, we use $head(r)$ to denote α_0, and $body(r)$ to denote the set of literals $\{\alpha_1, \ldots, \alpha_n\}$. Strict rules with empty body will also be called *facts*. With a little abuse of notation, in the following we will often denote a fact ($\alpha \leftarrow$) simply by α. Intuitively, strict rules represent non-defeasible information, while defeasible rules represent tentative information (that is, information that can be used if nothing can be posed against it). In a program $\mathcal{P}$, we will distinguish the subset Π of strict rules, and the subset Δ of defeasible rules. We denote $\mathcal{P}$ as (Π, Δ) when needed. Moreover, we use $Lit_{\mathcal{P}}$ to denote the set of literals derived from the atoms occurring in a rule in $\mathcal{P}$.

To illustrate the language semantics, we will use the following example.

Example 1 *(Running example).* Let $\mathcal{P}_1 = (\Pi_1, \Delta_1)$ be a DeLP program for supporting doctors in medical diagnoses. The atoms used in $\mathcal{P}_1$ and their meaning are as follows:

t	Test proves presence of depression-related disorders
d	Patient is diagnosed with depression-related disorders
h	The patient is diagnosed with hyperactivity
b	Test proves presence of toxins in blood
s	Patient shows signs of stress
a	Patient is prescribed sleeping aids
i	Patient is diagnosed with insomnia
e	Patient shows symptoms of attention deficit disorder
f	Patient suffers from forgetfulness

Let us assume that the strict part of $\mathcal{P}_1$ is $\Pi_1 = \{\sim\mathtt{a},\ \mathtt{t},\ \mathtt{b},\ (\mathtt{d} \leftarrow \mathtt{t})\}$, and that the set of defeasible rules is as follows:

$$\Delta_1 = \left\{ \begin{array}{lll} (\mathtt{i} \prec\!\!- \mathtt{s}), & (\mathtt{s} \prec\!\!- \mathtt{h}), & (\mathtt{h} \prec\!\!- \mathtt{b}), \\ (\sim\mathtt{h} \prec\!\!- \mathtt{d}, \mathtt{t}), & (\sim\mathtt{i} \prec\!\!- \sim\mathtt{a}, \mathtt{s}), & (\mathtt{a} \prec\!\!- \mathtt{t}), \\ (\mathtt{s} \prec\!\!- \mathtt{d}), & (\mathtt{h} \prec\!\!- \mathtt{d}), & (\sim\mathtt{f} \prec\!\!- \sim\mathtt{e}), \\ (\sim\mathtt{e} \prec\!\!- \sim\mathtt{h}, \sim\mathtt{a}) & & \end{array} \right\}$$

Given a DeLP program $\mathcal{P} = (\Pi, \Delta)$ and a literal $\alpha \in Lit_{\mathcal{P}}$, a *(defeasible) derivation* for α w.r.t. $\mathcal{P}$ is a finite sequence $\alpha_1, \alpha_2, \ldots, \alpha_n = \alpha$ of literals such that (i) each literal α_i is in the sequence because there exists a (strict or defeasible) rule $r \in \mathcal{P}$ with head α_i and body $\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_k}$ such that $i_j < i$ for all $j \in [1, k]$, and (ii) there do not exist two literals α_i and α_j such that $\alpha_j = \sim\alpha_i$. A derivation is said to be a *strict derivation* if only strict rules are used.

Given a DeLP program $\mathcal{P} = (\Pi, \Delta)$, we denote with $CL(\mathcal{P})$ the set of literals obtained by computing the *deductive closure* of facts and strict rules of $\mathcal{P}$. For instance, for the program $\mathcal{P}_1$ of Example 1, $CL(\mathcal{P}_1) = \{\sim\mathtt{a}, \mathtt{b}, \mathtt{d}, \mathtt{t}\}$. A set of rules is *contradictory* if and only if there exist two defeasible derivations for two complementary literals α and $\sim\alpha$ from that set. We assume that Π is not contradictory (*i.e.,* $CL(\mathcal{P})$ does not contain two complementary literals)—checking this can be done in PTIME. However, complementary literals can be derived from $\mathcal{P}$ when defeasible rules are used in the derivation. Two literals α and β are said to be *contradictory* if (i) neither $\Pi \cup \{\alpha\}$ nor $\Pi \cup \{\beta\}$ are contradictory, whereas (ii) $\Pi \cup \{\alpha, \beta\}$ is contradictory. Pairs of complementary literals are clearly contradictory. A set of literals is said to be contradictory if it contains two contradictory literals.

Example 2. Considering program $\mathcal{P}_1$ from Example 1, the literal $\sim\mathtt{i}$ can be derived using the following sets of rules and facts: $\{(\sim\mathtt{i} \prec\!\!- \sim\mathtt{a}, \mathtt{s}), (\sim\mathtt{a}), (\mathtt{s} \prec\!\!- \mathtt{d}), (\mathtt{d} \leftarrow \mathtt{t}), (\mathtt{t})\}$; the derivation $(\mathtt{t}, \mathtt{d}, \mathtt{s}, \sim\mathtt{a}, \sim\mathtt{i})$ describes how rules can be ap-

plied to derive $\sim$i. However, the set of rules $\Pi_1 \cup \Delta_1$ is contradictory as also i can be derived using the rules: {(i$-\!\prec$s), (s$-\!\prec$d), (d $\leftarrow$ t), (t)}. The non-contradictory set of literals that can be derived from Π_1 is {$\sim$a, t, b, d}.

2.1. Dialectical process

DeLP incorporates a defeasible argumentation formalism for the treatment of contradictory knowledge, which allows the identification of the pieces of knowledge that are in contradiction, and a *dialectical process* is used for deciding which information prevails as warranted. This process involves the construction and evaluation of arguments that either support or interfere with a user-issued query.

Definition 1 *(Argument).* Given a DeLP program $\mathcal{P} = (\Pi, \Delta)$ and a literal α, we say that $\langle \mathcal{A}, \alpha \rangle$ is an argument for α if $\mathcal{A}$ is a set of defeasible rules of Δ such that:

(i) there is a derivation for α from $\Pi \cup \mathcal{A}$;
(ii) the set $\Pi \cup \mathcal{A}$ is not contradictory; and
(iii) $\mathcal{A}$ is minimal (*i.e.,* there is no proper subset $\mathcal{A}'$ of $\mathcal{A}$ satisfying both (i) and (ii)).

Example 3. Given the DeLP program $\mathcal{P}_1$ from our running example, some of the arguments we can obtain are the following:

$\langle \mathcal{A}_1, \mathtt{i} \rangle = \langle \{(\mathtt{i} -\!\prec \mathtt{s}),\ (\mathtt{s} -\!\prec \mathtt{h}),\ (\mathtt{h} -\!\prec \mathtt{b})\}, \mathtt{i} \rangle;$

$\langle \mathcal{A}_2, \mathtt{i} \rangle = \langle \{(\mathtt{i} -\!\prec \mathtt{s}),\ (\mathtt{s} -\!\prec \mathtt{d})\}, \mathtt{i} \rangle;$

$\langle \mathcal{A}_3, \mathtt{h} \rangle = \langle \{(\mathtt{h} -\!\prec \mathtt{b})\}, \mathtt{h} \rangle;$

$\langle \mathcal{A}_4, \sim\mathtt{i} \rangle = \langle \{(\sim\mathtt{i} -\!\prec \sim\mathtt{a}, \mathtt{s}),\ (\mathtt{s} -\!\prec \mathtt{d})\}, \sim\mathtt{i} \rangle;$

$\langle \mathcal{A}_5, \sim\mathtt{h} \rangle = \langle \{(\sim\mathtt{h} -\!\prec \mathtt{t}, \mathtt{d})\}, \sim\mathtt{h} \rangle;$

$\langle \mathcal{A}_6, \mathtt{h} \rangle = \langle \{(\mathtt{h} -\!\prec \mathtt{d})\}, \mathtt{h} \rangle.$

An argument $\langle \mathcal{A}, \alpha \rangle$ is said to be a *sub-argument* of $\langle \mathcal{A}', \alpha' \rangle$ if $\mathcal{A} \subseteq \mathcal{A}'$. For instance, $\langle \mathcal{A}_7, \mathtt{s} \rangle = \langle \{(\mathtt{s} -\!\prec \mathtt{h}),\ (\mathtt{h} -\!\prec \mathtt{b})\}, \mathtt{s} \rangle$ is a sub-argument of $\langle \mathcal{A}_1, \mathtt{i} \rangle$. As another example, considering the program $\langle \{a \leftarrow b,\quad b \leftarrow c,\quad d \leftarrow\}, \{c -\!\prec d\} \rangle$, we have that both relations $\langle \{c -\!\prec d\}, b \rangle \subseteq \langle \{c -\!\prec d\}, c \rangle$ and $\langle \{c -\!\prec d\}, a \rangle \subseteq \langle \{c -\!\prec d\}, b \rangle$ hold.

A literal α is said to be *warranted* if there exists a non-defeated argument $\langle \mathcal{A}, \alpha \rangle$. To establish if an argument $\langle \mathcal{A}, \alpha \rangle$ is non-defeated, *defeaters* for $\langle \mathcal{A}, \alpha \rangle$ are considered. To define defeaters we refer to a (partial) order relation $\succ$ over arguments that will be discussed later.

Definition 2 *(Defeater).* Let $\mathcal{P} = (\Pi, \Delta)$ be a DeLP program, and $\succ$ be a preference relation defined over arguments of $\mathcal{P}$. An argument $\langle \mathcal{A}, \alpha \rangle$ is a *defeater* for an argument $\langle \mathcal{B}, \beta \rangle$ if and only if:

(i) there is a sub-argument $\langle \mathcal{C}, \gamma \rangle$ of $\langle \mathcal{B}, \beta \rangle$,
(ii) α and γ are contradictory, and
(iii) $\langle \mathcal{C}, \gamma \rangle$ is not preferred to $\langle \mathcal{A}, \alpha \rangle$ (*i.e.,* $\langle \mathcal{C}, \gamma \rangle \not\succ \langle \mathcal{A}, \alpha \rangle$).

A defeater $\langle \mathcal{A}, \alpha \rangle$ for $\langle \mathcal{B}, \beta \rangle$ is *proper* if $\langle \mathcal{A}, \alpha \rangle \succ \langle \mathcal{C}, \gamma \rangle$; otherwise, it is a blocking defeater.

Preferences among arguments can be defined explicitly, by introducing *priority* among rules, or implicitly, by formalizing criteria that allow to derive preferences among arguments. In this paper we consider the implicit criterion known as *generalized specificity* (used in the current implementation of DeLP), where arguments with greater information content or with less use of rules (w.r.t. subset minimality) are preferred [15]. The next definition characterizes the specificity criterion, defined in [2].

Definition 3 *(Specificity Preference Criterion [2]).* Let $\mathcal{P} = (\Pi, \Delta)$ be a DeLP program and let Π_s be the set of all strict rules from Π (without including facts). Let $\mathcal{F}$ be the set of all literals that have a defeasible derivation from $\mathcal{P}$ ($\mathcal{F}$ will be considered as a set of facts). Let $\langle \mathcal{A}_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, \alpha_2 \rangle$ be two arguments obtained from $\mathcal{P}$. $\langle \mathcal{A}_1, \alpha_1 \rangle$ is *strictly more specific than* $\langle \mathcal{A}_2, \alpha_2 \rangle$ (denoted $\langle \mathcal{A}_1, \alpha_1 \rangle \succ \langle \mathcal{A}_2, \alpha_2 \rangle$) if the following conditions hold:

1. For all $\mathcal{L} \subseteq \mathcal{F}$: if α_1 has a defeasible derivation from $\Pi_s \cup \mathcal{L} \cup \mathcal{A}_1$ and α_1 does not have a strict derivation from $\Pi_s \cup \mathcal{L}$, then α_2 has a defeasible derivation from $\Pi_s \cup \mathcal{L} \cup \mathcal{A}_2$, and
2. there exists $\mathcal{L}' \subseteq \mathcal{F}$ s.t. α_2 has a defeasible derivation from $\Pi_s \cup \mathcal{L}' \cup \mathcal{A}_2$, α_2 does not have a strict derivation from $\Pi_s \cup \mathcal{L}'$, and α_1 does not have a defeasible derivation from $\Pi_s \cup \mathcal{L}' \cup \mathcal{A}_1$.

In Definition 3, the subsets $\mathcal{L}$ of $\mathcal{F}$ are used to "activate" the arguments under comparison. We say that $\mathcal{L}$ activates an argument $\langle \mathcal{A}, \alpha \rangle$ when $\Pi_s \cup \mathcal{L} \cup \mathcal{A}$ derives the conclusion α. Then, an argument $\langle \mathcal{A}_1, \alpha_1 \rangle$ is strictly more specific than an argument $\langle \mathcal{A}_2, \alpha_2 \rangle$ when each $\mathcal{L} \subseteq \mathcal{F}$ that activates $\langle \mathcal{A}_1, \alpha_1 \rangle$ also activates $\langle \mathcal{A}_2, \alpha_2 \rangle$, but there is at least one $\mathcal{L}' \subseteq \mathcal{F}$ that activates $\langle \mathcal{A}_2, \alpha_2 \rangle$ and does not activate $\langle \mathcal{A}_1, \alpha_1 \rangle$. Notice that Π_s does not contain the facts in the program; therefore, the activation of the arguments will only depend on the derived literals. Intuitively, it is possible to think about the subsets of $\mathcal{F}$ as cut-sets of the tree defined by the derivation of the conclusion, *i.e.,* sets of literals that will stop the "flow" from the facts to the conclusion.

From now on we assume that the status of arguments is computed by using generalized specificity as the preference relation $\succ$, which is also the default criterion adopted by the DeLP solver used in the experiments. However, we will abstract away from this criterion and simply assume the existence of a preference relation $\succ$ defined on the arguments of $\mathcal{P}$.

We are now ready to define the concept of *dialectical tree* which is used to decide the status of a conclusion in DeLP.

Definition 4 *(Dialectical Tree).* Let $\langle \mathcal{A}_0, \alpha_0 \rangle$ be an argument for a DeLP program $\mathcal{P}$. A dialectical tree for $\langle \mathcal{A}_0, \alpha_0 \rangle$, denoted as $\mathcal{T}_{\langle \mathcal{A}_0, \alpha_0 \rangle}$, is defined as follows.
The root node of $\mathcal{T}_{\langle \mathcal{A}_0, \alpha_0 \rangle}$ is labeled with $\langle \mathcal{A}_0, \alpha_0 \rangle$.
Let N be a non-root node of $\mathcal{T}_{\langle \mathcal{A}_0, \alpha_0 \rangle}$ labeled with $\langle \mathcal{A}_n, \alpha_n \rangle$.
Let $\Lambda = [\langle \mathcal{A}_0, \alpha_0 \rangle, \ldots, \langle \mathcal{A}_n, \alpha_n \rangle]$ be a finite sequence of arguments (called *argumentation line*) of the path from the root to N such that the following four constraints hold:

i) every argument of the sequence defeats its predecessor;
ii) the arguments in odd (resp. even) positions of the sequence are concordant (*i.e.,* $\Pi \cup \{\mathcal{A}_i \mid 0 \leq i \leq n, i \bmod 2 = 0$ (resp. 1)$\}$ is not contradictory);
iii) two blocking defeaters cannot appear one immediately after the other in the sequence; and
iv) no sub-argument $\langle \mathcal{A}_i, \alpha_i \rangle$ of $\langle \mathcal{A}_j, \alpha_j \rangle$ with $i > j$ appears in the sequence.

If there exists a defeater $\langle \mathcal{B}, \beta \rangle$ of $\langle \mathcal{A}_n, \alpha_n \rangle$, such that the four constraints i)–iv) hold for the sequence $\Lambda' = [\langle \mathcal{A}_0, \alpha_0 \rangle, \ldots, \langle \mathcal{A}_n, \alpha_n \rangle, \langle \mathcal{B}, \beta \rangle]$, then the node labeled with $\langle \mathcal{B}, \beta \rangle$ is a child of N, otherwise N is a leaf.

It is easy to see that many acceptable argumentation lines (i.e. sequences of arguments observing constraints *i)–iv)* of Definition 4) could arise from one argument, leading to the tree structure called *dialectical tree*, which represents an exhaustive dialectical analysis for the argument in its root.

Example 4. Consider the arguments from Example 3 and the following preference relations (obtained through Definition 3): $\langle \mathcal{A}_5, \sim\mathtt{h} \rangle \succ \langle \mathcal{A}_6, \mathtt{h} \rangle$ and $\langle \mathcal{A}_3, \mathtt{h} \rangle \succ \langle \mathcal{A}_5, \sim\mathtt{h} \rangle$. Then, $[\langle \mathcal{A}_6, \mathtt{h} \rangle, \langle \mathcal{A}_5, \sim\mathtt{h} \rangle, \langle \mathcal{A}_3, \mathtt{h} \rangle]$ is a sequence of arguments that forms a path of the dialectical tree $\mathcal{T}_{\langle \mathcal{A}_6, \mathtt{h} \rangle}$.

A dialectical tree $\mathcal{T}_{\langle \mathcal{A}, \alpha \rangle}$ can be marked in order to decide the status of the literal α at its root through a bottom up marking procedure that leads to the corresponding so-called *marked dialectical tree*, which is denoted with $\mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$. It is built by first marking all leaves of $\mathcal{T}_{\langle \mathcal{A}, \alpha \rangle}$ as UNDEFEATED; then, every non-leaf node N is marked as DEFEATED if and only if at least one of its children is marked as UNDEFEATED, otherwise N is marked as UNDEFEATED.

The warrant status of literals is then easily determined by checking how the roots of the dialectical trees for arguments for them are marked. Particularly, for each literal α of the program, if there exists a marked dialectical tree whose root contains an argument for α, and it is marked as UNDEFEATED, then we will say that α is *warranted*.

Example 5. In our running example, h is a warranted literal for $\mathcal{P}_1$. In fact, $\mathcal{T}^*_{\langle \mathcal{A}_6, \mathtt{h} \rangle}$ for $\langle \mathcal{A}_6, \mathtt{h} \rangle$ consists of only the argumentation line $[\langle \mathcal{A}_6, \mathtt{h} \rangle, \langle \mathcal{A}_5, \sim\mathtt{h} \rangle, \langle \mathcal{A}_3, \mathtt{h} \rangle]$ of Example 4. Thus, $\langle \mathcal{A}_3, \mathtt{h} \rangle$ and $\langle \mathcal{A}_6, \mathtt{h} \rangle$ are marked as UNDEFEATED, while $\langle \mathcal{A}_5, \sim\mathtt{h} \rangle$ is DEFEATED. With a little effort it can be checked that t, d, b, s, and $\sim$a are warranted as well.

Given a DeLP program $\mathcal{P}$, we define a total function $S_{\mathcal{P}} : Lit \rightarrow \{\text{IN}, \text{OUT}, \text{UNDECIDED}\}$ assigning a *status* to each literal w.r.t. $\mathcal{P}$ as follows: $S_{\mathcal{P}}(\alpha) = \text{IN}$ if there exists a (marked) dialectical tree $\mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$ such that α is warranted; $S_{\mathcal{P}}(\alpha) = \text{OUT}$ if $S_{\mathcal{P}}(\sim\alpha) = \text{IN}$; $S_{\mathcal{P}}(\alpha) = \text{UNDECIDED}$ if neither $S_{\mathcal{P}}(\alpha) = \text{IN}$ nor $S_{\mathcal{P}}(\alpha) = \text{OUT}$. For literals not occurring in the program we also say that their status is unknown. In our example, $S_{\mathcal{P}_1}(\mathtt{t}) = S_{\mathcal{P}_1}(\mathtt{d}) = S_{\mathcal{P}_1}(\mathtt{b}) = S_{\mathcal{P}_1}(\mathtt{h}) = S_{\mathcal{P}_1}(\mathtt{s}) = S_{\mathcal{P}_1}(\sim\mathtt{a}) = \text{IN}$; the complementary literals of these IN literals are OUT (*e.g.,* $S_{\mathcal{P}_1}(\sim\mathtt{h}) = S_{\mathcal{P}_1}(\mathtt{a}) = \text{OUT}$), and the remaining literals are UNDECIDED (*e.g.,* $S_{\mathcal{P}_1}(\sim\mathtt{i}) = S_{\mathcal{P}_1}(\mathtt{i}) = S_{\mathcal{P}_1}(\sim\mathtt{e}) = S_{\mathcal{P}_1}(\mathtt{e}) = \text{UNDECIDED}$).

With a little abuse of notation, we use $S_{\mathcal{P}}$ (without any argument) to denote the status of all the literals of $\mathcal{P}$. Even though (similarly to abstract argumentation [10]) computing the semantics of DeLP programs consists in assigning a truth value (or label) to arguments, the technique for computing the status of arguments in DeLP programs is very different from those computing extensions of abstract argumentation frameworks (AFs), because of the existence of different semantics. Particularly, while AFs can be modeled by logic programs with strict rules and partial stable semantics [16,17], DeLP programs consist of both strict and defeasible rules with a specific semantics [2].

2.2. Updates

In this section we introduce the concept of *update* for a DeLP program. We consider general updates consisting of the addition and deletion of rules to be performed simultaneously.

An *update* modifies a DeLP program $\mathcal{P} = \langle \Pi, \Delta \rangle$ into a new one $\mathcal{P}' = \langle \Pi', \Delta' \rangle$ by adding or removing a strict or a defeasible rule. In particular, we allow the removal of any rule r of $\mathcal{P}$ through an update, and consider the addition of rules r such that $body(r) \subseteq Lit_{\mathcal{P}}$ and $head(r) \subseteq Lit$, thus allowing also the addition of rules whose head is a literal not belonging to $Lit_{\mathcal{P}}$. Given a DeLP program $\mathcal{P}$ and a strict or defeasible rule r, we use $+r$ (resp., $-r$) to denote a rule addition (resp., deletion) update to be performed on $\mathcal{P}$. We write $u = \pm r$ for denoting an update u consisting of either a rule addition ($+r$) or a rule removal ($-r$).

