UNIVERSITÀ DELLA CALABRIA

UNIVERSITA' DELLA CALABRIA

Dipartimento di **Ingegneria Informatica, Modellistica, Elettronica e Sistemistica**

Dottorato di Ricerca in

Information and Communication Technologies

CICLO

XXXII

DYNAMIC ARGUMENTATION IN ARTIFICIAL INTELLIGENCE

Settore Scientifico Disciplinare ING-INF/05

Coordinatore: Ch.mo Prof. Felice Crupi

Firma _ Firma oscurata in base alle linee guida del Garante della privacy

Supervisore/Tutor: Ch.mo Prof. Sergio Greco

Firma_ Firma oscurata in base alle linee guida del Garante della privacy

Co-Tutor: Prof. Francesco Parisi

Firma_ Firma oscurata in base alle linee guida del Garante della privacy

Dottorando: Dott. Gianvincenzo Alfano

Firma _____

Firma oscurata in base alle linee guida del Garante della privacy

to my family,
for their love...

Contents

Preface

Reasoning is usually seen as a means for improving understanding and making better choices. However, there is much evidence that reasoning often leads to epistemic distortions and wrong choices. This indicates that the reasoning process should be rethought. Philosophers Mercier and Sperber formulated the hypothesis that the function of reasoning is argumentative. It is to devise and evaluate arguments intended to persuade [93]. Making the arguments that support own view acceptable directly corresponds to persuade others to accept that view, without forcing the argumentation process but only seducing others to agree on our perspective.

Argumentation is the primary notion of reasoning, and it is flexible as new arguments may added or retracted in order to convince to believe a specific point of view or a position, as well as to accept some evidence. Arguing means to explicitly support a conclusion, a claim or a decision, as well as being able to reject other alternatives by defending against a counterargument. Argumentation is the basic way humans exhibit intelligence since it is native to human reasoning. We all remember the famous Aristotle's postulates formulated 2400 years ago in the ancient Greece, yielding several possible conclusions that can be drawn by arguing. Arguing is then helpful to develop cognition, since it is a universal form of cognitive inference reasoning [85], whose main task is handling conflicts in the real world, which is rich of uncertainty and incompleteness.

People can make errors in arguing because different knowledge schemes (e.g., conclusions) can be constructed when conflicts are resolved. These schemes are dynamic as they are based on personal experiences, which, in turn, evolves over the time because we change our mind several times in our daily life.

Argumentation is then an intelligent process due to its capability to conclude complicate context-based conclusions, yielding in possible fallacious conclusions. Nevertheless, this is the price to pay for creating intelligent systems, as it is very difficult to build intelligent systems that are not fallacious. In this view, trying to find efficient approaches to figure out how conclusions change in a dynamic argumentative context is a challenging problem to investigate.

Rende (CS), Italy
2019

Gianvincenzo Alfano
University of Calabria

1

Introduction

"The key to artificial intelligence has always been the representation."

– Jeff Hawkins

1.1 Argumentation in Artificial Intelligence

It's been a long time (since the time of ancient Greek thinkers and orators) that argumentation experts have looked for conditions that make an argument valid, by some acceptable level of evidence, by analyzing the errors of reasoning that we produce as we attempt to use arguments. Logic has always sought both to identify these fallacies and to manage them. There seemed that neither deductive logic nor obvious formal structure could usefully be applied to them in order to deal with these fallacies. New methods going beyond deductive logic for analyzing and evaluating arguments arose inside a new school of thought called informal logic, as well as a multi-disciplinary class of students associated with the term "argumentation" coming from linguistic fields like speech communication joined with the informal logic group in order to help develop these pragmatic approaches and to relate them to concrete examples of argumentation [34].

The research in the thesis deals with both practical and theoretical aspects of argumentation, namely a model to generally capture the concepts of defeasible reasoning. It consists of reasoning using rules of inference that are not (necessarily) deductively valid. In defeasible reasoning premises provide support for conclusions but do not guarantee their truth (it may be the case that the premises are true while the conclusion is false). This means that conclusions obtained by using defeasible reasoning may have to be retracted when additional information is available. This invalidates the monotonic property, which states that previously drawn conclusions are never revoked when further knowledge join the game. In this perspective, defeasible reasoning is non-monotonic, while classical monotonic reasoning is good for modeling formal or mathematical reasoning. Once the validity of a mathematical theorem

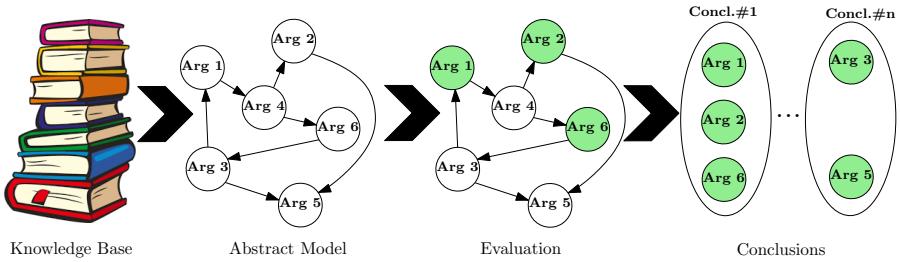


Fig. 1.1: The abstract argumentation process.

is proved, it is not called into question anymore, even if additional knowledge are taken into account.