We use U to denote *a set of updates*, each of them having one of the forms described above: strict rule addition/deletion or defeasible rule addition/deletion. Hence, U may contain both additions and deletions of strict or defeasible rules. For any set of updates U (also called *multiple update*), we denote by $U^p = \{r \mid +r \in U\}$ and $U^n = \{r \mid -r \in U\}$ the set of *rules* that will be added or deleted, respectively. Moreover, we also use the notation $U^+ = \{+r \mid +r \in U\}$ and $U^- = \{-r \mid -r \in U\}$ to denote the sets of positive and negative *updates*, respectively. Given a set U of updates, we denote by U_s (resp. U_d) the set of strict (resp. defeasible) updates in U. Finally, we use $U(\mathcal{P})$ to denote the result of applying update U to DeLP program $\mathcal{P} = (\Pi, \Delta)$, defined as follows:

$$U(\mathcal{P}) = \big\langle (\Pi \cup U_s^p) \setminus U_s^n,\ \ (\Delta \cup U_d^p) \setminus U_d^n \big\rangle.$$

We say that a set of updates U is *feasible* with respect to a DeLP program $\mathcal{P} = (\Pi, \Delta)$, if (i) $U^n \subseteq \Pi \cup \Delta$, (ii) $U^p \cap (\Pi \cup \Delta) = \emptyset$ (and thus $U^p \cap U^n = \emptyset$), and (iii) the strict part Π' of the updated program $U(\mathcal{P}) = (\Pi', \Delta')$ is not contradictory. *In the following we assume that our set of updates U is always feasible.*

We will also use the notation $CL(U, \mathcal{P}) = CL(\mathcal{P}) \cap CL(U(\mathcal{P}))$ to denote the set of literals that are in the deductive closure of facts and strict rules both before and after the updates. For instance, for the DeLP program $\mathcal{P}_1$ of Example 1 and the update $U = \{-(\mathtt{d} \leftarrow \mathtt{t})\}$, we have that $CL(U, \mathcal{P}_1) = \{\sim\mathtt{a}, \mathtt{b}, \mathtt{t}\}$. *With a little abuse of notation, in the following, when focusing on a single update $u = \pm r$, we simply write $u(\mathcal{P})$ for denoting the updated program, instead of writing $\{u\}(\mathcal{P})$.*

Example 6. Consider the DeLP program $\mathcal{P}_1 = (\Pi_1, \Delta_1)$ of Example 1, and the update $U = \{+r\}$ where $r = (\sim\mathtt{i} \prec \mathtt{h})$. The updated DeLP program $U(\mathcal{P}_1)$ is $(\Pi_1, (\Delta_1 \cup \{r\}))$. Note that new arguments may arise after performing an update; for instance, $\langle \mathcal{A}_8, \sim\mathtt{i} \rangle = \langle \{\sim\mathtt{i} \prec \mathtt{h},\ \mathtt{h} \prec \mathtt{b}\}, \sim\mathtt{i} \rangle$ exists for the updated program, but it did not exist for $\mathcal{P}_1$.

3. Labeled hypergraphs for DeLP programs and complexity analysis

In this section, we first introduce the concept of labeled hypergraph associated with a DeLP program. This structure will be used both in the complexity analysis provided in this section as well as in several definitions that make up the basis of our incremental approach.

Recall that a *directed hypergraph* is a pair $\langle N, H \rangle$, where N is the set of nodes and $H \subseteq (2^N \setminus \emptyset) \times N$ is the set of hyper-edges. For hyper-edge (S, t), S is a non-empty set called the *source set*, while t is called the *target node*.

The next definition introduces the concept of hypergraph $G(\mathcal{P})$ derived from a DeLP program.[2]

Definition 5 *(Labeled hypergraph for a DeLP program).* Let $\mathcal{P} = (\Pi, \Delta)$ be a DeLP program and $L = \{\mathtt{def}, \mathtt{str}, \mathtt{cfl}\}$ be an alphabet of labels. $G(\mathcal{P}) = \langle N, H \rangle$, where $H \subseteq (2^N \setminus \emptyset) \times N \times L$, is defined as follows:

- (Basic step) For each fact α in Π, then $\alpha \in N$;
- (Iterative step) For each strict (resp. defeasible) rule r in $\mathcal{P}$ with $head(r) = \alpha_0$, $body(r) = \alpha_1, \ldots, \alpha_n$ such that $n > 0$ and $\alpha_1, \ldots, \alpha_n \in N$, then $\alpha_0 \in N$ and $(\{\alpha_1, \ldots, \alpha_n\}, \alpha_0, \mathtt{str}) \in H$ (resp. $(\{\alpha_1, \ldots, \alpha_n\}, \alpha_0, \mathtt{def}) \in H$);
- (Final step) For each pair of nodes in N representing complementary literals α and $\sim\alpha$, both $(\{\alpha\}, \sim\alpha, \mathtt{cfl}) \in H$ and $(\{\sim\alpha\},\ \alpha, \mathtt{cfl}) \in H$.

It is worth noting that labels in the previous definition are used to distinguish hyper-edges corresponding to strict and defeasible rules (second item) and pairs of complementary literals (third item). Rules are represented by hyper-edges that are labeled depending on the kind of rule. Thus, facts in Π belong to the set N of nodes of $G(\mathcal{P})$. Then, for each (strict or defeasible) rule whose body is in N, the head is added to N and a hyper-edge corresponding to the rule is added to the set H of hyper-edges. Finally, there is a pair of (hyper-)edges for each pair of complementary literals. As discussed below, the hypergraph $G(\mathcal{P})$ contains all the literals for which there may exist an argument in $\mathcal{P}$.

[2] Although graphs could be used instead of hypergraphs, we consider that it is more natural (and readable) to associate rules with hyper-edges since the related components are made explicit.

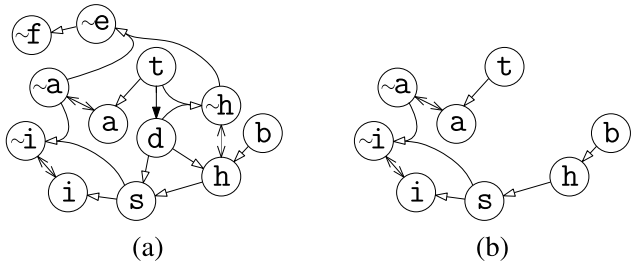


Fig. 1. (a) $G(\mathcal{P}_1)$ for program $\mathcal{P}_1$ from Example 1; (b) $G(\mathcal{P}'_1)$ for program $\mathcal{P}'_1 = \{-(\mathtt{d} \leftarrow \mathtt{t})\}(\mathcal{P}_1)$ from Example 7.

Example 7. The hypergraph $G(\mathcal{P}_1)$ for the DeLP program $\mathcal{P}_1$ of Example 1 is shown in Fig. 1(a) where $\leftrightarrow$ (resp. $\lhd\!\!-$ and $\blacktriangleleft\!\!-$) denotes hyper-edges labeled as `cfl` (resp. `def` and `str`). Consider the program $\mathcal{P}'_1$ obtained by applying update $u = -(\mathtt{d} \leftarrow \mathtt{t})$ on $\mathcal{P}_1$. The labeled hypergraph $G(\mathcal{P}'_1)$ is shown in Fig. 1(b).

The following proposition states an important property of the hypergraph $G(\mathcal{P})$ for a given DeLP program $\mathcal{P}$, and in particular it provides the relationship between the fact that a literal appears in $G(\mathcal{P})$ and the existence of an argument for it w.r.t. $\mathcal{P}$. As shown later, deciding whether there is an argument for a given literal is NP-hard.

Proposition 1. *Given a DeLP program $\mathcal{P}$ and an argument $\langle \mathcal{A}, \alpha \rangle$ of $\mathcal{P}$, then α occurs as a node of $G(\mathcal{P})$.*

The previous proposition entails that if a literal $\alpha \in Lit_{\mathcal{P}}$ does not belong to $G(\mathcal{P})$, then there is no argument for α w.r.t. $\mathcal{P}$; however, it is not guaranteed that if a literal $\alpha \in Lit_{\mathcal{P}}$ belongs to $G(\mathcal{P})$ then there exists an argument for α w.r.t. $\mathcal{P}$ (see the proof of Proposition 1 in Appendix A).

Given $G(\mathcal{P}) = \langle N, H \rangle$, we say that there is a path from a literal β to a literal α if either (i) there exists a hyper-edge whose source set contains β and whose target is α, or (ii) there exists a literal γ and two paths, from β to γ and from γ to α.

In the following, we say that a literal α *depends* on a literal β in $G(\mathcal{P})$ if there is a path from β to α in $G(\mathcal{P})$.

Structure $G(\mathcal{P})$ can be built in polynomial time in the size of $\mathcal{P}$; thus, deciding whether a literal depends on another one can be done in PTIME. The status of a literal is computed by the dialectical process that builds a (marked) dialectical tree where each node is an argument. Our first result regards the complexity of deciding whether there exists an argument for a given literal. Formally, given a DeLP program $\mathcal{P}$ and a literal $\alpha \in Lit_{\mathcal{P}}$, *ArgumentExistence*$[\mathcal{P}, \alpha]$ is the problem of deciding whether there exists an argument for α w.r.t. $\mathcal{P}$. The next theorem states that this decision problem is hard.

Theorem 1. *ArgumentExistence$[\mathcal{P}, \alpha]$ is NP-complete.*

The DeLP program defined in the hardness proof of Theorem 1 (reported in Appendix A) can be transformed to an equivalent one where all rules have at most two literals in the body. In fact, a rule with n body literals can be replaced with $n-1$ rules, each of them having two body literals. For instance, a rule of the form $a \leftarrow b, c, d, e$ can be replaced by the following three rules $a \leftarrow \gamma', \gamma''$, $\gamma' \leftarrow b, c$, and $\gamma'' \leftarrow d, e$, where γ' and γ'' are fresh literals. Then, as formally proved in Appendix A, using this construction we can show that the result of Theorem 1 holds even for DeLP programs where rules contain at most two body literals.

Corollary 1. *ArgumentExistence$[\mathcal{P}, \alpha]$ is NP-complete, even if $|body(r)| \leq 2$ for all $r \in (\Pi \cup \Delta)$ (where $\mathcal{P} = \langle \Pi, \Delta \rangle$).*

The following result identifies tractable cases for the problem of deciding whether there is an argument for a given literal.

Proposition 2. *ArgumentExistence$[\mathcal{P}, \alpha]$ is in PTIME if either* (i) *α does not depend in $G(\mathcal{P})$ on literals β and γ such that $\{\beta, \gamma\} \cup \Pi$ is contradictory, or* (ii) *α is not in $G(\mathcal{P})$.*

Intuitively, case *(i)* holds because if α does not depend on two literals in $G(\mathcal{P})$ whose presence leads to a contradiction, a derivation for α can be obtained from the rules in an inverse path from α to facts in $\mathcal{P}$. Case *(ii)* follows from Proposition 1 and the fact that $G(\mathcal{P})$ can be built in PTIME.

The next result regards the complexity of deciding the status of a given literal.[3] Formally, given a DeLP program $\mathcal{P}$, a literal $\alpha \in Lit_{\mathcal{P}}$, and a status $\sigma \in \{\text{IN}, \text{OUT}, \text{UNDECIDED}\}$, *LiteralStatus*$[\mathcal{P}, \alpha, \sigma]$ is the problem of deciding whether the status of α is σ, *i.e.,* deciding whether $S_{\mathcal{P}}(\alpha) = \sigma$.

Theorem 2. *LiteralStatus*$[\mathcal{P}, \alpha, \sigma]$ *is coNP-hard for* $\sigma \in \{\text{IN}, \text{OUT}\}$ *and NP-hard for* $\sigma = \text{UNDECIDED}$*, even if* $|body(r)| \leq 2$ *for all* $r \in (\Pi \cup \Delta)$ *(where* $\mathcal{P} = \langle \Pi, \Delta \rangle$*).*

The fact that computing the status of literals is hard motivated the investigation of incremental techniques, which is the topic of the next two sections.

4. Incremental computation

We now address the problem of recomputing the status $S_{\mathcal{P}'}$ of the literals w.r.t. an updated program $\mathcal{P}' = U(\mathcal{P})$, given $\mathcal{P}$ with status $S_{\mathcal{P}}$. Whenever the status of a literal α does not change, *i.e.,* $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$, we say that (the status of) α is *preserved*. Establishing whether the status of literals occurring in a DeLP program is preserved can be carried out by recomputing it from scratch. However, for some literals, this can be avoided or made more efficient by considering only a restricted portion of the program (or, equivalently, of the corresponding labeled hypergraph).

Our approach consists of two main steps. Firstly it is checked whether updates are *irrelevant*, which means that *all* literals in *Lit* are preserved. In such a case the initial status $S_{\mathcal{P}}$ is returned. Otherwise we have to: (*i*) compute the set of literals that are "influenced" by the updates; (*ii*) among the influenced literals determine the subset of literals (called *core literals*) whose status may change after performing the updates; (*iii*) compute the status of the core literals; and (*iv*) determine the status of *inferable* literals (i.e., the literals whose status can be immediately determined from the status of the core literals).

In the following, given a DeLP program $\mathcal{P}$, we will consider a set U of "homogeneous" updates, that is updates that are all insertions or all deletions, i.e. either $U = U^-$ or $U = U^+$. The idea is that any set U consisting of both positive and negative updates can be performed by first performing U^- and then performing U^+. In fact, since $U = U^- \cup U^+$, the status of a literal w.r.t. $U(\mathcal{P})$ is equal to that w.r.t. $U^+(U^-(\mathcal{P}))$, that is, $S_{U(\mathcal{P})}(\alpha) = S_{U^+(U^-(\mathcal{P}))}(\alpha)\, \forall \alpha \in Lit$. *Hence, from now on, unless otherwise specified, we assume we deal with sets of homogeneous updates only.*

Given a set U of (homogeneous) updates, we denote with $G(U, \mathcal{P})$ the labeled hypergraph of the program updated through U or of the original program, depending on whether U consists of insertions or deletions. That is, $G(U, \mathcal{P}) = G(U^+(\mathcal{P}))$ if $U = U^+$, while $G(U, \mathcal{P}) = G(\mathcal{P})$ if $U = U^-$. As it will become clearer in the following, the reason for considering $G(\mathcal{P})$ in rule removal cases—instead of $G(U^-(\mathcal{P}))$—is that to determine the set of literals whose status may change by deleting a rule r, we need to consider the hypergraph also containing the hyper-edge derived from r.

Reachable nodes. Given $G(\mathcal{P}) = \langle N, H \rangle$, a node/literal $y \in N$, and a set $X \subseteq N$ of nodes/literals, we say that y is *reachable* from X if there exists a path in $G(\mathcal{P})$ from some x in X to y. We use $Reach_{G(\mathcal{P})}(X)$ to denote the set of all nodes that are reachable from X in $G(\mathcal{P})$. For instance, considering the program in Example 1 and its hypergraph shown in Fig. 1(a), we have that $Reach_{G(\mathcal{P}_1)}(\{\mathtt{d}\}) = \{\mathtt{d}, \mathtt{h}, \sim\mathtt{h}, \mathtt{s}, \sim\mathtt{i}, \mathtt{i}, \sim\mathtt{e}, \sim\mathtt{f}\}$. Moreover, we also use the notation $ReachStr_{G(\mathcal{P})}(X)$ to denote the set of nodes reachable from X using only *strict* hyper-edges, and $ReachStr^{-1}_{G(\mathcal{P})}(X)$ to denote all the nodes y in $G(\mathcal{P})$ for which there exists a path from y to some node $x \in X$ using strict hyper-edges only (*e.g.,* $ReachStr^{-1}_{G(\mathcal{P}_1)}(\{\mathtt{d}\}) = \{\mathtt{t}\}$). Computing $Reach_{G(\mathcal{P})}(X)$, $ReachStr_{G(\mathcal{P})}(X)$, and $ReachStr^{-1}_{G(\mathcal{P})}(X)$ can all be done in PTIME.

It is worth noting that our concept of reachability among literals in the hypergraph $G(\mathcal{P})$ for a DeLP-program $\mathcal{P}$ is conceptually similar to that of reachability among abstract arguments in an argumentation graph in the context of AF [18–22], that was initially proposed for incremental computation in [23]. The main difference is that reachability for an AF is defined over the graph representing the AF, whereas in our case it is defined over the hypergraph reproducing a DeLP program. Moreover, in abstract argumentation the status of an (abstract) argument α is affected only by the status of arguments α' from which it is reachable (as it happens for semantics satisfying the directionality property [18]). This is not the case for DeLP programs, where the status of a literal α may also depend on the status of a literal α'' that is reachable from α through strict hyper-edges only (an example of such a situation is given in Example 14). For this reason, we now introduce the concept of literal *related* to a given update and program by extending the concept of reachable literals.

Literals related to updates. We now introduce an important concept that, as stated in Theorem 3, is useful to restrict the set of rules to be considered for recomputing the status of literals.[4]

Definition 6 *(Related literal).* Let $\mathcal{P}$ be a program, U a set of (homogeneous) updates, and let

(i) $\mathcal{R}^0(U, \mathcal{P}) = \{head(r) \mid \pm r \in U\}$;
(ii) $\mathcal{R}^i(U, \mathcal{P}) = Reach_{G(U,\mathcal{P})}(ReachStr^{-1}_{G(U,\mathcal{P})}(\mathcal{R}^{i-1}(U, \mathcal{P})))$.

[3] We thank one of the anonymous referees for suggesting the current formulation of the result of Theorem 2 and a proof strategy.
[4] A similar concept is that of *relevant argument* that has been discussed in [23,24] in the context of AF.

The set of literals that are *related* to U w.r.t. $\mathcal{P}$ is $\mathcal{R}(U,\mathcal{P}) = \mathcal{R}^n(U,\mathcal{P})$ such that $\mathcal{R}^n(U,\mathcal{P}) = \mathcal{R}^{n-1}(U,\mathcal{P})$.

Example 8. Consider the DeLP program $\mathcal{P}_1$ of Example 1 and the update $U = U^- = \{-(\mathtt{i} \prec \mathtt{s})\}$. Then, the set of literals that are related to U w.r.t. $\mathcal{P}_1$ is $\mathcal{R}(U,\mathcal{P}_1) = \{\mathtt{i}, \sim\mathtt{i}\}$.

For the update $\{-(\mathtt{d} \leftarrow \mathtt{t})\}$ of Example 7, the set of related literals consists of all the literals in the hypergraph $G(\mathcal{P}_1)$ in Fig. 1(a), except literal $\mathtt{b}$.

The following theorem provides sufficient conditions under which the status of a given literal does not change after performing a set of updates. In particular, one of the conditions in Theorem 3 ensuring that a literal α is preserved is that $\alpha \notin \mathcal{R}(U,\mathcal{P})$. That is, only literals reachable as specified in step (ii) of Definition 6 from literals occurring in the head of rules in U may change their status after performing U. We will further restrict this condition in Section 4.2, where we introduce the set of influenced literals.

Theorem 3 *(Preserved Literal). Let $\mathcal{P}$ be a DeLP program and U a set of updates for $\mathcal{P}$. Let $\mathcal{P}' = U(\mathcal{P})$ be the updated program, and $G(\mathcal{P}') = \langle N', H' \rangle$ be the updated labeled hypergraph. Then, a literal $\alpha \in N'$ is preserved (i.e., $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$) if either (i) $\{\alpha, \sim\alpha\} \cap CL(U,\mathcal{P}) \neq \emptyset$, or (ii) $\alpha \notin \mathcal{R}(U,\mathcal{P})$.*

4.1. Irrelevant updates

Let $\mathcal{P}$ be a DeLP program, U a set of updates for $\mathcal{P}$, and $\mathcal{P}' = U(\mathcal{P})$ the updated program; we say that U is *irrelevant* for $\mathcal{P}$ iff $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)\ \forall \alpha \in Lit$.

In this subsection, we identify sufficient conditions guaranteeing that a set of updates is irrelevant. Although the underlying idea is similar to that explored for AFs [20,21], the criteria to determine irrelevant updates are completely different as the underlying formalisms, DeLP and AF, rely on different semantics; *i.e.,* a specific semantics for defeasible logic programs in our case, and abstract argumentation semantics for AFs [16]. We start with the following lemma stating that, if we can find an appropriate order among the updates in a set such that by applying the single updates following the given order each single update is irrelevant, then the whole set of updates is irrelevant.

Lemma 1. *Let $\mathcal{P}$ be a DeLP program and U be a set of updates. If there exists a sequence $[u_1, \ldots, u_{j-1}, u_j, \ldots, u_k]$ of updates such that (i) $u_1 \in U$ is irrelevant for $\mathcal{P}$ and (ii) every update $u_j \in U$, with $j \geq 2$, is irrelevant for the updated program $u_{j-1}(\ldots(u_1(\mathcal{P})))$, then U is irrelevant for $\mathcal{P}$.*

The following example shows that the converse of the statement of Lemma 1 does not hold: a set U of updates may turn out to be irrelevant even though there is no ordering of the updates in U such that every update is irrelevant for the program obtained by performing the previous updates in the order.

Example 9. Consider the DeLP program $\mathcal{P} = \langle \Pi, \Delta \rangle$ where $\Pi = \{(\sim a \leftarrow b),\ (\sim b \leftarrow a),\ f\}$ and $\Delta = \emptyset$. The initial status is such that $S_{\mathcal{P}}(f) = \text{IN}$ and $S_{\mathcal{P}}(a) = S_{\mathcal{P}}(b) = S_{\mathcal{P}}(\sim a) = S_{\mathcal{P}}(\sim b) = \text{UNDECIDED}$. Let $U = \{u_1, u_2\}$ be the set of updates such that $u_1 = +(a \prec f)$ and $u_2 = +(b \prec f)$. It is easy to check that U is irrelevant for $\mathcal{P}$ *(i.e.,* $S_{\mathcal{P}}(\alpha) = S_{U(\mathcal{P})}(\alpha)\ \forall \alpha \in Lit$). However, neither u_1 nor u_2 is irrelevant for the initial program $\mathcal{P}$, and thus no sequence of irrelevant updates exists. In particular, if u_1 is applied to $\mathcal{P}$ then $S_{\mathcal{P}}(a) = \text{UNDECIDED}$ and $S_{u_1(\mathcal{P})}(a) = \text{IN}$ (the status of a changes from UNDECIDED to IN). Analogously, if u_2 is applied to $\mathcal{P}$ then $S_{\mathcal{P}}(b) = \text{UNDECIDED}$ and $S_{u_1(\mathcal{P})}(b) = \text{IN}$ (in this case, the status of b changes from UNDECIDED to IN).

We now provide conditions under which the status of all the literals in *Lit* are guaranteed to remain unchanged after performing an update, and thus it does not need to be recomputed.

Lemma 2. *Let $\mathcal{P}$ be a DeLP program. An update $\pm r$ is irrelevant for $\mathcal{P}$ if at least one of the following conditions holds:*

(i) head(r) does not belong to $G(\{\pm r\}, \mathcal{P})$;
(ii) $\{head(r), \sim head(r)\} \cap CL(\{\pm r\}, \mathcal{P}) \neq \emptyset$;
(iii) $\exists \beta \in body(r)$ such that $\sim\beta \in CL(\{\pm r\}, \mathcal{P})$.

Observe that, in light of Proposition 1, Case (i) of Lemma 2 states that the status of all literals does not change if there is no derivation for the literal occurring in the head of the added (resp., deleted) rule in the updated program $+r(\mathcal{P})$ (resp., program $\mathcal{P}$). Case (ii) (resp., (iii)) states that the status of all literals does not change if the status of the head (resp., at least a body) literal α (resp., β), or its complementary literal $\sim\alpha$ (resp., $\sim\beta$), does not change, as it is inferred using derivations with strict rules only in both the initial and updated program.

Example 10. Consider the DeLP program $\mathcal{P}_1$ from our running example, and $u = +(\mathtt{a} \prec \mathtt{s})$. We have that u is irrelevant for $\mathcal{P}$ as case (ii) of Lemma 2 holds (*i.e.,* $\sim\mathtt{a} \in CL(\{u\}, \mathcal{P})$).

Lemma 2 can be used to check if an update $u = \pm r$ is irrelevant for a given program $\mathcal{P}$. However, in general we have a set U of updates and want to identify a subset of irrelevant updates. The following example shows a case where the whole set of updates is not irrelevant, but it contains an irrelevant update.

Example 11. Consider again the DeLP program $\mathcal{P}_1$ from our running example, where we have that $S_{\mathcal{P}_1}(\mathtt{s}) = S_{\mathcal{P}_1}(\mathtt{t}) = \text{IN}$, $S_{\mathcal{P}_1}(\mathtt{a}) = \text{OUT}$ and $S_{\mathcal{P}_1}(\sim\mathtt{i}) = \text{UNDECIDED}$. Let $U = \{u_1,\ u_2\}$ be a set of updates where $u_1 = +(\mathtt{s} \leftarrow \mathtt{t})$ and $u_2 = +(\mathtt{a} \prec \mathtt{s})$. Observe that U is not irrelevant for $\mathcal{P}_1$ since $S_{U(\mathcal{P}_1)}(\sim\mathtt{i}) = \text{IN}$ while it was UNDECIDED before performing the updates. The change in the status of $\sim\mathtt{i}$ is caused by the new argument $\langle \mathcal{A}_{10}, \sim\mathtt{i}\rangle = \langle\{(\sim\mathtt{i} \prec \sim\mathtt{a}, \mathtt{s})\}, \sim\mathtt{i}\rangle$ for $U(\mathcal{P}_1)$ and the fact that $\mathcal{A}_{10}$ is preferred to all the other arguments of the form $\langle \mathcal{A}, \mathtt{i}\rangle$. However, although U is not irrelevant for $\mathcal{P}_1$, U contains a set of irrelevant updates. Indeed, as shown in Example 10, u_2 is irrelevant for $\mathcal{P}_1$, and thus a set of irrelevant updates for $\mathcal{P}_1$ is $\{u_2\} \subseteq U$.

Given a DeLP program $\mathcal{P}$ and a set U of updates, function *Split* (see Function 1) non-deterministically selects a sequence of updates $[u_1, \ldots, u_n]$ such that every $u_i \in U$, with $i \in [1..n]$, and is irrelevant for the program $u_{i-1}(\ldots(u_1(\mathcal{P})))$. As stated in Theorem 4, this means that, using the result of Lemma 1, function *Split* provides a set $\{u_1, \ldots, u_n\} \subseteq U$ of updates that is irrelevant for $\mathcal{P}$.

Function 1 *Split*.

Input: DeLP program $\mathcal{P}$, set of updates U;
Output: A subset of U that consists of irrelevant updates for $\mathcal{P}$;
1: $L = [\,]$; done = *false*;
2: **while** *not* done **do**
3: done = *true*;
4: **if** $\exists\ u \in U$ satisfying the condition of Lemma 2 w.r.t. $\mathcal{P}$ **then**
5: done = *false*;
6: $\mathcal{P} = u(\mathcal{P})$;
7: remove u from U;
8: append (u, L);
9: **return** $\{u \mid u$ occurs in $L\}$

Theorem 4. *Let $\mathcal{P}$ be a DeLP program, U a set of updates, and U^{ir} a set of updates returned by function Split (with input $\mathcal{P}$ and U). Then (i) $U^{ir} \subseteq U$ is irrelevant for $\mathcal{P}$, and (ii) the computational complexity of Split is PTIME.*

Therefore, given a DeLP program $\mathcal{P}$, using function *Split* a given set of updates U can be partitioned into two disjoint sets of updates: U^{ir} that is irrelevant for $\mathcal{P}$, and $U^r = U \setminus U^{ir}$ that is a set of updates that have not been identified as irrelevant. Continuing from Example 11, by applying function *Split* to $\mathcal{P}_1$ with input $U = \{u_1, u_2\}$, we obtain $U^{ir} = \{u_2\}$ and $U^r = \{u_1\}$. It is worth noting that function *Split* is non-deterministic, as different applications over the same program $\mathcal{P}$ and set of updates U may yield different outcomes in general.

4.2. Dealing with relevant updates

We now consider the computation of the status of literals for sets of updates that have not been identified as irrelevant. An update that has not been identified as irrelevant will be called *relevant*.

In this section we focus on relevant updates only. Relevant updates are the only ones that may cause changes in the status of literals. Thus, to avoid wasted effort, we determine the subset of literals whose status needs to be recomputed after an update. Towards this end, we introduce the concept of *influenced set*, which consists of a subset of the literals that are *related* to a given set of updates w.r.t. a DeLP program. This definition is carried out by introducing a program $\mathcal{P}^*_U = \langle \Pi^*_U, \Delta^*_U\rangle$ obtained from $\mathcal{P}$ by rewriting rules after discarding some of them that, based on the given update U and on the status of literals in $\mathcal{P}$, are not needed to determine the subset of literals whose status may change after performing U (informally, discarding those rules can be viewed as performing an irrelevant update). The influenced set can then be defined by relying only on the literals related to U w.r.t. the new program $\mathcal{P}^*_U = \langle \Pi^*_U, \Delta^*_U\rangle$. The hypergraph of $\mathcal{P}^*_U$ is, in general, less connected, in the sense that a smaller set of literals is related to the update w.r.t. $\mathcal{P}^*_U$, and the missing connections essentially make unreachable the additional literals whose status does not change.

Before defining $\mathcal{P}^*_U$, we introduce some notation. Let $\mathcal{P} = \langle \Pi, \Delta\rangle$ be a DeLP program, U a set of (homogeneous, relevant) updates, and $\Gamma = CL(U, \mathcal{P})$. Let $\rho_\Gamma : \Pi \cup \Delta \to \Pi' \cup \Delta'$ be a (rule rewriting) function such that $\rho_\Gamma(r) = r'$ where r' is obtained from r by replacing every literal $\beta \in body(r) \cap \Gamma$ with a fresh literal β'_r; hence, $\rho_\Gamma(r) = r$ if no such β exists in r (herein, facts are viewed as rules with empty body, *i.e.,* $\rho_\Gamma(r) = r$ if r defines a fact).

We denote by $\mathcal{P}^*_U = \langle \Pi^*_U, \Delta^*_U\rangle$ the program derived from $\mathcal{P}$ by applying the following two steps:

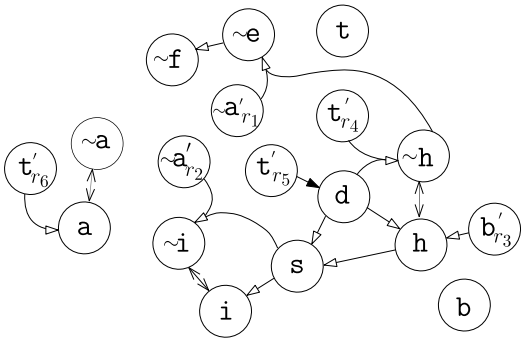


Fig. 2. $G(\mathcal{P}^*_{1_U})$ for program $\mathcal{P}^*_{1_U}$ from Example 12.

1. *Rule deletion.* Delete from $\mathcal{P}$ every rule whose head α is related to U w.r.t. $\mathcal{P}$ (*i.e.,* $\alpha \in \mathcal{R}(U, \mathcal{P})$) and has a body literal β such that $\sim\beta \in \Gamma$;
2. *Rule rewriting.* Let $\langle \overline{\Pi}, \overline{\Delta} \rangle$ be the program obtained from Step 1. The program $\mathcal{P}^*_U = \langle \Pi^*_U, \Delta^*_U \rangle$ is obtained from $\langle \overline{\Pi}, \overline{\Delta} \rangle$ by:
   (*i*) rewriting the rules of $\langle \overline{\Pi}, \overline{\Delta} \rangle$, that is, $\Pi^*_U \cup \Delta^*_U$ consists of $\rho_\Gamma(r)$ for each $r \in \overline{\Pi} \cup \overline{\Delta}$;
   (*ii*) adding the fact β'_r to Π^*_U for each rule $r \in \overline{\Pi} \cup \overline{\Delta}$ with literal $\beta \in$ *body(r)*, which is rewritten as r' with body literal β'_r (*i.e.,* $\rho_\Gamma(r)$ yields a rule r' different from r).

In the following, we use $\mathcal{F}(\mathcal{P}^*_U)$ to denote the set of fresh literals added to $\mathcal{P}^*_U$ in Step 2(*ii*), *i.e.,* $\mathcal{F}(\mathcal{P}^*_U)$ consists of literals β'_r such that β is rewritten by $\rho_\Gamma(r)$. The following examples illustrate the construction of $\mathcal{P}^*_U$.

Example 12. Consider the DeLP program $\mathcal{P}_1$ of Example 1 and the set of updates $U = \{-(\mathtt{d} \leftarrow \mathtt{t})\}$ of Example 7. To obtain the program $\mathcal{P}^*_{1_U} = \langle \Pi^*_{1_U}, \Delta^*_{1_U} \rangle$ we use $\Gamma = CL(U, \mathcal{P}_1) = \{\sim\mathtt{a}, \mathtt{b}, \mathtt{t}\}$. Thus, $\langle \Pi^*_{1_U}, \Delta^*_{1_U} \rangle$ is obtained as follows:

1. No rule is deleted at Step 1 since there is no rule whose head belongs to $\mathcal{R}(U, \mathcal{P})$ (that, as shown in Example 8, consists of all literals except literal $\mathtt{b}$) and having a body literal whose complementary literal is in Γ. Thus, $\overline{\mathcal{P}}_1 = \langle \overline{\Pi}_1, \overline{\Delta}_1 \rangle = \langle \Pi_1, \Delta_1 \rangle$.
2. At Step 2(*i*), rules $r_1 = (\sim\mathtt{e} \prec \sim\mathtt{h}, \sim\mathtt{a})$, $r_2 = (\sim\mathtt{i} \prec \sim\mathtt{a}, \mathtt{s})$, $r_3 = (\mathtt{h} \prec \mathtt{b})$, $r_4 = (\sim\mathtt{h} \prec \mathtt{d}, \mathtt{t})$, $r_5 = (\mathtt{d} \leftarrow \mathtt{t})$, and $r_6 = (\mathtt{a} \prec \mathtt{t})$ that contain a body literal in Γ are rewritten in $\mathcal{P}^*_{1_U}$ as $\rho_\Gamma(r_1) = (\sim\mathtt{e} \prec \sim\mathtt{h}, \sim\mathtt{a}'_{r_1})$, $\rho_\Gamma(r_2) = (\sim\mathtt{i} \prec \sim\mathtt{a}'_{r_2}, \mathtt{s})$, $\rho_\Gamma(r_3) = (\mathtt{h} \prec \mathtt{b}'_{r_3})$, $\rho_\Gamma(r_4) = (\sim\mathtt{h} \prec \mathtt{d}, \mathtt{t}'_{r_4})$, $\rho_\Gamma(r_5) = (\mathtt{d} \leftarrow \mathtt{t}'_{r_5})$, and $\rho_\Gamma(r_6) = (\mathtt{a} \prec \mathtt{t}'_{r_6})$ respectively. All the other rules (including facts) belong to $\mathcal{P}^*_{1_U}$ as well (the rewriting step does not change them). Moreover, at Step 2(*ii*) the facts $\sim\mathtt{a}'_{r_1}$, $\sim\mathtt{a}'_{r_2}$, $\mathtt{b}'_{r_3}$, $\mathtt{t}'_{r_4}$, $\mathtt{t}'_{r_5}$, and $\mathtt{t}'_{r_6}$ are added to Π^*_U.

The hypergraph $G(\mathcal{P}^*_{1_U})$ for the program $\mathcal{P}^*_{1_U}$ is shown in Fig. 2.

We now introduce the notion of set of influenced literals. It allows us to further restrict the set of literals that need to be recomputed after performing an update by obtaining a set possibly smaller than $\mathcal{R}(U, \mathcal{P})$. Intuitively, the influenced set contains the literals related to a set of updates U w.r.t. the program $\mathcal{P}^*_U$, but after rewriting the updates in the language of $\mathcal{P}^*_U$ as follows.

Given a set U of (homogeneous, relevant) updates, we use U^* to denote the set of updates obtained from U as follows. Let $\Gamma = CL(U, \mathcal{P})$ and ρ_Γ be the (rule rewriting) function defined earlier. Then, $U^* = \{\pm\rho_\Gamma(r) \mid \pm r \in U\} \cup \{\pm\beta'_r \leftarrow \mid \pm r \in U, \beta \in$ *body(r)* $\cap\, \Gamma\}$. For instance, the update $U = \{-(\mathtt{d} \leftarrow \mathtt{t})\}$ of Example 12 is rewritten as $U^* = \{-(\mathtt{d} \leftarrow \mathtt{t}'_{r_5}), -(\mathtt{t}'_{r_5} \leftarrow)\}$ w.r.t. the program $\mathcal{P}^*_{1_U}$ whose hypergraph is shown in Fig. 2. Observe that for each update in U consisting of the addition/removal of a rule r that does not contain a body literal $\beta \in \Gamma$, then $\pm\rho_\Gamma(r) = \pm r$ is in U^* (no addition/removal concerning β'_r is in U^*). Thus, $U^* = U$ if no literal $\beta \in \Gamma$ occurs in the body of a rule that is added/removed by an update in U.

We are now ready to define the influenced set.

Definition 7 *(Influenced Set).* Given a DeLP program $\mathcal{P}$ and a set of (homogeneous, relevant) updates U, the set of literals that are influenced by U w.r.t. $\mathcal{P}$ is defined as $\mathcal{I}(U, \mathcal{P}) = \mathcal{R}(U^*, \mathcal{P}^*_U) \setminus \mathcal{F}(\mathcal{P}^*_U)$.

It is worth noting that the definition of influenced set $\mathcal{I}(U, \mathcal{P})$ behaves similarly to the definition of $\mathcal{R}(U, \mathcal{P})$ of literals related to U w.r.t. a program $\mathcal{P}$. The difference is that the computation of influenced literals is carried out on the program $\mathcal{P}^*_U$ for $\mathcal{P}$ and U, which allows us to take into account the status of some literals before the update is performed (since $\mathcal{P}^*_U$ is defined by using $\Gamma = CL(U, \mathcal{P})$).

Example 13. Consider the DeLP program $\mathcal{P}_1$ of Example 1 and the set of updates $U = \{-(\mathtt{d} \leftarrow \mathtt{t})\}$. The set of literals that are influenced by U w.r.t. $\mathcal{P}_1$ is obtained by considering the set $\mathcal{R}(U^*, \mathcal{P}_U^*)$ of literals related to $U^* = \{-(\mathtt{d} \leftarrow \mathtt{t}'_{r_5}), -(\mathtt{t}'_{r_5} \leftarrow)\}$ w.r.t. the program $\mathcal{P}_1{}_U^*$ of Example 12, whose hypergraph is shown in Fig. 2. Specifically, $\mathcal{R}(U^*, \mathcal{P}_U^*) = \{\mathtt{d}, \mathtt{t}'_{r_5}, \mathtt{h}, \sim\mathtt{h}, \mathtt{s}, \sim\mathtt{e}, \mathtt{i}, \sim\mathtt{i}, \sim\mathtt{f}\}$, and since $\mathcal{F}(\mathcal{P}_U^*) = \{\mathtt{t}'_{r_5}\}$, we have that $\mathcal{I}(U, \mathcal{P}) = \{\mathtt{d}, \mathtt{h}, \sim\mathtt{h}, \mathtt{s}, \sim\mathtt{e}, \mathtt{i}, \sim\mathtt{i}, \sim\mathtt{f}\}$ which is contained in $\mathcal{R}(U, \mathcal{P})$ given in Example 8.

The following proposition states that the status of the literals of $\mathcal{P}$ does not change in $\mathcal{P}_U^*$, and in order to obtain the updated status of the literals of $\mathcal{P}$ it is possible to focus only on the program $U^*(\mathcal{P}_U^*)$.

Proposition 3. *For any DeLP program $\mathcal{P}$ and set of (homogeneous, relevant) updates U, (i) $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}_U^*}(\alpha)$ $\forall \alpha \in Lit_{\mathcal{P}}$, and (ii) $S_{U(\mathcal{P})}(\alpha) = S_{U^*(\mathcal{P}_U^*)}(\alpha) \forall \alpha \in Lit_{U(\mathcal{P})}$.*

Our notion of influenced set is conceptually similar to the influenced set recently introduced by [20,21] in the context of abstract argumentation [10] in order to identify arguments whose status may change after performing an update (under a given argumentation semantics). Although the aim is analogous, here we deal with the incremental computation of the status of *structured* arguments defined by both strict and defeasible rules. Consequently, looking at the definition of influenced set for a DeLP program, the situation becomes more involved than in the case of abstract argumentation; this is because, when determining a portion of the hypergraph that contains literals whose status may change, we need to take into account the presence of strict and defeasible rules and the specific semantics of DeLP. In particular, in this setting, the status of a literal α may be influenced (apart from the literals from which it is reachable) by the literals that are reachable from α using strict hyper-edges only. The following example illustrates such situations.

Example 14. Let $\mathcal{P}_{ex} = \langle \Pi_{ex}, \Delta_{ex} \rangle$ be a DeLP program where $\Pi_{ex} = \{a,\ b,\ e \leftarrow c,\ \sim e \leftarrow d\}$ and $\Delta_{ex} = \{c \prec\!\!\!- a\}$, and let $U = \{+(d \prec\!\!\!- b)\}$ be an update yielding the updated DeLP program $\mathcal{P}'_{ex}$. The influenced set is $\mathcal{I}(U, \mathcal{P}_{ex}, S_{\mathcal{P}_{ex}}) = \{d,\ \sim e,\ e, c\}$. In particular, c is included in the influenced set by navigating backward the (hyper)edge corresponding to the strict rule $e \leftarrow c$, while the other literals are reached by forward reachability. Note that including c in $\mathcal{I}(U, \mathcal{P}_{ex}, S_{\mathcal{P}_{ex}})$ is important as its status changes ($S_{\mathcal{P}'_{ex}}(c) = \text{UNDECIDED}$, it was IN w.r.t. $\mathcal{P}_{ex}$).

Using the influenced set, we can identify preserved literals even among those that are in $\mathcal{R}(U, \mathcal{P})$, thus strengthening the result of Theorem 3 concerning literals related to an update w.r.t. a given program. The following result follows from Definition 7, Proposition 3, and Theorem 3,

Corollary 2 *(Influenced and Preserved Literals). Let $\mathcal{P}$ be a DeLP program, U a set of (homogeneous) updates, $\mathcal{P}' = U(\mathcal{P})$, and $G(\mathcal{P}') = \langle N', H' \rangle$ the updated labeled hypergraph. Then, every literal $\alpha \in N'$ such that $\alpha \notin \mathcal{I}(U, \mathcal{P})$ is preserved, that is $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$.*

In the rest of the paper, when referring to preserved literals, we use the result of Corollary 2 for the literals not belonging to the influenced set, and the result of item (i) in Theorem 3 for the literals in the deductive closure of facts and strict rules before and after the update. That is, in the following, a literal α belonging to the updated hypergraph is said to be *preserved* if either (i) $\{\alpha, \sim\alpha\} \cap CL(U, \mathcal{P}) \neq \emptyset$, or (ii) $\alpha \notin \mathcal{I}(U, \mathcal{P})$.

The status of a literal for which there is no argument in the (updated) program may depend only on the status of its complementary literal—we call such literals *inferable*. Using the hypergraph of updated programs, we can define inferable literals as follows.

Definition 8 *(Set of Inferable and Core Literals).* Let $\mathcal{P}$ be a DeLP program, $S_{\mathcal{P}}$ the status of the literals of $\mathcal{P}$, U a set of (homogeneous) updates, $\mathcal{P}' = U(\mathcal{P})$, and $G(\mathcal{P}') = \langle N', H' \rangle$.

- The set of inferable literals for U w.r.t. $\mathcal{P}$ is $Infer(U, \mathcal{P}) = Lit_{\mathcal{P}'} \setminus N'$.
- The set of core literals for U w.r.t. $\mathcal{P}$ is $Core(U, \mathcal{P}) = (\mathcal{I}(U, \mathcal{P}) \setminus Infer(U, \mathcal{P})) \cap Lit_{\mathcal{P}'}$.

The core literals for a set of updates U are those in $Lit_{\mathcal{P}'}$ that are influenced but are not inferable. Literals that are not in $Lit_{\mathcal{P}'}$ (*e.g.,* a literal occurring only in the head of a rule deleted by the update) are not considered as their status is readily determined to be UNDECIDED.

Example 15. Continuing with Example 13, consider again $\mathcal{P}_1$ from Example 1, whose hypergraph is shown in Fig. 1(a), and the update $U = \{-(\mathtt{d} \leftarrow \mathtt{t})\}$ that yields program $U(\mathcal{P}_1)$, whose hypergraph is shown in Fig. 1(b). Then, $Infer(U, \mathcal{P}_1) = \{\mathtt{d}, \sim\mathtt{d}, \mathtt{h}, \sim\mathtt{h}, \mathtt{e}, \sim\mathtt{e}, \sim\mathtt{f}\}$. Moreover, we have that $Core(U, \mathcal{P}_1) = (\{\mathtt{d}, \mathtt{h},\ \sim\mathtt{h},\ \mathtt{s},\ \sim\mathtt{e},\ \mathtt{i},\ \sim\mathtt{i},\ \sim\mathtt{f}\} \setminus \{\mathtt{d}, \sim\mathtt{d}, \mathtt{h}, \sim\mathtt{h}, \mathtt{e}, \sim\mathtt{e}, \sim\mathtt{f}\}) \cap Lit_{\mathcal{P}'} = \{\mathtt{h}, \mathtt{s}, \mathtt{i}, \sim\mathtt{i}\}$.

Algorithm 1 Dynamic DeLP-Solver.

Input: DeLP program $\mathcal{P}$, initial status $S_{\mathcal{P}}$, set of homogeneous updates U.
Output: Status $S_{\mathcal{P}'}$ w.r.t. the updated program $\mathcal{P}' = U(\mathcal{P})$.

1: $S_{\mathcal{P}'}(\alpha) = S_{\mathcal{P}}(\alpha) \quad \forall \alpha \in Lit$;
2: Let $U^{ir} = Split(\mathcal{P}, U)$ and $U^{r} = U \setminus U^{ir}$ (cf. Theorem 4);
3: **if** $U^{r} = \emptyset$ **then**
4: **return** $S_{\mathcal{P}'}$;
5: Let $\hat{\mathcal{P}} = U^{ir}(\mathcal{P})$;
6: Let $U^{new} = \{u \mid u \in U^{r} \wedge \{head(r), \sim head(r)\} \cap Lit_{\hat{\mathcal{P}}} = \emptyset\}$
7: Let $U^{k} = U^{r} \setminus U^{new}$
8: **for** $+r \in U^{new}$ **do**
9: $S_{\mathcal{P}'}(head(r)) \leftarrow$ DeLP-Solver$(\mathcal{P}', head(r))$;
10: **if** $U^{k} = \emptyset$ **then**
11: **return** $S_{\mathcal{P}'}$;
12: **for** $\alpha \in Core(U^{k}, \hat{\mathcal{P}})$ **do**
13: $S_{\mathcal{P}'}(\alpha) \leftarrow$ DeLP-Solver$(\mathcal{P}', \alpha)$;
14: **for** $\alpha \in Infer(U^{k}, \hat{\mathcal{P}})$; **do**
15: **if** $S_{\mathcal{P}'}(\sim\alpha) =$ IN
16: **then** $S_{\mathcal{P}'}(\alpha) \leftarrow$ OUT;
17: **else** $S_{\mathcal{P}'}(\alpha) \leftarrow$ UNDECIDED;
18: **for** $\alpha \in Lit \setminus Lit_{\mathcal{P}'}$ **do**
19: $S_{\mathcal{P}'}(\alpha) =$ UNDECIDED;
20: **return** $S_{\mathcal{P}'}$.

Observe that the definition of inferable literals is specifically designed for DeLP programs and no equivalent definition can be conceived for abstract argumentation. As a consequence, the definition of core literals, which restricts the set of influenced literals (by deleting inferable literals) is also specific to DeLP programs.

The following theorem states that the relationship between inferable literals and other literals is as follows: the status of an inferred literal w.r.t. the updated program can be either OUT or UNDECIDED, and if it is OUT it is entailed by the status of a core or preserved literal that is IN.

Theorem 5 *(Status of Inferable Literals). Let $\mathcal{P}$ be a DeLP program, U a set of (homogeneous) updates, and $\mathcal{P}' = U(\mathcal{P})$. For each literal $\alpha \in Infer(U, \mathcal{P})$, it holds that:*

- $S_{\mathcal{P}'}(\alpha) =$ OUT *iff* $S_{\mathcal{P}'}(\sim\alpha) =$ IN *and either* $\sim\alpha \in Core(U, \mathcal{P})$ *or* $\sim\alpha$ *is preserved;*
- $S_{\mathcal{P}'}(\alpha) =$ UNDECIDED *otherwise.*

4.3. Incremental algorithm

We now introduce our incremental algorithm (Algorithm 1) that, given a program $\mathcal{P}$, a set U of homogeneous updates, and the status of literals $S_{\mathcal{P}}$, computes the status of literals $S_{U(\mathcal{P})}$ w.r.t. the updated program $U(\mathcal{P})$. Observe that if the set of updates to be performed is not homogeneous, as it happens for the updates considered in the experiments in Section 5, then it suffices to split U into U^{-} and U^{+}, and call Algorithm 1 twice with inputs $\langle \mathcal{P}, S_{\mathcal{P}}, U^{-} \rangle$ and $\langle U^{-}(\mathcal{P}), S_{U^{-}(\mathcal{P})}, U^{+} \rangle$ to get $S_{U^{+}(U^{-}(\mathcal{P}))}$, which coincides with $S_{U(\mathcal{P})}$.

Algorithm 1 works as follows. First, by default, the status of the literals w.r.t. the updated program is assigned to that of the initial program (Line 1). Then, using function $Split(\mathcal{P}, U)$ and the result of Theorem 4, the set U of updates is partitioned into the two disjoint sets of updates, U^{ir} and U^{r}, containing irrelevant and relevant updates, respectively (Line 2). If $U^{r} = \emptyset$ then U is irrelevant for $\mathcal{P}$, and thus the updated status, which coincides with the initial one, is returned (Line 4). Otherwise, the DeLP program $\hat{\mathcal{P}} = U^{ir}(\mathcal{P})$ is computed by applying only the irrelevant updates (Line 5)—the status of the literals w.r.t. $\hat{\mathcal{P}}$ can be trivially computed by copying that of $\mathcal{P}$.

Next, U^{r} is split into the sets U^{new} and U^{k} containing, respectively, the updates whose head is a *new* literal that does not occur in $\hat{\mathcal{P}}$ (and thus not occurring in $\mathcal{P}$) and the remaining ones, whose head is *known* (lines 6–7). For the literals not in the language of the program $\hat{\mathcal{P}}$, the status w.r.t. $\mathcal{P}'$ is computed by calling the DeLP-Solver[5] with input $\mathcal{P}'$ and the new literal (Line 9). If U^{k} is empty, then the initial status updated by changing only that of the new literals is returned at Line 9. Otherwise, the status of the literals is updated by first calling the solver at Lines 12–13 for the core literals, and then using Theorem 5 to derive the status of inferable literals at Lines 14–17. The status of preserved literals remains as assigned in Line 1. Finally, the status of literals no longer belonging to the updated program (due to rule removal) is set to UNDECIDED, and the updated status $S_{\mathcal{P}'}$ for all literals of the updated program $\mathcal{P}'$ is returned.

The following proposition states that Algorithm 1 is sound and complete.

[5] A web client for DeLP is available at http://lidia.cs.uns.edu.ar/delp_client/index.php. In the experiments, we used a standalone implementation of the DeLP solver that runs locally.

Proposition 4. *Let $\mathcal{P}$ be a DeLP program, U a set of homogeneous updates, and $\mathcal{P}' = U(\mathcal{P})$. For each $\alpha \in Lit$, Algorithm 1 returns $S_{\mathcal{P}'}(\alpha)$.*

It is easy to see that the time complexity of Algorithm 1 is $O((|Core(U, \mathcal{P})| + |U^{new}|) \times D(\mathcal{P}, \alpha))$, where $D(\mathcal{P}, \alpha)$ is the cost of a call to the DeLP-solver for computing the status of a literal α w.r.t. $\mathcal{P}$—Theorem 2 entails that solving this problem is hard in general. Thus, theoretically, Algorithm 1 has the same worst-case complexity as total recomputation, as $Core(U, \mathcal{P})$ may consist of the whole set of literals whose status needs to be recomputed. However, as we experimentally show in Section 5, in practice the incremental approach turns out to be much more efficient than recomputing everything from scratch.

5. Implementation and experiments

In this section we report on a set of experiments designed to compare our incremental approach (Algorithm 1) against recomputation from scratch (*i.e.,* the direct computation of the status of all the literals in an updated DeLP program using the DeLP-Solver). We performed experiments that aimed at evaluating both the efficiency and effectiveness of our approach on two datasets:

- REAL, consisting of a set of DeLP programs built from the knowledge base used by the music recommender system proposed in [25].
- SYN, consisting of a set of DeLP programs generated by the benchmark generator introduced in Section 5.2.1. To the best of our knowledge, this is one of the first attempts to produce benchmarks for structured argumentation, along with [26] where they were generated for ABA [6].

In the rest of this section, we first describe REALand then present the results on efficiency (Section 5.1.1) and effectiveness (Section 5.1.2) of our technique for this dataset. Then, after introducing the benchmark generator, we discuss the results on efficiency (Section 5.2.2) and effectiveness (Section 5.2.3) for SYN.

All the experiments were carried out on a computer with an Intel Core i7-3630QM 2.4 GHz processor with 8GB of RAM.

5.1. Experiments on the REAL dataset

REALcontains a set of DeLP programs for music recommendation [25] that are built from a knowledge base storing users' preferences for music tracks and artists (*e.g.,* tracks/artists that users like listening to). According to their recent listened tracks, the system using such programs aims to support the recommendation, in an argumentative manner, of songs to users. The recommendation process is based on DeLP programs consisting of rules modeling so-called postulates that represent conditions that need to be met for a specific track to be recommended to a given user (*e.g.,* a user may like a song if he likes another song by the same artist). Twelve postulates are listed in [25]. We can think of postulates as "general" DeLP rules using predicates with variables ranging over the sets of 'objects' (*e.g.,* users, songs, artists) in the knowledge base. Hence, every postulate results in a set of DeLP rules generated after binding variables to objects in the knowledge base.

The underlying knowledge base consists of 3,835 artists, 793 users, 13,104 tracks, and 20,753 positive and 889 negative user preferences (*e.g.,* a given user likes/dislikes a given song). In particular, the presence of both positive and negative preferences results in conflicts between arguments in a dialectical inference mechanism, making the recommendation process non-trivial. For this reason, in REALwe focused on a set of artists for which there is at least one positive and one negative preference towards one of their songs, thus discarding DeLP programs modeling trivial recommendations (carried out without dealing with conflicting information). Therefore, we considered 30 DeLP programs, divided into 3 groups of increasing size. In particular, REAL-1(resp., REAL-3, REAL-5) consists of 10 DeLP programs each focusing on 1 (resp., 3, 5) authors. The programs in set REAL-1(resp., REAL-3, REAL-5) have, on average, 82 (resp., 697, 2,669) literals, of which 24 (resp., 369, 700) are negated literals, 47 (resp., 494, 1,463) facts, and 133 (resp., 1,433, 5,387) rules, of which 85 (resp., 939, 3,924) are defeasible, having at most two literals in rule bodies.

5.1.1. Efficiency

We considered the three groups of DeLP programs described above (REAL-1, REAL-3, and REAL-5) and different types of updates (addition and deletion of strict/defeasible rules). As observed in Section 4.3, sets U of non-homogeneous updates are split into U^- and U^+, and Algorithm 1 is called twice by first performing negative updates and then the positive ones. In what follows, $n \in \{1, 4, 8, 16\}$ denotes the number of simultaneous updates. For each program in REAL, we have generated 10 different sets of updates of cardinality n that are uniformly distributed among the different types of operations (addition/deletion) and rules (strict/defeasible).

The results of the experiments on REALare reported in Fig. 3, where the diagram on the left-hand side (resp., middle, right-hand side) refers to REAL-1(resp., REAL-3, REAL-5). Each diagram consists of 5 boxes, one for each value of $n \in \{1, 4, 8, 16\}$ plus one for the total recomputation from scratch (denoted with S); each box consists of 100 observed running times resulting from 10 sets of updates performed over 10 programs for $n \in \{1, 4, 8, 16\}$, and from 100 uniformly sampled running times for the computation from scratch. Note that each box provides a five-number summary of observed

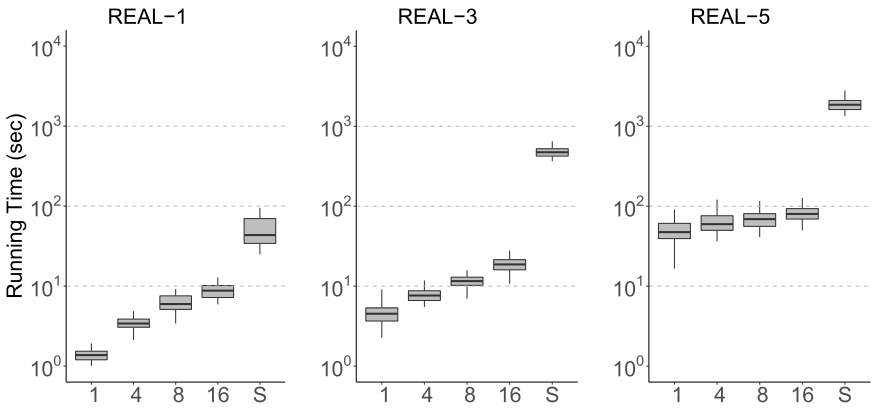


Fig. 3. Running times (in seconds, log scale) for the incremental computation of the status of all literals of the DeLP programs in REAL-1, REAL-3, REAL-5(left-hand side, middle, right-hand side, respectively) for 1, 4, 8, or 16 updates and for the total recomputation from scratch (S).

running times showing five essential results: the minimum running time observed, the first quartile, the median, the third quartile, and the maximum running time observed. Thus, boxes give information about both the central tendency and the spread of the running times observed for each given configuration. Two-sample t-tests indicate that the difference between the observed running times is statistically significant: such tests yielded a *p-value* very close to zero ($p \leq 0.01$) for each pair of adjacent boxes in each diagram of Fig. 3, except for the pair corresponding to 4 and 8 updates in REAL-5, where $p = 0.10$.

Analyzing Fig. 3, we can draw the following conclusions:

- The incremental computation approach outperforms the full recomputation from scratch in all the settings considered. On average, the incremental approach takes only 25 seconds, while the full recomputation takes more than 13 minutes (809 seconds); therefore, considering every setting, the incremental approach is about 32 times faster than full recomputation.
- In general, the improvement increases as the size of the programs increases.
  - For REAL-1, the incremental approach takes only 3.8 seconds, while full recomputation takes 51.9 seconds, on average.
  - For REAL-3, the incremental approach takes only 9.6 seconds, while full recomputation takes about eight minutes (487.2 seconds), on average.
  - Finally, for REAL-5, the incremental approach takes just over one minute (64.3 seconds), while full recomputation takes about half an hour (1898.6 seconds), on average.
- Although the running time of the incremental approach increases when the number of updates performed simultaneously increases, the average running time per update decreases when more updates are performed simultaneously. Specifically, the average running time for 4 (resp., 8, 16) updates is only 2.6 (resp., 4.2, 5.9) times that for computing one update.

In summary, the incremental computation generally outperforms the full recomputation from scratch, and the improvement is more significant when the size of the programs increases; moreover, the average running time per update decreases when more updates are performed simultaneously.

In order to provide further insights into the performance of our technique, we also analyzed how the improvement of the incremental approach depends on the topology of the hypergraph representing a program, and in particular on the *relatedness* of literals w.r.t. updates, that is, the percentage of literals related to a set of updates w.r.t. a program (*cf.* Definition 6). Fig. 4 shows how that improvement (*i.e.,* ratio between the running time of the computation from scratch and the running time of the incremental approach, for a given configuration) varies when the relatedness coefficient (rounded to the nearest integer) increases. We note that Fig. 4 shows that relatedness is an important topological property of the hypergraph that substantially affects the improvement, as it gives a measure of the part of the program that is of interest for the update. Specifically, the improvement decreases when the percentage of related literals increases, though it remains above one for all the percentages of related literals observed. In the context of AF, similar results have been discussed in [24,27] showing that the improvement yielded by incremental approaches depends on the graph's topology.

Furthermore, instead of applying all the updates in U at once, in the sequential approach we applied every update in U one after another in a stepwise fashion, invoking Algorithm 1 with input consisting of a single update in the sequence. Table 1 shows the average running times for the same sets of updates performed either simultaneously or sequentially. Specifically, the sequential approach's running times are the average of 8 uniformly sampled sequences out of the 24 possible ones that can be obtained by permuting the elements in a set of 4 updates. Although the performance of the sequential approach is comparable with that of the parallel one on average for REAL-1(smallest dataset), the sequential approach takes almost twice the time of the parallel one for REAL-3and REAL-5.

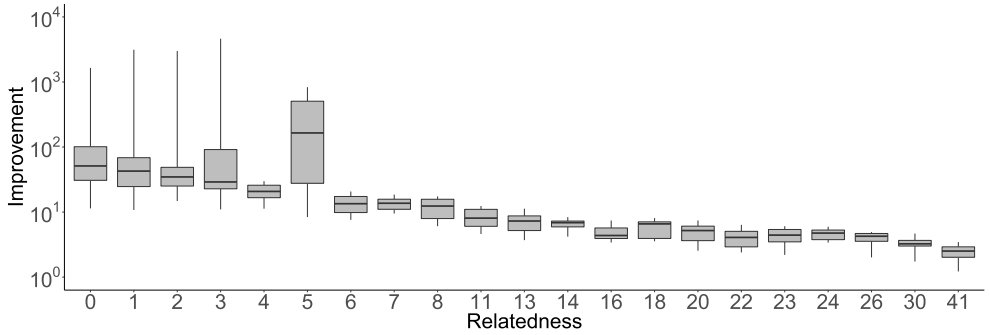


Fig. 4. Improvement (log scale) versus relatedness for the DeLP programs in REAL(percentages of related literals are rounded to the nearest integers; missing numbers on the x-axis are due to unobserved relatedness coefficients).

Table 1
Average running times (in seconds) for the incremental computation of the status of all literals of the DeLP-programs in REALfor 4 updates performed either simultaneously or sequentially; performance for total recomputations are also reported.

	Data set		
	REAL-1	REAL-3	REAL-5
Sequentially	3.9	15.9	128.3
Simultaneously	3.6	7.9	64.4
Computation from scratch	51.4	478.9	1890.4

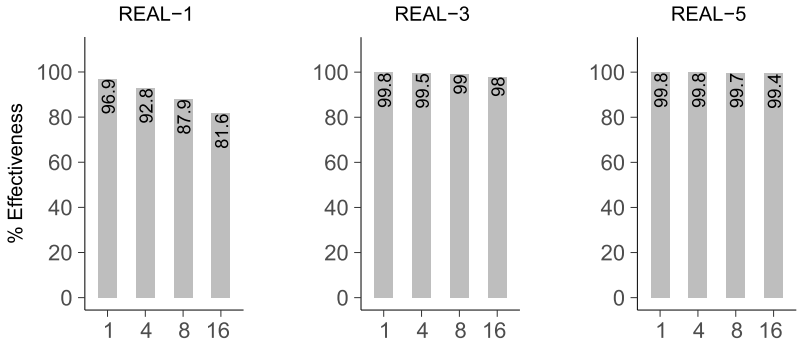


Fig. 5. Average effectiveness for the incremental computation of the status of all literals of the DeLP programs in REAL-1, REAL-3, REAL-5(left-hand side, middle, right-hand side, respectively) for 1, 4, 8, or 16 updates.

5.1.2. Effectiveness

Given a set of updates U and a program $\mathcal{P}$, we define the effectiveness $E(U,\mathcal{P})$ of the incremental computation w.r.t. U and $\mathcal{P}$ as follows. $E(U,\mathcal{P})$ is 100% if we are able to recognize that U is irrelevant, that is, function *Split* returns the whole set of updates taken as input, for both U^+ and U^- (*cf.* Line 2 of Algorithm 1). Otherwise (either U is relevant or we are not able to recognize that it is irrelevant), our approach identifies

$$E(U,\mathcal{P}) = 1 - \frac{|Core(U,\mathcal{P}) \cup Infer(U,\mathcal{P})|}{|Lit_{U(\mathcal{P})}|}$$

where $U(\mathcal{P}) = U^+(U^-(\mathcal{P}))$ and $Core(U,\mathcal{P}) = Core(U^-,\mathcal{P}) \cup Core(U^+,U^-(\mathcal{P}))$, and $Infer(U,\mathcal{P}) = Infer(U^-,\mathcal{P}) \cup Infer(U^+, U^-(\mathcal{P}))$ as the percentage of literals of the DeLP program whose status does not need to be recomputed by Algorithm 1. Given that $(Core(U,\mathcal{P}) \cup Infer(U,\mathcal{P})) \subseteq Lit_{U(\mathcal{P})}$, we have that as the difference between $Core(U,\mathcal{P}) \cup Infer(U,\mathcal{P})$ and $Lit_{U(\mathcal{P})}$ becomes larger, $E(U,\mathcal{P})$ increases and the technique is more effective.

Fig. 5 shows the average effectiveness for REAL-1, REAL-3, REAL-5for 1, 4, 8, and 16 updates. The effectiveness varies from 99.8% for 1 update performed on REAL-5(*i.e.,* only 0.2% of the literals are considered for recomputing the warrant status of the whole set of literals of the program) to 81.6% for 16 updates in REAL-1. The average effectiveness remains high (more than 96.9%) for 1 update, and in general, it increases when the size of the DeLP programs increases or the number of updates decreases. However, even for the cases where the effectiveness is relatively low (compared with the values obtained in general), Fig. 3 shows that the wasted effort during recomputation does not change the fact that the incremental approach outperforms the total recomputation in all cases.

Table 2
Parameters for benchmark DeLP programs generated using Algorithm 2.

Series name	Values for fixed parameters						Values for the parameter that varies	
	N_F	N_H	N_R	N_S	N_M	K		
SYN-K	40	100	19	3	4	K	$K \in$	$\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$
SYN-N_H	40	N_H	19	3	4	10	$N_H \in$	$\{80, 90, 100, 110, 120\}$
SYN-N_R	40	100	N_R	3	4	10	$N_R \in$	$\{19, 23, 27, 31, 35, 39\}$

In summary, the experiments on REALshowed that the incremental approach is quite effective in that it often computes only the status of the literals whose status changes after an update, and it is always faster than total recomputation—*i.e.*, the overhead of applying the technique always pays off. As we show in the next section, the same result is also observed for synthetic data.

5.2. Experiments on the synthetic dataset

In this section, we report on a set of experiments performed over a synthetic dataset consisting of three series of DeLP programs obtained by the benchmark generator introduced in Section 5.2.1. Table 2 summarizes the main properties of the series obtained by the benchmark generator; before describing the benchmark generator, we explain these parameters.

To build derivations, we associate rules and literals of a DeLP program $\mathcal{P}$ to a level defined as follows: (*i*) the level of a rule r is equal to the maximum level of literals occurring in its body plus 1 (rules defining facts are assumed to be of level 1); (*ii*) the level of a literal is equal to the maximum level of rules having the literal in the head (literals defined by facts only have level 1). We also assume that the number of negated literals is equal to the number of positive literals multiplied by a factor λ (in our experiments $\lambda = 0.1$).

Every series in Table 2 is characterized by a tuple $(N_F, N_H, N_R, N_S, N_M, K)$, where:

1. N_F is the number of facts in $\mathcal{P}$.
2. N_H (resp., $N_H \cdot \lambda$) is the number of distinct positive (resp., negative) literals used as head of at least one rule in $\mathcal{P}$.
3. N_R is the maximum number of (strict or defeasible) rules defining a literal occurring in the head of some rule.
4. N_S ($\leq N_R$) is the maximum number of strict rules defining a literal occurring in the head of some rule.
5. N_M is the maximum number of literals occurring in each rule's body.
6. K is the maximum level associated with literals occurring in the program.

It is worth noting that for uniform or normal distributions (*i*) the average number of rules (resp., strict rules) defining a literal is $\frac{N_R+1}{2}$ (resp. $\frac{N_S+1}{2}$), (*ii*) the average number of rules of $\mathcal{P}$ is $(1+\lambda) \cdot N_H \cdot \frac{N_R+1}{2}$, and (*iii*) the average number of strict (resp., defeasible) rules of $\mathcal{P}$ is $(1+\lambda) \cdot N_H \cdot \frac{N_S+1}{2}$ (resp., $(1+\lambda) \cdot N_H \cdot \frac{N_R-N_S}{2}$).

To make Table 2 concise, the last column defines the variability of one parameter after having fixed the remaining ones. For instance, in the first line we have that the maximum level K varies in the interval $[2, 10]$. Thus, the first line describes the characteristics of 9 types of distinct programs, whereas the whole table summarizes the characteristics of the 20 distinct types of programs we have used in our experiments.

5.2.1. Benchmark generation

The programs described above are generated using Algorithm 2, which takes as input a tuple of (feasible) parameters $(N_F, N_H, N_R, N_S, N_M, K, \lambda)$ and generates a DeLP program compliant with them.

Algorithm 2 starts by first defining the sets of positive and negative literals H_1 and $\overline{H}_1$ of the DeLP program $\mathcal{P}$, that could be selected as facts of $\mathcal{P}$ (Lines 1 and 2). At Line 3 a number N_F of facts are randomly selected among positive and negative literals. The function *subset* is designed to avoid both the selection of two complementary literals and the selection of the same literal twice. The selected facts are then added to the strict part Π of the program $\mathcal{P}$ at Line 3. Next, the set Δ of defeasible rules is initialized at Line 4. Then, the algorithm generates rules and proceeds level by level. Before that, the number N_H^* of literals that will occur in the head of rules having the same level is defined at Line 5. In Lines 6–23, the literals associated with level i and the rules defining them are generated. To this end, firstly positive and negative literals that will be defined at the current level i are defined at Lines 7 and 8, respectively. Then, for each literal α belonging to the current level i, the rules defining it are defined in Lines 10–23. At this point, the value N_R^* (resp. N_S^*) defining the number of rules (resp. strict rules) having α in the head is computed (Lines 10 and 11). Then, rules are iteratively defined from Line 13 to 23. Before generating the N_R^* rules having α as head, the sets of strict rules Π_α and defeasible rules Δ_α are introduced and initialized to $\emptyset$ (Line 12). Next, the size N_M^* of its body is determined (Line 14), where

$$C_{\Sigma_{l=1}^{i-1}|H_l|, N_M^*} = \binom{\Sigma_{l=1}^{i-1}|H_l|}{N_M^*}$$

denotes the number of feasible subsets of $\cup_{l=1}^{i-1} H_l$ having size N_M^*, whereas the body B of the rule is computed at Line 15. Here it is assumed that the *subset* function returns a set of N_M^* literals that have been introduced at the previous levels

Algorithm 2 DeLP programs Generator.

Input: N_F, N_H, N_R, N_S, N_M, K, λ.
Output: DeLP program (Π, Δ) compliant with the input parameters, if it exists.

1: $H_1 = \{\alpha_j \mid 1 \leq j \leq N_F\}$;
2: $\overline{H}_1 = \{\sim\alpha_j \mid \alpha_j \in H_1\}$;
3: $\Pi = \{\alpha \leftarrow \mid \alpha \in subset(H_1 \cup \overline{H}_1, N_F)\}$;
4: $\Delta = \emptyset$;
5: $N_H^* = \lceil \frac{N_H}{K} \rceil$;
6: **for** $i = 2$ to K **do**
7: $\quad H_i = \{\alpha_j \mid N_F + N_H^* \cdot (i-1) + 1 \leq j \leq N_F + N_H^* \cdot i\}$;
8: $\quad \overline{H}_i = \{\sim\alpha_j \mid N_F + N_H^* \cdot (i-1) + 1 \leq j \leq N_F + N_H^* \cdot (i-1) + \lceil N_H^* \cdot \lambda \rceil\}$;
9: $\quad$ **for** $\alpha \in H_i \cup \overline{H}_i$ **do**
10: $\quad\quad N_R^* = choose(\{1, \ldots, N_R\})$;
11: $\quad\quad N_S^* = choose(\{1, \ldots, N_S\})$ with $N_S^* \leq N_R^*$;
12: $\quad\quad \Pi_\alpha = \Delta_\alpha = \emptyset$;
13: $\quad\quad$ **for** $j = 1..N_R^*$ **do**
14: $\quad\quad\quad N_M^* = choose(\{1, \ldots, N_M\}))$ s.t. $N_R^* \leq C_{\sum_{l=1}^{i-1} |H_l|, N_M^*}$;
15: $\quad\quad\quad B = subset(\cup_{l=0}^{i}(H_l \cup \overline{H}_l), N_M^*)$;
16: $\quad\quad\quad$ **if** $\nexists r \in \Pi_\alpha \cup \Delta_\alpha$ s.t $body(r) = B$ **then**
17: $\quad\quad\quad\quad$ **if** $j \leq N_S^*$ **then**
18: $\quad\quad\quad\quad\quad$ **if** $\Pi \cup \Pi_\alpha \cup \{\alpha \leftarrow B\}$ is non contradictory **then**
19: $\quad\quad\quad\quad\quad\quad \Pi_\alpha = \Pi_\alpha \cup \{\alpha \leftarrow B\}$;
20: $\quad\quad\quad\quad$ **else**
21: $\quad\quad\quad\quad\quad \Delta_\alpha = \Delta_j \cup \{\alpha \prec B\}$;
22: $\quad\quad \Pi = \Pi \cup \Pi_\alpha$;
23: $\quad\quad \Delta = \Delta \cup \Delta_\alpha$;
24: **return** (Π, Δ).

and that such a set is not contradictory. A further check to avoid generating the same rule more than once is performed at Line 16. Then, firstly, N_S^* rules $\alpha \leftarrow B$, that are not contradictory with the previously generated ones, are added to the strict set Π_α (Line 19), while, subsequently, $N_R^* - N_S^*$ defeasible rules are added to Δ_α (Line 21). Before exiting the cycle started at Line 9, the rules collected in Π_α and Δ_α are added to Π and Δ, respectively (Lines 22 and 23). The algorithm ends by returning the built program (Π, Δ).

5.2.2. Efficiency

For each type of program in Table 2, we considered 10 different programs generated by running Algorithm 2. As done in Section 5.1.1, for each specific program, we generated 10 sets of updates of cardinality $n \in \{1, 4, 8, 16\}$ that are uniformly distributed among the different types of updates.

The results of the experiments on Synare reported in Figs. 6, 7, and 8, where each box refers to a type of program (denoted by the parameters reported in Table 2) and a number n of updates. Specifically, Fig. 6 (resp., 7, 8) reports the results for the series Syn-K (resp., Syn-N_H, Syn-N_R) where parameter K (resp., N_H, N_R) varies, while all other parameters are kept fixed. In each diagram, boxes corresponding to sets of updates of different cardinality are divided by vertical lines, which divide the diagram in five lanes. That is, starting from the left-hand-side, the first (resp., second, third, fourth) lane contains boxes of running times for the incremental computation with $n = 1$ (resp., $n = 4$, $n = 8$, $n = 16$) updates. The running times for the total recomputation from scratch is also shown on the rightmost lane.

The experiments on Syn-K, Syn-N_H, and Syn-N_R confirm the behavior observed in our analysis on Real. The incremental computation approach outperforms the full recomputation from scratch in all the settings considered. On average, the incremental approach takes only 16.1 seconds, while full recomputation takes more than 2.5 minutes (153.8 seconds); therefore, considering every setting, the incremental approach is about 9.5 times faster than full recomputation. Again, the incremental approach's running times increase slightly when the number of updates performed simultaneously increases. Specifically, the average running time for 4 (resp., 8, 16) updates is only 3.2 (resp., 4.9, 6.3) times that for computing 1 update. Two-sample t-tests indicate that the difference between the observed running times for different lanes in Figs. 6, 7, and 8 (*i.e.,* for sets of updates of different cardinality) is statistically significant, with a p-value $p \leq 0.01$. Also, for Syn, the improvement slightly decreases as the number of updates performed simultaneously becomes larger.

The fact that the average improvement obtained for Synis lower than that obtained for Real, and in particular for Real-3, which is comparable to Synon the number of literals and rules, suggests that the structure of the programs in Synis to some extent more complex than that of programs in Real(where the parameters considered in Table 2 do not vary considerably within Real-1, Real-3, and Real-5).

As for the variation of parameters K, N_H, and N_R in Syn, it turns out that the running time of the incremental approach is

- only slightly positively correlated to the number of levels K; the difference between the running times of pairs of boxes belonging to the same lane in Fig. 6 is only marginally significant ($p = 0.12$ on average);

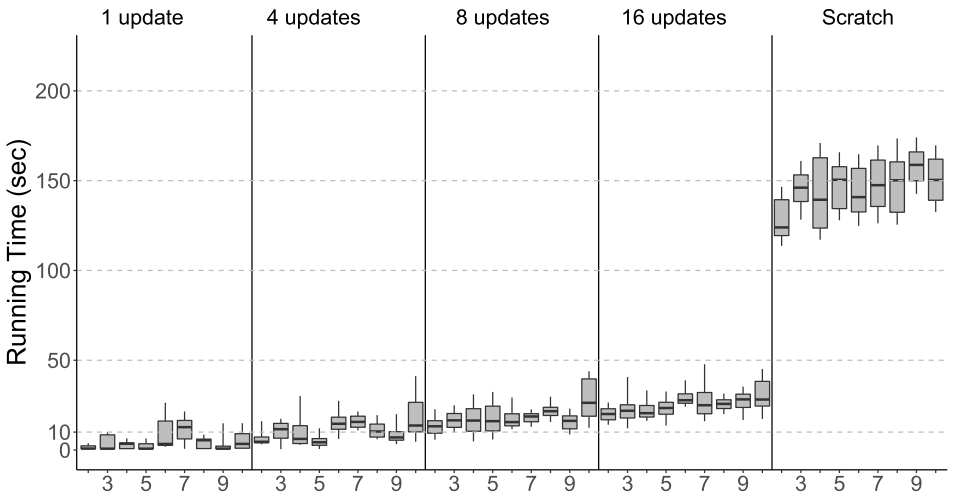


Fig. 6. Running times (in seconds) for the incremental computation of the status of all literals of the DeLP programs in SYN-K (with $K \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$) for 1, 4, 8, 16 updates, and for the total recomputation from scratch.

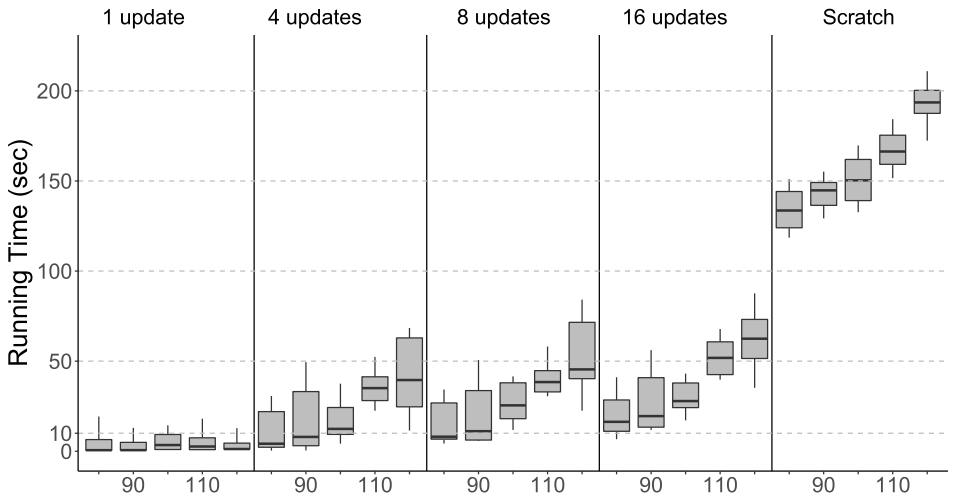


Fig. 7. Running times (in seconds) for the incremental computation of the status of all literals of the DeLP programs in SYN-N_H (with $N_H \in \{80, 90, 100, 110, 120\}$) for 1, 4, 8, 16 updates, and for the total recomputation from scratch.

- positively correlated to the number N_H of distinct rule heads; the difference between the running times of pairs of boxes belonging to the same lane in Fig. 7 is significant ($p \leq 0.01$), except for the lane of 1 update (where p is 0.30 on average) as well as the pair of boxes corresponding to $N_H = 90$ and $N_H = 100$ for 4 and 16 updates, where p is 0.19;
- quite insensitive to the number N_R of rules defining a literal; the difference between the running times of pairs of boxes belonging to the same lane in Fig. 8 is not significant ($p = 0.38$ on average).

It can be observed that this is almost the same behavior as for the total recomputation; therefore, these effects can be ascribed to the DeLP-solver itself. Moreover, these effects are suppressed for the case of single updates where the incremental approach is quite insensitive to variations of K, N_H, and N_R—the improvement obtained by reducing the size of the program to be analyzed balances the effects of increasing K, N_H, and N_R.

To allow a more direct comparison between the results obtained for SYN and those obtained for REAL, we introduce Fig. 9 where diagrams are presented in a way similar to that of Fig. 3. Specifically, the diagram on the left-hand side (resp., middle, right-hand side) of Fig. 9 reports the results for series SYN-K (resp., SYN-N_H, SYN-N_R) of Fig. 6 (resp., 7, 8), but by grouping in single boxes the running times belonging to the same lanes (i.e., considering together running times obtained for the same number $n \in \{1, 4, 8, 16\}$ of updates). Fig. 9 for SYN essentially shows the overall trend observed in Fig. 3 for REAL, though with lower improvements as discussed above.

Finally, Fig. 10 shows improvement versus relatedness (i.e., percentages of related literals rounded to the nearest integer) for SYN. The figure confirms the behavior observed for REAL; that is, the improvement negatively depends on the percent-

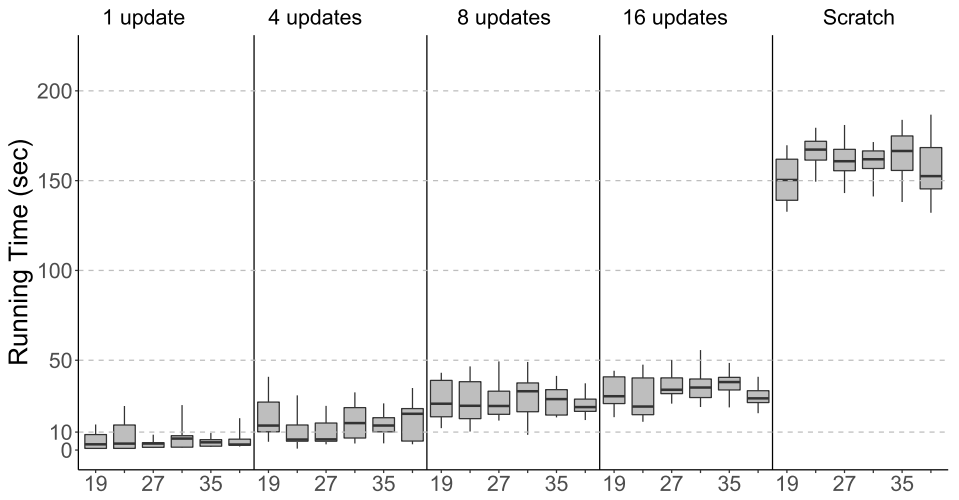


Fig. 8. Running times (in seconds) for the incremental computation of the status of all literals of the DeLP programs in SYN-N_R (with $N_R \in \{19, 23, 27, 31, 35, 39\}$) for 1, 4, 8, 16 updates, and for the total recomputation from scratch.

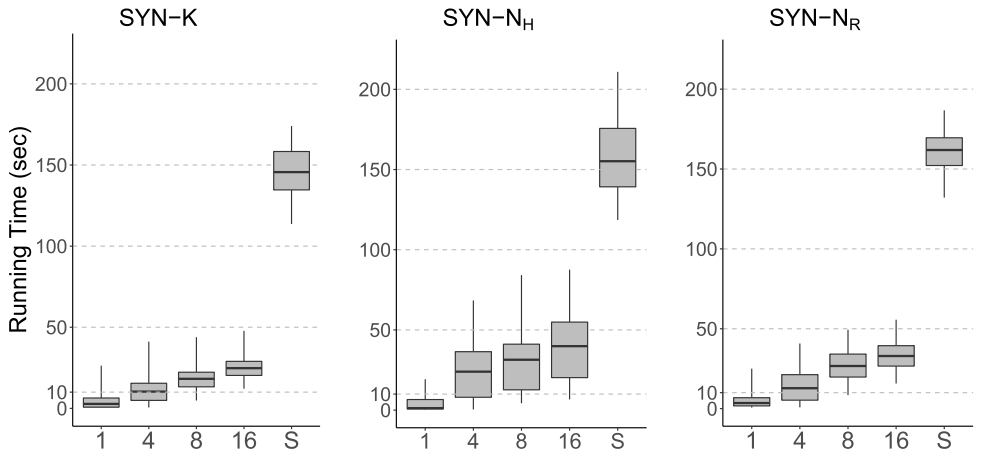


Fig. 9. Running times (in seconds) for the incremental computation of the status of all literals of the DeLP programs in SYN-K, SYN-N_H, SYN-N_R (left-hand side, middle, right-hand side, respectively) for 1, 4, 8, or 16 updates and for the total recomputation from scratch (S).

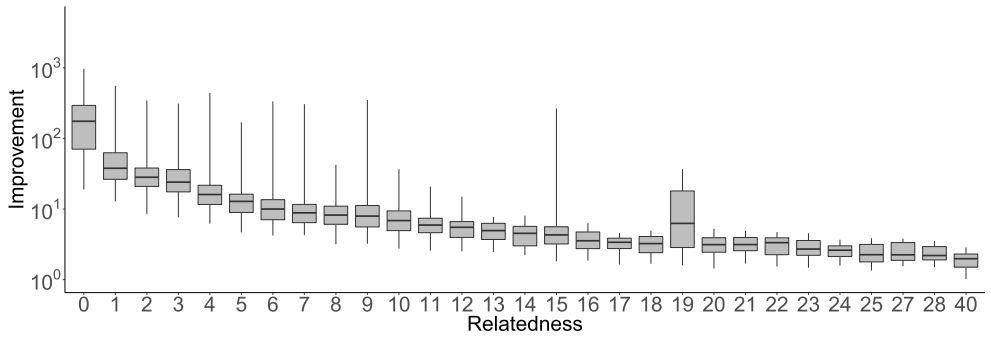


Fig. 10. Improvement (log scale) versus relatedness for the DeLP programs in SYN(percentages of related literals are rounded to the nearest integers).

age of related literals. It is worth noting that the correspondence observed between the experimental results for REALand SYNintuitively supports the validity of the strategy used by Algorithm 2 for the benchmark generation.

5.2.3. Effectiveness

We will now discuss the effectiveness of our approach for SYN; $E(U, \mathcal{P})$ is measured as done for REAL(see Section 5.1.2).

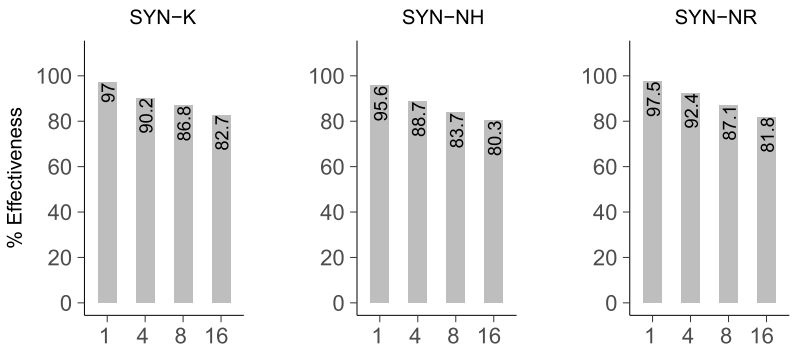


Fig. 11. Average effectiveness for the incremental computation of the status of all literals of the DeLP programs in SYN-K, SYN-N_H, SYN-N_R (left-hand side, middle, right-hand side, respectively) for 1, 4, 8, or 16 updates.

Fig. 11 shows the average effectiveness for SYN-K, SYN-N_H, and SYN-N_R. The effectiveness varies from 97.5% for 1 update performed on SYN-N_R (*i.e.,* only 2.5% of the literals are considered for recomputing the warrant status of the whole set of literals of the program) to 80.3% for 16 updates performed on SYN-N_H. The average effectiveness remains high (more than 95.6%) for 1 update, and in general, it slightly decreases when the number of updates performed simultaneously increases.

The experiments on SYNessentially confirm the behavior observed for REAL; that is, the incremental approach often computes only the status of the literals whose status changes after an update, and it is in all cases faster than total recomputation.

6. Related work

Overviews of key concepts in argumentation theory and formal models of argumentation can be found in [28–31]. Further discussion regarding the uses of computational argumentation as an Agreement Technology can be found in Modgil et al. [32].

Several significant efforts dealing with dynamic aspects of abstract argumentation frameworks have been developed (*e.g.,* [33–35]), where arguments are abstract entities with no internal structure [10]. The techniques proposed in [23,19] and [20,36,21,37–42] are related to our approach in the sense that both exploit the general concept of reachability in the graph corresponding directly to the given abstract argumentation framework (*i.e.,* each node represents an abstract argument). None of the above-mentioned works deal with structured arguments.

Although our technique is related to incremental approaches for abstract argumentation, the complexity of the setting considered here where arguments are "sub-programs" makes those approaches unusable for structured argumentation. Indeed, there are several differences to be considered when moving from abstract to structured argumentation. For instance, viewing an abstract argumentation framework as a logic program [16], the addition of an attack consists in modifying the body of a rule, whereas in our structured argumentation framework we dealt with the addition and deletion of (whole) rules. Moreover, in our setting adding/deleting a single rule could generate/drop multiple defeats between structured arguments.

In the area of structured argumentation, there have been fewer attempts to consider the dynamics of defeasible argumentation; see [3] for a collection of different approaches to the topic. As in the abstract argumentation case, there have been some works following the belief revision approach. In [43], the issue of modifying strict rules to become defeasible was analyzed in the context of revisions effected over a knowledge base, while in [12] the authors thoroughly explored the different cases that may occur when a DeLP program is modified by adding, deleting, or changing its elements. Neither of these works explored the implementation issues related to the problems studied here. Regarding implementations of approaches focusing on improving the tractability of determining the status of pieces of knowledge, in [44,45] the authors consider several alternatives to avoid recomputing warrants. In [46], the authors focus on challenges arising in the development of recommender systems, addressing them via the design of novel architectures that improve the computation of answers. Finally, [47] makes use of heuristics designed to improve efficiency. None of these approaches have a deep connection with our work.

A preliminary study of the problem of incrementally computing the warranted status of arguments in DeLP programs was carried out in [48], where updates consisting only of rule additions were considered. Although the technique proposed in [48] also relies on computing the status of a subset of literals, there are several differences with this paper and with [1], which we extend here: (i) we deal with general updates, consisting of both addition and deletion of strict or defeasible rules even considered simultaneously; (ii) we investigated the complexity of problems related to the incremental computation in dynamic DeLP programs; and (iii) our approach builds on the novel concepts of *influenced set* and *core literals* for an update w.r.t. a DeLP program, from which the sets of *preserved* and *inferable* literals are defined; (iv) we provided a thorough experimental analysis considering several parameters and a novel benchmark generator, as well as multiple updates. Finally, apart from being more general, our technique turned out to be more efficient and effective w.r.t. that proposed in [48]: we

experimentally found that our algorithm is over 30% faster, and exhibits twice the effectiveness (for updates consisting of a single rule addition, the only case considered there).

We believe that the ideas proposed in this work may motivate the research of similar techniques for the incremental computation of other structured argumentation frameworks such as, for example, ABA [5,6] and ASPIC$^+$ [4]. Regarding ABA, the construction of deductions is very similar to that of arguments for DeLP, although the way arguments attack each other is different. Therefore, similarly, the ABA framework could be represented using hypergraphs (where assumptions may be modeled as defeasible facts) to identify irrelevant updates and restrict the hypergraph to compute updated programs' semantics efficiently. The similarities between DeLP and ASPIC$^+$ programs are even more substantial: both have a distinction between strict and defeasible rules and both use preferences to resolve attacks into defeats; but, while ASPIC$^+$ evaluates arguments using one of the available AF semantics, DeLP has a special-purpose definition of argument evaluation [49]. Therefore, the ideas developed here can be of inspiration to devise optimizing techniques for the recomputation of the semantics of ASPIC$^+$ programs subject to updates.

Another field of great interest related to the updating of DeLP programs is belief revision. The belief revision problem, consisting of the reconciliation of beliefs after an epistemic input to keep a theory consistent, is closely connected with the problem of recomputing the semantics of a DeLP program. In this case, the problem studied in this paper may be useful to inspire similar techniques in belief revision. Essentially, beliefs can be viewed as conclusions that can be supported by explanations [43], explanations that in the case of a DeLP program are warranting arguments for those conclusions; thus, adding or removing rules from a DeLP program can be described as a belief revision problem over the set of conclusions (always a consistent set of warranted literals). In this paper, we have explored a construction technique that can be used to cope with an updating process of a DeLP program. To close the loop, what remains to be done is to attempt to design a set of postulates that will characterize the changes, a task outside of this work's aim and scope, but that we will pursue as part of our research project. For instance, one of such postulates should require that updating a DeLP program must produce a DeLP program; in our construction, this is guaranteed by requiring that the updates be feasible. Another possible requirement is that of minimal change in the set of conclusions, *i.e.,* only the conclusions affected by the update of the program should be subject to change, and our construction supports this. If obtaining such a set of postulates is possible, our construction will be just one of the possible updating processes making the changes effective, and a representation theorem will have to be proved. Note that unlike the case of logic programming [50–53], a defeasible logic program cannot have an inconsistent strict part (strict rules and facts), and any potential inconsistency that is introduced via defeasible rules will be resolved by the argumentative inference engine that will only produce warranted literals. The process of updating can be guided by a meta-method inspired by the work presented in [12], where a prioritized argument revision operator for DeLP programs is introduced so that the newly inserted argument (a set of rules) ends up undefeated after the revision, thus warranting its conclusion.

7. Conclusions and future work

We have addressed the problem of avoiding wasted effort when determining the warrant status of literals in a DeLP program after it is changed by applying a set of strict/defeasible rule additions and deletions. We identified certain conditions under which single or multiple updates can be guaranteed to have no effect whatsoever, and then proposed a data structure that helps us determine the potential effects of the update, thus allowing us to conclude that the rest of the literals will be unaffected. The resulting incremental computation algorithm is shown—via a thorough set of experiments—to yield significantly lower running times in practice.

Regarding the generality of the approach, the analysis in a concrete formalism like DeLP is richer conceptually than in an abstract setting because the internal structure of arguments is readily available, and thus one can leverage this instead of reasoning about the relationships between abstract arguments only. However, as discussed earlier, we believe the basic ideas in the framework could carry over to other frameworks, *e.g.,* ASPIC$^+$ or ABA. Although our algorithm could be used to improve the performance of the inference process of DeLP, it is orthogonal to the process itself and the engine that implements it. In fact, the ideas behind the concepts of relevant/irrelevant updates, as well as preserved/core/inferable literals, could eventually be extended to work with other formalisms/inference processes. This is a direction we are planning to take in future work.

Finally, given the recent interest in probabilistic extensions of structured argumentation [54], we envisage that our approach could be also used in a probabilistic context where a probability distribution is associated with possible worlds, each of them representing a DeLP program (consisting of the rules and facts that hold in that world). Actually, our incremental technique is apt to be exploited in computing the status of literals in a world (i.e., a DeLP program) after performing an update—as there are in general many different worlds, with different initial statuses, the same update may yield different updated statuses. Therefore, in order to be most effective, the techniques developed in this paper need to be extended to identify which worlds are affected by updates, as well as extending the concept of update to probabilistic models and annotation functions. Computing the status of literals in a world is also useful in Monte Carlo approaches or in general approximation schemes that provide approximate answers by looking only at subsets of all possible worlds.

CRediT authorship contribution statement

All authors have participated in (a) conception and design, or analysis and interpretation of the data; (b) drafting the article or revising it critically for important intellectual content; and (c) approval of the final version.

This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors are grateful to the participants of the workshop *Current Trends in Formal Argumentation* (held at the University Centre of Bertinoro from November 3rd to 6th, 2019) for discussions on the topic of this paper. Special thanks to Matthias Thimm for pointing out limitations of the technique previously presented in [1] that led to a more complete theoretical analysis and characterization of the irrelevant updates, and of the set of influenced literals, which are at the basis of our incremental approach. The authors wish to thank the referees for providing detailed comments and numerous suggestions that helped in substantially improving this paper. Finally, this work was partly supported in Argentina by Universidad Nacional del Sur (UNS) under grant PGI 24/ZN34, Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), and Agencia Nacional de Promoción Científica y Tecnológica under grant PICT-2018-0475.

Appendix A. Proofs

In this appendix we provide the proofs for results stated in the core of the paper. To aid readability, we restate the results.

Proposition 1. *Given a DeLP program $\mathcal{P}$ and an argument $\langle \mathcal{A}, \alpha \rangle$ of $\mathcal{P}$, then α occurs as a node of $G(\mathcal{P})$.*

Proof. The result follows from the fact that if a literal $\alpha \in Lit_{\mathcal{P}}$ does not belong to $G(\mathcal{P})$ then, by definition of $G(\mathcal{P})$ (basic step and iterative steps), there is no derivation for α w.r.t. $\mathcal{P}$, and thus there is no argument for α w.r.t. $\mathcal{P}$ since item (i) of Definition 1 does not hold.

Moreover, it is important to observe that the converse does not hold in the general case. For instance, consider the DeLP program $\langle \{f\}, \{(\sim a \prec f), (p \prec \sim a), (a \prec p)\} \rangle$. Then, $G(\mathcal{P})$ contains all the literals in $Lit_{\mathcal{P}}$, while there is no argument for a as the sequence of literals for a (*i.e.,* $(f, \sim a, p, a)$) is not a derivation since it is contradictory. □

Theorem 1. *ArgumentExistence$[\mathcal{P}, \alpha]$ is NP-complete.*

Proof. Membership in NP follows from the results in [55], where it was shown that checking whether a given subset of defeasible rules is an argument is P-complete. This result guarantees that a polynomial-time guess and check strategy exists for the problem of deciding the existence of an argument. Thus, the problem we are considering is in NP.

To prove hardness, we provide a LOGSPACE reduction to our problem from the 3-SATISFIABILITY (3SAT) problem [56], whose definition is recalled next. An instance of 3SAT is a pair $\langle V, \Phi \rangle$, where $V = \{x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_k, \overline{x}_k\}$ is a set of propositional variables (where each $\overline{x}_i$ is the negation of x_i) and Φ is a propositional CNF formula of the form $\hat{C}_1 \wedge \ldots \wedge \hat{C}_n$ defined over V. Specifically, each $\hat{C}_i$ (with $1 \leq i \leq n$) is a clause of the form $x_{i,1} \vee x_{i,2} \vee x_{i,3}$, where $x_{i,j}$ $(1 \leq j \leq 3)$ is a (positive or negated) variable in V. A *truth assignment* for the variables in V is a function $\tau : V \to \{true, false\}$ such that, for each pair of variables x, $\overline{x}$ in V, it holds that $\tau(x) = \neg\tau(\overline{x})$. The truth value of the proposition resulting by substituting every variable x in Φ with $\tau(x)$ is denoted as $\tau(\Phi)$. The answer for a 3SAT instance $\langle V, \Phi \rangle$ is true iff there is a truth assignment τ for V such that $\tau(\Phi)$ is *true*.

Given an instance $\langle V, \Phi \rangle$ of 3SAT, we obtain an instance $\langle \mathcal{P}, \alpha \rangle$ of our problem as follows. Let $Lit_{\mathcal{P}} = \{\phi, C_1, \ldots C_n\} \cup \{x_i, \sim x_i, f_i \mid x_i, \overline{x}_i \in V\}$. The DeLP program $\mathcal{P} = (\Pi, \Delta)$ is such that:

- Π consists of a fact f_i for each pair of variables $x_i, \overline{x}_i \in V$;
- Δ consists of the following defeasible rules:
  - For each pair of variables $x_i, \overline{x}_i \in V$, the defeasible rules $x_i \prec f_i$ and $\sim x_i \prec f_i$;
  - For each clause $\hat{C}_i$ (with $1 \leq i \leq n$) and variable in it, the defeasible rule:
    $C_i \prec x_{i,j}$ if $x_{i,j}$ occurs in $\hat{C}_i$ (with $1 \leq j \leq 3$),
    $C_i \prec \sim x_{i,j}$ if $\overline{x}_{i,j}$ occurs in $\hat{C}_i$;
  - The defeasible rule $\phi \prec C_1, \ldots, C_n$.

Finally, let the literal α in the instance $\langle \mathcal{P}, \alpha \rangle$ of our problem be $\phi \in Lit_{\mathcal{P}}$.

We now show that $\langle V, \Phi \rangle$ is true if and only if there is an argument $\langle \mathcal{A}, \phi \rangle$ for ϕ w.r.t. $\mathcal{P}$.

($\Rightarrow$) Let τ be a truth assignment for V such that $\tau(\Phi)$ is true. Let $\mathcal{B} \subseteq \Delta$ consist of the following defeasible rules:

- $\phi \prec C_1, \ldots, C_n$;
- $C_i \prec x_{i,j}$ (resp., $C_i \prec \sim x_{i,j}$) (with $1 \leq i \leq n$ and $1 \leq j \leq 3$) if $\tau(x_{i,j})$ is true and $x_{i,j}$ is in $\hat{C}_i$ (resp., $\tau(\overline{x}_{i,j})$ is true and $\overline{x}_{i,j}$ is in $\hat{C}_i$);
- $x_i \prec f_i$ (resp., $\sim x_i \prec f_i$) if $\tau(x_i)$ (resp. $\tau(\overline{x}_i)$) is true.

Therefore, $\mathcal{B}$ contains (at least) a rule of the form $C \prec x$ for each clause $\hat{C}$ in Φ and (positive or negated) variable x causing C to evaluate to true. Moreover, $\mathcal{B}$ contains a rule for deriving either x or $\sim x$, depending on which corresponding variable (x or $\overline{x}$) is assigned to true by τ.

Since $\tau(\Phi)$ is true, $\mathcal{B}$ is a derivation for ϕ from $\Pi \cup \mathcal{B}$ and such that $\Pi \cup \mathcal{B}$ is not contradictory (as either $x_i \prec f_i$ or $\sim x_i \prec f_i$ is in $\mathcal{B}$, and no other pair of complementary literals can be derived using the rules in $\mathcal{B}$). Moreover, the existence of $\mathcal{B}$ implies that there exists $\mathcal{A} \subseteq \mathcal{B}$ such that i) $\mathcal{A}$ is a derivation for ϕ from $\Pi \cup \mathcal{A}$, ii) $\Pi \cup \mathcal{A}$ is not contradictory, and iii) $\mathcal{A}$ is minimal. Thus, $\langle \mathcal{A}, \phi \rangle$ is an argument for ϕ w.r.t. $\mathcal{P}$.

($\Leftarrow$) Given an argument $\langle \mathcal{A}, \phi \rangle$ for ϕ w.r.t. $\mathcal{P}$, we define a truth assignment τ for the variables in V such that $\tau(\Phi)$ is true as follows.

1) For each defeasible rule in $\mathcal{A}$ of the form $x_i \prec f_i$, we define $\tau(x_i) = \mathit{true}$ and $\tau(\overline{x}_i) = \mathit{false}$.
2) For each defeasible rule in $\mathcal{A}$ of the form $\sim x_i \prec f_i$, we define $\tau(x_i) = \mathit{false}$ and $\tau(\overline{x}_i) = \mathit{true}$.
3) Let $V' \subseteq V$ be the subset of the variables that have not been assigned a truth value at steps 1) and 2) above. We can assign to each variable $x \in V'$ either *true* or *false*, provided that $\tau(x) = \neg\tau(\overline{x})$. Let us assume that $\tau(x) = \mathit{true}$ and $\tau(\overline{x}) = \mathit{false}$ for each $x, \overline{x} \in V'$.

It is easy to see that τ is a truth assignment, *i.e.,* a function such that for each pair of variables x, $\overline{x}$ in V, $\tau(x) = \neg\tau(\overline{x})$; this follows from the fact that $\Pi \cup \mathcal{A}$ is not contradictory, and item 3) of the above-described construction. Moreover, since $\mathcal{A}$ is a derivation for ϕ from $\Pi \cup \mathcal{A}$, it means that every clause of Φ is satisfied by τ, and thus that $\tau(\Phi)$ is true. □

Corollary 1. *ArgumentExistence*$[\mathcal{P}, \alpha]$ *is NP-complete, even if* $|body(r)| \leq 2$, *for all* $r \in (\Pi \cup \Delta)$ *(where* $\mathcal{P} = \langle \Pi, \Delta \rangle$*).*

Proof. Membership follows from Theorem 1. Hardness can be proved by slightly modifying the construction in the proof of Theorem 1 so that all rules have at most two literals in the body. Specifically, we only need to rewrite the defeasible rule $\phi \prec C_1, \ldots, C_n$, as the others have at most one literal in the body. It is easy to see that $\phi \prec C_1, \ldots, C_n$ can be rewritten into an equivalent set of rules of the following form (w.l.o.g., assume $n = 2^k$):

$\psi_{2^1,1} \prec C_1, C_2$
$\psi_{2^1,2} \prec C_3, C_4$
...
$\psi_{2^1,\frac{n}{2}} \prec C_{n-1}, C_n$
$\psi_{2^2,1} \prec \psi_{2^1,1}, \psi_{2^1,2}$
...
$\psi_{2^2,\frac{n}{4}} \prec \psi_{2^1,\frac{n}{2}-1}, \psi_{2^1,\frac{n}{2}}$
...
$\psi_{2^{\log_2 n},1} \prec \psi_{2^{(\log_2 n)-2},1}, \psi_{2^{(\log_2 n)-2},2}$
$\phi \prec \psi_{2^{\log_2 n},1}$.

Roughly speaking, we build a (binary) tree whose leaf nodes represent the clauses $C_1, \ldots, C_n$, and the internal nodes $\psi_{i,j}$ at level $i \in [2^1, 2^{\log_2 n}]$ represent of conjunctions of their children. So, the root node $\psi_{2^{\log_2 n},1}$ logically represents the conjunction of all clauses.

As there are n groups of rules, each of them consisting of at most $n/2$ rules, the number of rules is polynomial w.r.t. the number n of clauses, and thus it is polynomial w.r.t. the size of the input 3SAT instance. This way, we obtain a DeLP program equivalent to that of the proof of Theorem 1 and such that each rule has at most two literals in the body. The statement follows. □

Proposition 2. *ArgumentExistence*$[\mathcal{P}, \alpha]$ *is in PTIME if either* (i) α *does not depend in* $G(\mathcal{P})$ *on literals* β *and* γ *such that* $\{\beta, \gamma\} \cup \Pi$ *is contradictory, or* (ii) α *is not in* $G(\mathcal{P})$.

Proof. First observe that $G(\mathcal{P})$ can be built in polynomial time w.r.t. the size of $\mathcal{P}$, and checking whether α depends on two contradictory literals is polynomial, too. Then, if α does not depend on two contradictory literals in $G(\mathcal{P})$, a derivation for α can be obtained by using the rules belonging to an inverse path from α to facts in $\mathcal{P}$: once a node t has been visited, if there is a hyper-edge (S, t), then S can become visited as well. Checking for the existence of such path in $G(\mathcal{P})$ can be accomplished in polynomial time.

Case (*ii*) follows from Proposition 1, according to which if a literal does not belong to the hypergraph $G(\mathcal{P})$ then there is no argument for it. □

Theorem 2. *LiteralStatus*$[\mathcal{P}, \alpha, \sigma]$ *is coNP-hard for* $\sigma \in \{\text{IN}, \text{OUT}\}$ *and NP-hard for* $\sigma = \text{UNDECIDED}$*, even if* $|body(r)| \leq 2$ *for all* $r \in (\Pi \cup \Delta)$ *(where* $\mathcal{P} = \langle \Pi, \Delta \rangle$*).*

Proof. We first show that deciding whether $S_{\mathcal{P}}(\alpha) = \text{IN}$ is coNP-hard.

We provide a reduction from the complement of 3SAT by building on the construction given in the proof of Theorem 1. In the following, we use the notation introduced in the proof of Theorem 1. Given an instance $\langle V, \Phi \rangle$ of 3SAT, we define an instance $\langle \mathcal{P}', \alpha, \text{IN} \rangle$ of our problem as follows. Let $\mathcal{P} = (\Pi, \Delta)$ be the DeLP program defined in the proof of Theorem 1. Then, $\mathcal{P}' = (\Pi', \Delta')$ where:

- Π' consists of the facts in Π plus new fact g; and
- Δ' consists of the defeasible rules in Δ plus defeasible rule $\sim\phi -\!\prec g$.

The arguments that can be built from $\mathcal{P}'$ are as follows.

- In the proof of Theorem 1 we have shown that there is (at least) an argument of the form $\langle \mathcal{A}, \phi \rangle$ for ϕ w.r.t. $\mathcal{P}$ iff Φ is satisfiable. This continues to hold for $\mathcal{P}'$, as the addition of fact g and defeasible rule $\sim\phi -\!\prec g$ to $\mathcal{P}'$ does not invalidate that argument for ϕ. Therefore, there is (at least) an argument of the form $\langle \mathcal{A}, \phi \rangle$ for ϕ w.r.t. $\mathcal{P}'$ iff Φ is satisfiable.
- We also have arguments of the following forms:
  - arguments for C_i (with $1 \leq i \leq n$) $\langle \mathcal{C}_i, C_i \rangle = \langle \{C_i -\!\prec x_j,\ x_j -\!\prec f_i\}, C_i \rangle$ or $\langle \mathcal{C}'_i, C_i \rangle = \langle \{C_i -\!\prec \sim x_j,\ \sim x_j -\!\prec f_i\}, C_i \rangle$;
  - arguments for x_i (with $1 \leq i \leq k$) $\langle \mathcal{X}_i, x_i \rangle = \langle \{x_i -\!\prec f_i\}, x_i \rangle$;
  - arguments for $\sim x_i$ (with $1 \leq i \leq k$) $\langle \mathcal{X}_i, \sim x_i \rangle = \langle \{\sim x_i -\!\prec f_i\},\ \sim x_i \rangle$.

  It is worth noting that some of these arguments are the sub-arguments of $\langle \mathcal{A}, \phi \rangle$, and they exist even if $\langle \mathcal{A}, \phi \rangle$ does not exist, *i.e.,* Φ is not satisfiable.
- Finally, we have an argument $\langle \mathcal{A}', \sim\phi \rangle = \langle \{\sim\phi -\!\prec g\}, \sim\phi \rangle$ for $\sim\phi$.

As for the preference relation $\succ$ between arguments, we have that $\succ$ is empty, *i.e.,* no argument for a literal γ listed above is preferred to an argument for $\sim\gamma$; observe that the relation yielded by generalized specificity in this case is indeed empty, and the result holds for any criterion that leads to an empty $\succ$.

Let the literal α in the instance $\langle \mathcal{P}', \alpha, \text{IN} \rangle$ of our problem be $\sim\phi$, that is, we ask if the status of $\sim\phi$ is IN.

We now show that $\langle V, \Phi \rangle$ is *false* iff $S_{\mathcal{P}}(\sim\phi) = \text{IN}$.

($\Rightarrow$) We show that if Φ is not satisfiable then there exists a dialectical tree $\mathcal{T}_{\langle \mathcal{A}', \sim\phi \rangle}$ w.r.t. $\mathcal{P}'$ that is marked UNDEFEATED. In particular, $\mathcal{T}_{\langle \mathcal{A}', \sim\phi \rangle}$ consists of argumentation line $[\langle \mathcal{A}', \sim\phi \rangle]$. In fact, the rooted argument $\langle \mathcal{A}', \sim\phi \rangle$ is not defeated by any other argument as only arguments of the form $\langle \mathcal{A}, \phi \rangle$ could be defeaters of $\langle \mathcal{A}', \sim\phi \rangle$; but they do not exist since Φ is not satisfiable (as shown in the proof of Theorem 1). Therefore $\sim\phi$ is warranted and $S_{\mathcal{P}}(\sim\phi) = \text{IN}$.

($\Leftarrow$) If $S_{\mathcal{P}}(\sim\phi) = \text{IN}$ then there is a dialectical tree whose root is an argument for $\sim\phi$ and is marked UNDEFEATED, as shown in what follows. The argument in the root must be $\langle \mathcal{A}', \sim\phi \rangle$, as there is no other argument for $\sim\phi$. Moreover, as only an argument of the form $\langle \mathcal{A}, \phi \rangle$ can be a defeater of $\langle \mathcal{A}', \sim\phi \rangle$, in order for the root node of dialectical tree $\mathcal{T}_{\langle \mathcal{A}', \sim\phi \rangle}$ to be marked UNDEFEATED: either (*i*) $\mathcal{T}_{\langle \mathcal{A}', \sim\phi \rangle}$ consists of argumentation line $[\langle \mathcal{A}', \sim\phi \rangle]$; or (*ii*) $\mathcal{T}_{\langle \mathcal{A}', \sim\phi \rangle}$ consists of an argumentation line of the form $[\langle \mathcal{A}', \sim\phi \rangle, \langle \mathcal{A}, \phi \rangle, \langle \mathcal{Y}, y \rangle, \ldots, \langle \mathcal{Z}, z \rangle]$ where $\langle \mathcal{Y}, y \rangle$ can be marked UNDEFEATED.

In case (*i*), as $\langle \mathcal{A}', \sim\phi \rangle$ is not preferred to $\langle \mathcal{A}, \phi \rangle$, to have argumentation line $[\langle \mathcal{A}', \sim\phi \rangle]$, it must be the case that $\langle \mathcal{A}, \phi \rangle$ does not occur as a defeater because it does not exist, meaning that Φ is not satisfiable (cf. proof of Theorem 1).

We show that case (*ii*) cannot occur. Consider a sequence of arguments of the form $[\langle \mathcal{A}', \sim\phi \rangle, \langle \mathcal{A}, \phi \rangle]$ (it can be obtained if Φ is satisfiable). As there is no preference between $\langle \mathcal{A}', \sim\phi \rangle$ and $\langle \mathcal{A}, \phi \rangle$, argument $\langle \mathcal{A}, \phi \rangle$ is a blocking defeater for $\langle \mathcal{A}', \sim\phi \rangle$. Therefore, to extend sequence $[\langle \mathcal{A}', \sim\phi \rangle, \langle \mathcal{A}, \phi \rangle]$ we need a *proper* defeater for $\langle \mathcal{A}, \phi \rangle$. However, among the arguments that could be defeaters for $\langle \mathcal{A}, \phi \rangle$ (*i.e.,* arguments of the forms $\langle \mathcal{X}_i, x_i \rangle$ or $\langle \mathcal{X}_i, \sim x_i \rangle$), there is no argument that is preferred to a sub-argument of $\langle \mathcal{A}, \phi \rangle$, because there are no preferences between arguments. Thus, an argumentation line of the form $[\langle \mathcal{A}', \sim\phi \rangle, \langle \mathcal{A}, \phi \rangle, \langle \mathcal{Y}, y \rangle, \ldots, \langle \mathcal{Z}, z \rangle]$ cannot exist, meaning that if $S_{\mathcal{P}}(\sim\phi) = \text{IN}$ then $\mathcal{T}_{\langle \mathcal{A}', \sim\phi \rangle}$ consists of its root node only that is marked UNDEFEATED, which suffices to complete the proof for the IN case.

Concerning the problem of deciding whether $S_{\mathcal{P}}(\alpha) = \text{OUT}$, its coNP-hardness follows from the fact that $S_{\mathcal{P}}(\alpha) = \text{OUT}$ iff $S_{\mathcal{P}}(\sim\alpha) = \text{IN}$ for any literal α. In particular, in the construction given above, we have that $\langle V, \Phi \rangle$ is false iff $S_{\mathcal{P}}(\sim\phi) = \text{IN}$ iff $S_{\mathcal{P}}(\phi) = \text{OUT}$.

We now show that deciding whether $S_{\mathcal{P}}(\alpha) = \text{UNDECIDED}$ is NP-hard. In particular, considering the above-described construction, we now show that $\langle V, \Phi \rangle$ is *true* iff $S_{\mathcal{P}}(\phi) = \text{UNDECIDED}$.

($\Rightarrow$) We show that if Φ is satisfiable then neither ϕ nor $\sim\phi$ are warranted, that is, there are no dialectical trees $\mathcal{T}_{\langle \mathcal{A}, \phi \rangle}$ and $\mathcal{T}_{\langle \mathcal{A}', \sim\phi \rangle}$ w.r.t. $\mathcal{P}'$ that can be marked UNDEFEATED.

Consider a dialectical tree whose root is an argument for $\sim\phi$. From the analysis above, the root node is $\langle \mathcal{A}', \sim\phi \rangle$ and the tree $\mathcal{T}_{\langle \mathcal{A}', \sim\phi \rangle}$ consists of argumentation line $[\langle \mathcal{A}', \sim\phi \rangle, \langle \mathcal{A}, \phi \rangle]$. This sequence of arguments cannot be extended since $\langle \mathcal{A}, \phi \rangle$

is a blocking defeater for $\langle \mathcal{A}', \sim\phi \rangle$ and there is no proper defeater for $\langle \mathcal{A}, \phi \rangle$. Moreover, there is no other defeater for the root node, meaning that $\mathcal{T}_{\langle \mathcal{A},\phi \rangle}$ consists of only that argumentation line. Therefore, $\mathcal{T}_{\langle \mathcal{A}',\sim\phi \rangle}$ cannot be marked UNDEFEATED.

Analogously, consider a dialectical tree whose root is an argument for ϕ. Thus, the root node is $\langle \mathcal{A}, \phi \rangle$ and the tree $\mathcal{T}_{\langle \mathcal{A},\phi \rangle}$ consists of argumentation lines of the form $[\langle \mathcal{A}, \phi \rangle, \langle \mathcal{W}, w \rangle]$, where $\langle \mathcal{W}, w \rangle$ is either $\langle \mathcal{A}', \sim\phi \rangle$, or $\langle \mathcal{X}_i, x_i \rangle$ or $\langle \mathcal{X}_i, \sim x_i \rangle$ (with $1 \leq i \leq k$). However, $[\langle \mathcal{A}, \phi \rangle, \langle \mathcal{W}, w \rangle]$ cannot be extended since arguments $\langle \mathcal{A}', \sim\phi \rangle$, or $\langle \mathcal{X}_i, x_i \rangle$ or $\langle \mathcal{X}_i, \sim x_i \rangle$ are blocking defeaters for $\langle \mathcal{A}, \phi \rangle$ and there is no proper defeater for them. Therefore, $\mathcal{T}_{\langle \mathcal{A},\phi \rangle}$ cannot be marked UNDEFEATED.

($\Leftarrow$) Assume that if Φ is not satisfiable, we show that $S_{\mathcal{P}}(\phi) \neq$ UNDECIDED.

In the left-to-right direction of the proof for showing that deciding whether $S_{\mathcal{P}}(\alpha) =$ IN is coNP-hard, we have shown that if Φ is not satisfiable then $S_{\mathcal{P}}(\sim\phi) =$ IN. Since $S_{\mathcal{P}}(\phi) =$ OUT iff $S_{\mathcal{P}}(\sim\phi) =$ IN, we have that $S_{\mathcal{P}}(\phi) \neq$ UNDECIDED. This suffices to complete the proof for the UNDECIDED case.

Finally, reasoning as in the proof of Corollary 2, it can be shown that the statement holds even if the DeLP program contains at most two literals in each rule's body. □

Theorem 3 *(Preserved Literal). Let $\mathcal{P}$ be a DeLP program and U a set of updates for $\mathcal{P}$. Let $\mathcal{P}' = U(\mathcal{P})$ be the updated program, and $G(\mathcal{P}') = \langle N', H' \rangle$ be the updated labeled hypergraph. Then, a literal $\alpha \in N'$ is preserved (i.e., $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$) if either i) $\{\alpha, \sim\alpha\} \cap CL(U, \mathcal{P}) \neq \emptyset$, or ii) $\alpha \notin \mathcal{R}(U, \mathcal{P})$.*

Proof. We separately consider the two cases.

i) If $\alpha \in CL(U, \mathcal{P})$ then no argument for $\sim\alpha$ exists neither before nor after performing the update, entailing that $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha) =$ IN. The proof for the case of $\sim\alpha \in CL(U, \mathcal{P})$ is analogous.

ii) Let $\mathcal{H} = \{head(r) \mid \pm r \in U\}$. Let $\mathcal{R}^*$ be the set of literals computed as the set $\mathcal{R}(U, \mathcal{P})$ of related literals but using $G(\mathcal{P})$ instead of $G(U, \mathcal{P})$ in the forward and backward reachability. Since $G(U, \mathcal{P})$ subsumes $G(\mathcal{P})$, it is the case that if $\alpha \notin \mathcal{R}(U, \mathcal{P})$ then $\alpha \notin \mathcal{R}^*$. The latter entails that no $h \in \mathcal{H}$ can belong to an argument of a dialectical tree for $\langle \mathcal{A}, \alpha \rangle$, for any $\mathcal{A}$—with a little abuse of notation, we write this as $h \notin \mathcal{T}^*_{\langle \mathcal{A},\alpha \rangle}$ for all $\mathcal{A}$. We now separately consider the three possible statuses $S_{\mathcal{P}}(\alpha)$ of α for which it may turn out to be preserved:
(1) If $S_{\mathcal{P}}(\alpha) =$ IN, then there exists a marked dialectical tree for $\mathcal{P}$ $\mathcal{T}^*_{\langle \mathcal{A},\alpha \rangle}$ whose root is marked as UNDEFEATED. Since $h \notin \mathcal{T}^*_{\langle \mathcal{A},\alpha \rangle}$ iff there is no node $\langle \mathcal{B}, \beta \rangle$ in $\mathcal{T}^*_{\langle \mathcal{A},\alpha \rangle}$ such that h belongs to $\langle \mathcal{B}, \beta \rangle$ for $h \in \mathcal{H}$, it follows that the marked dialectical tree continues to exist for the updated DeLP program $\mathcal{P}' = U(\mathcal{P})$. Thus α is preserved.
(2) If $S_{\mathcal{P}}(\alpha) =$ OUT, then $S_{\mathcal{P}}(\sim\alpha) =$ IN, and $\alpha \notin \mathcal{R}(U, \mathcal{P})$ iff $\sim\alpha \notin \mathcal{R}(U, \mathcal{P})$. Then, reasoning as in item (1) it follows that $S_{\mathcal{P}}(\sim\alpha)$ does not change.
(3) If $S_{\mathcal{P}}(\alpha) =$ UNDECIDED, then the root of every $\mathcal{T}^*_{\langle \mathcal{A},\alpha \rangle}$ is marked as DEFEATED for all possible arguments $\mathcal{A}$. Also, we have that $S_{\mathcal{P}}(\sim\alpha) =$ UNDECIDED, and then the root of every $\mathcal{T}^*_{\langle \mathcal{A},\sim\alpha \rangle}$ is marked DEFEATED for all possible arguments $\mathcal{A}$. Reasoning as above, it can be shown that each such tree still exists, and its marking remains the same. Then, $S_{\mathcal{P}}(\alpha)$ and $S_{\mathcal{P}}(\sim\alpha)$ continues to be UNDECIDED. □

Lemma 1. *Let $\mathcal{P}$ be a DeLP program and U be a set of updates. If there exists a sequence $[u_1, \ldots, u_{j-1}, u_j, \ldots, u_k]$ of updates such that (i) $u_1 \in U$ is irrelevant for $\mathcal{P}$ and (ii) every update $u_j \in U$, with $j \geq 2$, is irrelevant for the updated program $u_{j-1}(\ldots(u_1(\mathcal{P})))$, then U is irrelevant for $\mathcal{P}$.*

Proof. Let $\mathcal{P}^0 = \mathcal{P}$ and $\mathcal{P}^j = u_j(u_{j-1}(\ldots(u_1(\mathcal{P}))))$, with $1 \leq j \leq k$, be the DeLP program obtained by iteratively performing the first j updates $u_1, \ldots, u_j$ of the sequence. Clearly, $\mathcal{P}^k = U(\mathcal{P})$, and since $\forall j \in [1..k]$, u_j is irrelevant for the updated program $\mathcal{P}^{j-1}$, it holds that $S_{\mathcal{P}^j}(\alpha) = S_{\mathcal{P}^{j-1}}(\alpha)$, where $S_{\mathcal{P}^0}(\alpha) = S_{\mathcal{P}}(\alpha)$. Thus, it follows that $S_{\mathcal{P}^k}(\alpha) = S_{U(\mathcal{P})}(\alpha) = S_{\mathcal{P}}(\alpha)$ for each $\alpha \in Lit$. □

Lemma 2. *Let $\mathcal{P}$ be a DeLP program. An update $\pm r$ is irrelevant for $\mathcal{P}$ if at least one of the following conditions holds:*

(i) head(r) does not belong to $G(\{\pm r\}, \mathcal{P})$, or
(ii) $\{head(r), \sim head(r)\} \cap CL(\{\pm r\}, \mathcal{P}) \neq \emptyset$, or
(iii) $\exists\, \beta \in body(r)$ such that $\sim\beta \in CL(\{\pm r\}, \mathcal{P})$.

Proof. (Case (*i*)). If *head(r)* does not belong to $G(\{\pm r\}, \mathcal{P})$ then *body(r)* does not belong to $G(\{\pm r\}, \mathcal{P})$ either (otherwise *head(r)* would be in $G(\{\pm r\}, \mathcal{P})$ by Definition 5). In particular, $G(\{\pm r\}(\mathcal{P})) = G(\mathcal{P})$, that is, after performing $\pm r$ the updated hypergraph is the same as the previous one. Since *head(r)* does not belong to $G(\{\pm r\}(\mathcal{P}))$, no argument for *head(r)* exists w.r.t. $\{\pm r\}(\mathcal{P})$, and no argument exists for *head(r)* w.r.t. $\mathcal{P}$ (*cf.* Proposition 1)—arguments for $\mathcal{P}$ are the same as those for $\{\pm r\}(\mathcal{P})$. Thus, every dialectical tree w.r.t. $\mathcal{P}$ is still a dialectical tree w.r.t. $\{\pm r\}(\mathcal{P})$, and the status of all the literals remains unchanged, meaning that $\pm r$ is irrelevant for $\mathcal{P}$; that is, $S_{\mathcal{P}}(\alpha) = S_{\{\pm r\}(\mathcal{P})}(\alpha)$ for any $\alpha \in Lit$.

(Case (*ii*)). Now consider the update $\pm r$ such that $\{head(r), \sim head(r)\} \cap CL(\{\pm r\}, \mathcal{P}) \neq \emptyset$. If $head(r) \in CL(\{\pm r\}, \mathcal{P})$, then no argument for $\sim head(r)$ exists and $S_{\mathcal{P}}(head(r)) = S_{\{\pm r\}(\mathcal{P})}(head(r)) =$ IN because there is a dialectical tree $\mathcal{T}^*_{\langle \mathcal{A},head(r) \rangle}$ for

$\mathcal{P}$ whose root is marked as UNDEFEATED. This dialectical tree still exists after performing $\pm r$ as $head(r)$ is in the deductive closure of both the programs $\mathcal{P}$ and $\{\pm r\}(\mathcal{P})$; the case of $\sim head(r) \in CL(\{\pm r\}, \mathcal{P})$ is analogous. Thus, no dialectical tree will change, meaning that $S_{\mathcal{P}}(\alpha) = S_{\{\pm r\}(\mathcal{P})}(\alpha)$ for any $\alpha \in Lit$.

(Case (iii)). Finally, consider the update $\pm r$ such that $\exists \beta \in body(r)$ s.t. $\sim\beta \in CL(\{\pm r\}, \mathcal{P})$. Rule r cannot belong to any argument $\langle \mathcal{A}, \alpha \rangle$ for $\mathcal{P}$ or for $\{\pm r\}(\mathcal{P})$ because condition ii) of Definition 1 would be violated if $r \in \mathcal{A}$ (in fact, $\mathcal{A}$ union the set of facts and strict rules of $\mathcal{P}$, or $\{\pm r\}(\mathcal{P})$, would be contradictory since $\beta \in CL(\{\pm r\}, \mathcal{P})$). Thus each dialectical tree of $\mathcal{P}$ continues to exist as it is for $\{\pm r\}(\mathcal{P})$, since the set of arguments remains the same after the update. □

Theorem 4. *Let $\mathcal{P}$ be a DeLP program, U a set of updates, and U^{ir} a set of updates returned by function Split (with input $\mathcal{P}$ and U). Then (i) $U^{ir} \subseteq U$ is irrelevant for $\mathcal{P}$, and (ii) the computational complexity of Split is PTIME.*

Proof. Observe that in function *Split* (*i.e.,* Function 1), at each iteration i (with $i \in [1..n]$) of the while loop, an update $u_i \in U$ irrelevant for the updated program $u_{i-1}(\ldots(u_1(\mathcal{P})))$ can be selected and appended to L. Then, $L = [u_1, u_2, \ldots, u_n]$ is such that (i) $u_1 \in U$ is irrelevant for $\mathcal{P}$ and (ii) every $u_j \in U$, with $j \geq 2$, is irrelevant for the program $u_{j-1}(\ldots(u_1(\mathcal{P})))$, as prescribed by Lemma 1. Therefore, $U^{ir} = \{u_1, \ldots, u_n\} \subseteq U$ is irrelevant for $\mathcal{P}$.

As for the computational complexity of *Split*, the number of iterations of the while loop is bounded by $|U|$, at each iteration at most $|U|$ updates are checked for satisfaction of condition of Lemma 2, and checking whether condition of Lemma 2 hold is polynomial, too. □

Proposition 3. *For any DeLP program $\mathcal{P}$ and set of (homogeneous, relevant) updates U, (i) $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}^*_U}(\alpha)$ $\forall \alpha \in Lit_{\mathcal{P}}$, and (ii) $S_{U(\mathcal{P})}(\alpha) = S_{U^*(\mathcal{P}^*_U)}(\alpha) \forall \alpha \in Lit_{U(\mathcal{P})}$.*

Proof. We recall that $\mathcal{P}^*_U = \langle \Pi^*_U, \Delta^*_U \rangle$ is the program derived from $\mathcal{P} = \langle \Pi, \Delta \rangle$ by applying the following steps, where $\Gamma = CL(U, \mathcal{P})$ and ρ_Γ is the rule rewriting function:

1. (*Rule deletion.*) Delete from $\mathcal{P}$ every rule whose head α is related to U w.r.t. $\mathcal{P}$ (*i.e.,* $\alpha \in \mathcal{R}(U, \mathcal{P})$) and having a body literal β such that $\sim\beta \in \Gamma$;
2. (*Rule rewriting.*) $\mathcal{P}^*_U = \langle \Pi^*_U, \Delta^*_U \rangle$ is obtained from the program $\langle \overline{\Pi}, \overline{\Delta} \rangle$ resulting from Step 1 by:
   (i) rewriting the rules of $\langle \overline{\Pi}, \overline{\Delta} \rangle$, that is, $\Pi^*_U \cup \Delta^*_U$ consist of $\rho_\Gamma(r)$ for each $r \in \overline{\Pi} \cup \overline{\Delta}$;
   (ii) adding the fact β'_r to Π^*_U for rule each $r \in \overline{\Pi} \cup \overline{\Delta}$ with literal $\beta \in body(r)$, which is rewritten as r' with body literal β'_r (*i.e.,* $\rho_\Gamma(r)$ yields a rule r' different from r).

We first show that the rule deletions performed in Step 1 correspond to performing a sequence of irrelevant updates. Let Y be the set of rules of $\mathcal{P}$ such that $head(r) \in \mathcal{R}(U, \mathcal{P})$, $\beta \in body(r)$, and $\sim\beta \in \Gamma$. Let $U_Y = \{-r \mid r \in Y\}$ be a set of updates, each of them corresponding to the deletion of a rule in Y.

We show that $U_Y = \{-r \mid r \in Y\}$ is irrelevant for $\mathcal{P}$. In particular, we show that there is a sequence $[u_1, \ldots, u_n]$ of updates, where $u_1 \in U_Y, \ldots, u_n \in U_Y$, such that $u_j \in U_Y$ (with $j \in [1..n]$) is irrelevant for $\mathcal{P}^{j-1} = u_{j-1}(\cdots(u_1(\mathcal{P}))\cdots)$, with $\mathcal{P}^0 = \mathcal{P}$.

For every update $u_j = -r \in U_Y$, since there is $\beta \in body(r)$ such that $\sim\beta \in CL(U, \mathcal{P})$, we have that $\sim\beta \in CL(\mathcal{P})$. Therefore, there is a strict derivation for $\sim\beta$ w.r.t. $\mathcal{P}$. Clearly, such a derivation does not contain the complementary literal β, and thus no rule r of update $u_j = -r \in U_Y$ is involved in that derivation for $\sim\beta$ since $\beta \in body(r)$. Consequently, for each update $u_j = -r \in U_Y$, it is the case that $\sim\beta \in CL(-r(\mathcal{P}))$ and thus $\sim\beta \in CL(\{-r\}, \mathcal{P})$. Therefore, every update $u_j = -r \in U_Y$ is irrelevant for $\mathcal{P}$ due to Lemma 2(iii), and thus $S_{\mathcal{P}}(\alpha) = S_{u_j(\mathcal{P})}(\alpha) \forall \alpha \in Lit \forall u_j \in U_Y$.

Let $[u_1, \ldots, u_n]$ be any arbitrary but fixed sequence of the updates such that $u_1 \in U_Y, \ldots, u_n \in U_Y$. We have that u_1 is irrelevant for $\mathcal{P}$ since, as shown earlier, every update $u_j \in U_Y$ is irrelevant for $\mathcal{P}$. Moreover, since $u_1 = -r$, $\beta \in body(r)$, $\sim\beta \in CL(\{u_1\}, \mathcal{P})$, we have that $\sim\beta \in CL(u_1(\mathcal{P}))$, that is, $\sim\beta \in CL(\mathcal{P}^1)$. For each $j \in [2..n]$, if $u_j = -r \in U_Y$ is such that $\beta \in body(r)$ and $\sim\beta \in CL(\{u_{j-1}\}, \mathcal{P}^{j-2})$, then $\sim\beta \in CL(u_{j-1}(\mathcal{P}^{j-2})) = CL(\mathcal{P}^{j-1})$. Thus, there is a strict derivation for $\sim\beta$ w.r.t. $\mathcal{P}^{j-1}$, such a derivation does not contain β, and no rule r of update $u_j = -r$ is involved in that derivation for $\sim\beta$ since $\beta \in body(r)$. Therefore, for each update u_j, it is the case that $\sim\beta \in CL(u_j(\mathcal{P}^{j-1})) = CL(\mathcal{P}^j)$, and thus $\sim\beta \in CL(\{u_j\}, \mathcal{P}^{j-1})$. That is, every update u_j (with $j \in [2..n]$) is irrelevant for $\mathcal{P}^{j-1}$ (as it satisfies the conditions of the statement of Lemma 2(iii) for $\mathcal{P}^{j-1}$). Then, using the result of Lemma 1 we obtain that U_Y is irrelevant for $\mathcal{P}$, that is $S_{U_Y(\mathcal{P})}(\alpha) = S_{\mathcal{P}}(\alpha)$ $\forall \alpha \in Lit$.

Let $\mathcal{P}^n = U_Y(\mathcal{P})$ be the program obtained by performing the updates in U_Y. From what is said above, U_Y is irrelevant for $\mathcal{P}$, and thus $S_{\mathcal{P}^n}(\alpha) = S_{\mathcal{P}}(\alpha)$ $\forall \alpha \in Lit$. Moreover, since U_Y is irrelevant for $\mathcal{P}$, while U is not, it holds that $U \cap U_Y = \emptyset$.

We now consider the rule rewriting step, and show that the status of every literal of $\mathcal{P}^n = \langle \overline{\Pi}, \overline{\Delta} \rangle$ does not change either w.r.t. $\mathcal{P}^*_U = \langle \Pi^*_U, \Delta^*_U \rangle$.

Let $\Gamma = CL(U, \mathcal{P})$. Since irrelevant updates in U_Y cannot add or remove a strict derivation for a literal in $CL(\mathcal{P})$, $CL(\mathcal{P}^n) = CL(\mathcal{P})$; moreover, $CL(U, \mathcal{P}) = CL(U, \mathcal{P}^n)$. $\mathcal{P}^*_U = \langle \Pi^*_U, \Delta^*_U \rangle$ consists of the rules $\rho_\Gamma(r) = r'$ for each $r \in \overline{\Pi} \cup \overline{\Delta}$, where r' is obtained from r by replacing every literal $\beta \in body(r) \cap \Gamma$ with a fresh literal β'_r; moreover, fact β'_r is added to Π^*_U for each such rule r. This means adding the (fresh) facts in $\mathcal{F}(\mathcal{P}^*_U)$ for some literals in $CL(U, \mathcal{P}) = CL(U, \mathcal{P}^n)$. Therefore,

$CL(\mathcal{P}^*_U) = CL(\mathcal{P}^n) \cup \mathcal{F}(\mathcal{P}^*_U)$. Let R be the set of rules $r \in \overline{\Pi} \cup \overline{\Delta}$ such that $\rho_\Gamma(r)$ is different from r, *i.e.,* the set of rules where at least one literal $\beta \in body(r) \cap \Gamma$ is rewritten as $\beta'_r \in \mathcal{F}(\mathcal{P}^*_U)$. Let $\langle \mathcal{A}, \alpha \rangle$ be an argument for α w.r.t. $\mathcal{P}^n$, and consider how the rewriting step can affect such argument. If α is rewritten as $\alpha'_r \in \mathcal{F}(\mathcal{P}^*_U)$, then $\alpha \in CL(\mathcal{P}^n)$ and $\mathcal{A}$ is empty, and thus $\langle \mathcal{A}, \alpha'_r \rangle$ is an argument for $\mathcal{P}^*_U$; moreover, $\langle \emptyset, \alpha \rangle$ is an argument for $\mathcal{P}^*_U$ as well. If some rule of $\langle \mathcal{A}, \alpha \rangle$ is rewritten, *i.e.,* $\mathcal{A} \cap R \neq \emptyset$, then $\langle \mathcal{A}', \alpha \rangle$ with $\mathcal{A}' = \{\rho_\Gamma(r) \mid r \in \mathcal{A}\}$ is an argument for $\mathcal{P}^*_U$ modulo renaming of some literals. Thus, every argument for $\mathcal{P}^n$ is an argument for $\mathcal{P}^*_U$ modulo renaming some literals, and there are arguments in $\mathcal{P}^*_U$ for the literals $\alpha \in CL(\mathcal{P})$ and $\alpha'_r \in \mathcal{F}(\mathcal{P}^*_U)$. Therefore, every dialectical tree for $\mathcal{P}^n$ continues to be a dialectical tree for $\mathcal{P}^*_U$, with rewritten arguments, and $S_{\mathcal{P}^*_U}(\alpha) = S_{\mathcal{P}^n}(\alpha) = S_{\mathcal{P}}(\alpha)\ \forall \alpha \in Lit \setminus \mathcal{F}(\mathcal{P}^*_U)$ and $S_{\mathcal{P}^*_U}(\alpha'_r) = S_{\mathcal{P}^*_U}(\alpha)\ \forall \alpha'_r \in \mathcal{F}(\mathcal{P}^*_U)$. Thus, $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}^*_U}(\alpha)$ $\forall \alpha \in Lit_{\mathcal{P}}$.

Update U^* rewrites the update U in the language of $\mathcal{P}^*_U$, *i.e.,* using the literals $\alpha' \in \mathcal{F}(\mathcal{P}^*_U)$ in place of the original literals α for rule deletions and introducing new facts α' for the addition of rules having a body literal $\alpha \in CL(U, P) = CL(U, \mathcal{P}^n)$. Since $S_{\mathcal{P}^*_U}(\alpha'_r) = S_{\mathcal{P}^*_U}(\alpha)\ \forall \alpha'_r \in \mathcal{F}(\mathcal{P}^*_U)$ and $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}^*_U}(\alpha)\ \forall \alpha \in Lit_{\mathcal{P}}$, given the definition of the updates in U^*, we have that $S_{U(\mathcal{P})}(\alpha) = S_{U^*(\mathcal{P}^*_U)}(\alpha)\, \forall \alpha \in Lit_{U(\mathcal{P})}$. □

Corollary 2 *(Influenced and Preserved Literals). Let $\mathcal{P}$ be a DeLP program, U a set of (homogeneous) updates, $\mathcal{P}' = U(\mathcal{P})$, and $G(\mathcal{P}') = \langle N', H' \rangle$ the updated labeled hypergraph. Then, every literal $\alpha \in N'$ such that $\alpha \notin \mathcal{I}(U, \mathcal{P})$ is preserved, that is $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$.*

Proof. By Definition 7, if a literal $\alpha \in N'$ is such that $\alpha \notin \mathcal{I}(U, \mathcal{P})$ then $\alpha \notin \mathcal{R}(U^*, \mathcal{P}^*_U)$. Using the result of Theorem 3, α is preserved w.r.t. $\mathcal{P}^*_U$, that is, $S_{\mathcal{P}^*_U}(\alpha) = S_{U^*(\mathcal{P}^*_U)}(\alpha)$. Then, since by Proposition 3 it holds that $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}^*_U}(\alpha)$ and $S_{U^*(\mathcal{P}^*_U)}(\alpha) = S_{U(\mathcal{P})}(\alpha)$, we have that $S_{\mathcal{P}}(\alpha) = S_{U(\mathcal{P})}(\alpha)$, from which the statement follows. □

Theorem 5 *(Status of Inferable Literals). Let $\mathcal{P}$ be a DeLP program, U a set of homogeneous updates, and $\mathcal{P}' = U(\mathcal{P})$. For each literal $\alpha \in Infer(U, \mathcal{P})$, it holds that:*

- $S_{\mathcal{P}'}(\alpha) = \text{OUT}$ *iff* $S_{\mathcal{P}'}(\sim\alpha) = \text{IN}$ *and either* $\sim\alpha \in Core(U, \mathcal{P})$ *or* $\sim\alpha$ *is preserved;*
- $S_{\mathcal{P}'}(\alpha) = \text{UNDECIDED}$ *otherwise.*

Proof. By the definitions of $Infer(U, \mathcal{P})$, it follows that if $\alpha \in Infer(U, \mathcal{P})$ then α does not belong to $G(\mathcal{P}')$, and thus no arguments can be built for it to determine its status w.r.t. $\mathcal{P}'$—the status of α depends on that of its complementary literal $\sim\alpha$.

We first consider the cases that $\sim\alpha$ belongs to $Core(U, \mathcal{P})$ or it is preserved. If $\sim\alpha \in Core(U, \mathcal{P})$ or $\sim\alpha$ is preserved, then the status $S_{\mathcal{P}'}(\sim\alpha)$ will be IN if there is a dialectical tree for it marked UNDEFEATED, otherwise the status will be UNDECIDED (the status of $\sim\alpha$ cannot be OUT since no dialectical tree for α exists, as no argument for it can be built w.r.t. $\mathcal{P}'$). Then the status $S_{\mathcal{P}'}(\alpha)$ is entailed from that of $S_{\mathcal{P}'}(\sim\alpha)$: $S_{\mathcal{P}'}(\alpha) = \text{OUT}$ if $S_{\mathcal{P}'}(\sim\alpha) = \text{IN}$, while $S_{\mathcal{P}'}(\alpha) = \text{UNDECIDED}$ if $S_{\mathcal{P}'}(\sim\alpha) = \text{UNDECIDED}$.

If neither $\sim\alpha \notin Core(U, \mathcal{P})$ nor $\sim\alpha$ is preserved, then $\sim\alpha \in Infer(U, \mathcal{P})$ and thus neither arguments for α nor for $\sim\alpha$ can be built. Therefore, $S_{\mathcal{P}'}(\alpha) = \text{UNDECIDED}$. □

Proposition 4. *Let $\mathcal{P}$ be a DeLP program, U a set of homogeneous updates, and $\mathcal{P}' = U(\mathcal{P})$. For each $\alpha \in Lit$, Algorithm 1 returns $S_{\mathcal{P}'}(\alpha)$.*

Proof. Soundness follows from the fact that Algorithm 1 returns the right status $S_{\mathcal{P}'}(\alpha)$. Specifically, if the whole set of updates is irrelevant, using the result of Theorem 4, $S_{\mathcal{P}'}(\alpha) = S_{\mathcal{P}}(\alpha)$ for each $\alpha \in Lit$ is returned at Line 4. Otherwise, there exists a non-empty set of relevant updates U^r, and the initial status needs to be updated for some literals.

For each rule addition update r whose head $head(r)$ as well as its complementary literal $\sim head(r)$ do not appear in $Lit_{\hat{\mathcal{P}}}$, then the only literal whose status changes after performing the update is $head(r)$ (the status of $\sim head(r)$ remains UNDECIDED). In fact, in this case, the given initial status of all literals carry over to $\mathcal{P}'$—the same marked dialectical trees can be used to obtain the status w.r.t. $\hat{\mathcal{P}}$ and $\mathcal{P}'$ for all literals in $Lit_{\hat{\mathcal{P}}}$. Therefore, the status of these literals $head(r)$ is computed at Line 9, and returned along with the previous status for all the other literals in Lit if U^k is empty (Line 11).

Next, for the updates in U^k, the status of core literals is computed from scratch at Lines 12–13, and the status of inferable literals is computed using the result of Theorem 5. By Theorem 3, the status of preserved literals does not change, and thus remains as initially assigned (see Line 1). Finally, since the literals that are not in $Lit_{\mathcal{P}'}$ are not considered in the set of core literals, in Lines 16–17 the status of the literals that do not belong to the updated program anymore is set to UNDECIDED. This suffices to show the soundness part of the proof.

As for completeness, it follows from the fact that $Lit_{\mathcal{P}'} = \bigcup_{+r \in U^{new}} head(r) \cup PR \cup Core(U^k, \hat{\mathcal{P}}) \cup Infer(U^k, \hat{\mathcal{P}})$, where PR is the set of preserved literals, and that the status of all these literals is either re-computed or confirmed by Algorithm 1. Finally, the status of literals in $Lit \setminus Lit_{\mathcal{P}'}$ is assigned UNDECIDED. □

References

[1] G. Alfano, S. Greco, F. Parisi, G.I. Simari, G.R. Simari, An incremental approach to structured argumentation over dynamic knowledge bases, in: Proceedings of the Sixteenth International Conference on Principles of Knowledge Representation and Reasoning (KR), 2018, pp. 78–87.
[2] A.J. García, G.R. Simari, Defeasible logic programming: an argumentative approach, Theory Pract. Log. Program. 4 (1–2) (2004) 95–138.
[3] P. Besnard, A.J. Garcia, A. Hunter, S. Modgil, H. Prakken, G.R. Simari, F. Toni, Introduction to structured argumentation, Arg. Comput. – Special Issue: Tutorials on Structured Argumentation 5 (1) (2014) 1–4.
[4] S. Modgil, H. Prakken, The ASPIC$^+$ framework for structured argumentation: a tutorial, Arg. Comput. 5 (1) (2014) 31–62.
[5] A. Bondarenko, F. Toni, R.A. Kowalski, An assumption-based framework for non-monotonic reasoning, in: Proceedings of the Second International Workshop on Logic Programming and Non-monotonic Reasoning (LPNMR), 1993, pp. 171–189.
[6] F. Toni, A tutorial on assumption-based argumentation, Arg. Comput. 5 (1) (2014) 89–117.
[7] P. Besnard, A. Hunter, Constructing argument graphs with deductive arguments: a tutorial, Arg. Comput. 5 (1) (2014) 5–30.
[8] A.J. García, G.R. Simari, Defeasible logic programming: DeLP-servers, contextual queries, and explanations for answers, Arg. Comput. 5 (1) (2014) 63–88.
[9] A.J. Garcia, H. Prakken, G.R. Simari, A comparative study of some central notions of ASPIC$^+$ and DeLP, Theory Pract. Log. Program. (2019) 1–33.
[10] P.M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artif. Intell. 77 (2) (1995) 321–358.
[11] I.D. Viglizzo, F.A. Tohmé, G.R. Simari, The foundations of DeLP: defeating relations, games and truth values, Ann. Math. Artif. Intell. 57 (2) (2009) 181–204.
[12] M.O. Moguillansky, N.D. Rotstein, M.A. Falappa, A.J. García, G.R. Simari, Dynamics of knowledge in DeLP through argument theory change, Theory Pract. Log. Program. 13 (6) (2013) 893–957.
[13] M. Thimm, S. Villata, The first international competition on computational models of argumentation: results and analysis, Artif. Intell. 252 (2017) 267–294.
[14] M.A. Falappa, A.J. Garcia, G. Kern-Isberner, G.R. Simari, On the evolving relation between belief revision and argumentation, Knowl. Eng. Rev. 26 (1) (2011) 35–43.
[15] F. Stolzenburg, A.J. García, C.I. Chesñevar, G.R. Simari, Computing generalized specificity, J. Appl. Non-Class. Log. 13 (1) (2003) 87–113.
[16] M. Caminada, S. Sá, J. Alcântara, W. Dvorák, On the equivalence between logic programming semantics and argumentation semantics, Int. J. Approx. Reason. 58 (2015) 87–111.
[17] G. Alfano, S. Greco, F. Parisi, I. Trubitsyna, On the semantics of abstract argumentation frameworks: a logic programming approach, Theory Pract. Log. Program. 20 (5) (2020) 703–718.
[18] P. Baroni, M. Giacomin, On principle-based evaluation of extension-based argumentation semantics, Artif. Intell. 171 (10–15) (2007) 675–700.
[19] P. Baroni, M. Giacomin, B. Liao, On topology-related properties of abstract argumentation semantics. A correction and extension to dynamics of argumentation systems: a division-based method, Artif. Intell. 212 (2014) 104–115.
[20] S. Greco, F. Parisi, Efficient computation of deterministic extensions for dynamic abstract argumentation frameworks, in: Proc. of ECAI, 2016, pp. 1668–1669.
[21] G. Alfano, S. Greco, F. Parisi, Efficient computation of extensions for dynamic abstract argumentation frameworks: an incremental approach, in: Proc. of IJCAI, 2017, pp. 49–55.
[22] G. Alfano, S. Greco, F. Parisi, On scaling the enumeration of the preferred extensions of abstract argumentation frameworks, in: Proc. of SAC, 2019, pp. 1147–1153.
[23] B. Liao, L. Jin, R.C. Koons, Dynamics of argumentation systems: a division-based method, Artif. Intell. 175 (11) (2011) 1790–1814.
[24] B. Liao, H. Huang, Partial semantics of argumentation: basic properties and empirical, J. Log. Comput. 23 (3) (2013) 541–562.
[25] C.E. Briguez, M.C. Budán, C.A.D. Deagustini, A.G. Maguitman, M. Capobianco, G.R. Simari, Towards an argument-based music recommender system, in: Computational Models of Argument - Proceedings of COMMA 2012, Vienna, Austria, September 10–12, 2012, 2012, pp. 83–90.
[26] R. Craven, F. Toni, Argument graphs and assumption-based argumentation, Artif. Intell. 233 (2016) 1–59.
[27] B. Liao, Toward incremental computation of argumentation semantics: a decomposition-based approach, Ann. Math. Artif. Intell. 67 (3–4) (2013) 319–358.
[28] T.J.M. Bench-Capon, P.E. Dunne, Argumentation in artificial intelligence, Artif. Intell. 171 (10-15) (2007) 619–641.
[29] P. Besnard, A. Hunter, Elements of Argumentation, MIT Press, 2008.
[30] I. Rahwan, G.R. Simari, Argumentation in Artificial Intelligence, Springer, 2009.
[31] K. Atkinson, P. Baroni, M. Giacomin, A. Hunter, H. Prakken, C. Reed, G.R. Simari, M. Thimm, S. Villata, Towards artificial argumentation, AI Mag. 38 (3) (2017) 25–36.
[32] S. Modgil, F. Toni, F. Bex, I. Bratko, C.I. Chesñevar, W. Dvorák, M.A. Falappa, X. Fan, S.A. Gaggl, A.J. García, M.P. González, T.F. Gordon, J. a. Leite, M. Mozina, C. Reed, G.R. Simari, S. Szeider, P. Torroni, S. Woltran, Agreement Technologies, Law, Governance and Technology, vol. 8, Springer, New York, 2013, pp. 357–404, Ch. 21: The Added Value of Argumentation: Examples and Challenges.
[33] C. Cayrol, F. Dupin, M. Lagasquie-Schiex, Change in abstract argumentation frameworks: adding an argument, J. Artif. Intell. Res. 38 (2010) 49–84.
[34] R. Baumann, Splitting an argumentation framework, in: Proc. of LPNMR, 2011, pp. 40–53.
[35] E. Oikarinen, S. Woltran, Characterizing strong equivalence for argumentation frameworks, Artif. Intell. 175 (14–15) (2011) 1985–2009.
[36] S. Greco, F. Parisi, Incremental computation of deterministic extensions for dynamic argumentation frameworks, in: Proc. of JELIA, Larnaca, Cyprus, 2016, pp. 288–304.
[37] G. Alfano, S. Greco, F. Parisi, Computing stable and preferred extensions of dynamic bipolar argumentation frameworks, in: Proc. of AI^3 Workshop, co-located with AI*IA, 2017, pp. 28–42.
[38] G. Alfano, S. Greco, F. Parisi, Computing extensions of dynamic abstract argumentation frameworks with second-order attacks, in: Proc. of IDEAS, 2018, pp. 183–192.
[39] G. Alfano, S. Greco, F. Parisi, An efficient algorithm for skeptical preferred acceptance in dynamic argumentation frameworks, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI), 2019, pp. 18–24.
[40] G. Alfano, A. Cohen, S. Gottifredi, S. Greco, F. Parisi, G.R. Simari, Dynamics in abstract argumentation frameworks with recursive attack and support relations, in: Proc. of ECAI, 2020, pp. 577–584.
[41] G. Alfano, S. Greco, Incremental skeptical preferred acceptance in dynamic argumentation frameworks, IEEE Intell. Syst. 36 (2) (2021) 6–12.
[42] G. Alfano, S. Greco, F. Parisi, Incremental computation in dynamic argumentation frameworks, IEEE Intell. Syst. (2021), https://doi.org/10.1109/MIS.2021.3077292.
[43] M.A. Falappa, G. Kern-Isberner, G.R. Simari, Explanations, belief revision and defeasible reasoning, Artif. Intell. 141 (1/2) (2002) 1–28.
[44] M. Capobianco, C.I. Chesñevar, G.R. Simari, Argumentation and the dynamics of warranted beliefs in changing environments, Auton. Agents Multi-Agent Syst. 11 (2) (2005) 127–151.
[45] M. Capobianco, G.R. Simari, A proposal for making argumentation computationally capable of handling large repositories of uncertain data, in: Proc. of SUM, 2009, pp. 95–110.

[46] C.A.D. Deagustini, S.E. Fulladoza Dalibón, S. Gottifredi, M.A. Falappa, C.I. Chesñevar, G.R. Simari, Relational databases as a massive information source for defeasible argumentation, Knowl.-Based Syst. 51 (2013) 93–109.
[47] S. Gottifredi, N.D. Rotstein, A.J. García, G.R. Simari, Using argument strength for building dialectical bonsai, Ann. Math. Artif. Intell. 69 (1) (2013) 103–129.
[48] G. Alfano, S. Greco, F. Parisi, G.I. Simari, G.R. Simari, Incremental computation of warranted arguments in dynamic defeasible argumentation: the rule addition case, in: Proc. of ACM/SIGAPP SAC, 2018, pp. 911–917.
[49] A.J. García, H. Prakken, G.R. Simari, A comparative study of some central notions of ASPIC$^+$ and DeLP, Theory Pract. Log. Program. 20 (3) (2020) 358–390.
[50] J.J. Alferes, L.M. Pereira, T.C. Przymusinski, Belief revision in non-monotonic reasoning and logic programming, Fundam. Inform. 28 (1–2) (1996) 1–22.
[51] J.P. Delgrande, T. Schaub, H. Tompits, S. Woltran, Belief revision of logic programs under answer set semantics, in: Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, 2008, pp. 411–421.
[52] G. Governatori, F. Olivieri, M. Cristani, S. Scannapieco, Revision of defeasible preferences, Int. J. Approx. Reason. 104 (2019) 205–230.
[53] F. Furfaro, S. Greco, C. Molinaro, A three-valued semantics for querying and repairing inconsistent databases, Ann. Math. Artif. Intell. 51 (2–4) (2007) 167–193.
[54] M.A. Leiva, G.I. Simari, G.R. Simari, P. Shakarian, Cyber threat analysis with structured probabilistic argumentation, in: F. Santini, A. Toniolo (Eds.), Proceedings of the 3rd Workshop on Advances In Argumentation In Artificial Intelligence (AI^3) co-located with the 18th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2019), 2019, pp. 50–64.
[55] L.A. Cecchi, P.R. Fillottrani, G.R. Simari, On the complexity of DeLP through game semantics, in: Proc. of NMR, 2006, pp. 386–394.
[56] C.M. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, Massachusetts, 1994.