Starting in the '80s, much work was done on defeasible and non-monotonic reasoning in the wide area of Artificial Intelligence. This is due to the fact that classical monotonic logics is not useful for commonsense reasoning because the monotonicity constraint is mostly violated in common-sense reasoning—it suffices to think about how many times we make use of assumptions, rules with exception or rules of thumb. Argumentation is at the root of human reasoning and interaction, it is also about how decisions are taken and opinions are shaped and how decisions and beliefs are legitimized and understood by people. That is why it is a central discipline in the area of artificial intelligence.

An important question that has deserved the attention of the AI field over the last twenty years and which leads to the development of a wide research area called formal argumentation has been the integration of human capacities to convince people and draw conclusions within smart and sophisticated systems. In fact, in the last twenty years argumentation has emerged as a distinct field within Artificial Intelligence by reflecting in philosophy, law and formal logic [25, 29, 102]. As it was said briefly before, argumentation in AI differs from classical logic. In fact, in logic one proves statements. If a proof exists then the statement is not refutable (i.e., what is proven correct once, remains correct). In contrast, the aim of arguments is to persuade, not to formally prove statements [25]. Moreover, arguments are defeasible. Thus, what made an argument convincing, might not be convincing anymore in the light of new information, and thus retracting conclusions is common when arguing, yielding argumentation non-monotonic.

Reasoning in an automated way is one of the main objectives towards the development of artificial intelligent systems and thus, argumentation provides a particular challenge because of the complexity related to its non-monotonicity. Nevertheless, argumentation has found its way into a variety of applications, such as in legal reasoning [24, 26], decision support systems [10], E-Democracy tools [46, 47], E-Health tools [111], medical applications [76, 76], multi-agent systems [69], and also in social networks [86] where threads' conclusions are taken by using concepts of argumentation theory. All these approaches and systems make use of some sort of abstraction, which is a cardinal point of the argumentation process, which can be

synthesized as a four step process (see Figure 1.1). It starts with the first step by defining a knowledge base containing conflicting information about a topic of interest. Then, an abstract model consisting in a representation of the arguments inside the knowledge base as well as conflicts between them is built in the second step. In this step it is very widespread to abstain the internal structure of the arguments [79, 101], in fact an argument here is an abstract entity such as a letter, a word, or a phrase of a discourse, or eventually, a propositional logical formula [31]. Several frameworks for the abstract representation have been proposed in literature [40]. Third step refers to evaluate arguments. This is done by calling an algorithm for identifying a set of acceptable arguments that can stay together without invalidating some semantic constraints. According to the applied constraints the semantics may change, yielding in a possible different set of acceptable arguments. From the latter, conclusions are drawn for the domain of interest in the last step.

While argumentation is an inherently dynamic activity, the definition of evaluation algorithms and the analysis of the computational complexity taking into account such dynamic aspects have been mostly neglected, whereas in these situations incremental computation techniques could greatly improve performance. However, a better understanding of the behavior and applicability of the argumentation theory requires to consider a dynamic perspective. This perspective leads to several questions related to updating the argumentation framework behind the argumentation process. Then, the goal of the thesis is to provide efficient algorithms for several problems of argumentation theory in a dynamic perspective. Before precisely stating the addressed problems, some preliminary notions will be outlined in the next section.

1.2 Argumentation Frameworks

A milestone in formal argumentation theory is the Dung's seminal work [61] where the fundamental concepts of *abstract argumentation framework* (AF) and *argumentation semantics* were introduced. An AF consists of a set of *arguments* and a binary *attack relation* over them. Simply put, an argument is a potentially complex sentence or statement. For instance, we may have an argument `a` defined as "*John is a knight*" and argument `b` defined as "*John is a knave*". However, in Dung's AFs, the focus in neither on the internal structure of arguments nor on the way they are obtained but it is on the relationships between arguments which are expressed by the attack relation—that is why arguments are said to be *abstract*. Thus, the role of an argument is entirely determined by its connections with other arguments. Basically, an AF can be viewed as a directed graph whose nodes represent the arguments while the edges represent attacks between arguments. An attack from an argument a toward an argument b intuitively means that if a is accepted to support a point of view in a discussion then b should be rejected; on the other hand b can be accepted if all its attacker are not.

Consider the following example, inspired from the famous "Knights and Knaves" Smullyan's logic puzzle [105] (also used in [83, 104] to illustrate logic programming semantics). *We are in an island inhabited by knights and knaves. The knaves always*

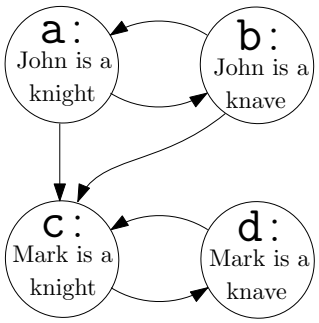


Fig. 1.2: An AF for modelling the Smullyan's puzzle of our example.

lie, while the knights always tell the truth. Coming upon a pair of individuals, John and Mark, a detective questions them about their nature. Then John answers: "Only one of us is a knight". What can the detective infer about Mark?

To answer the detective question, let us model the puzzle throughout an AF. First, we need to model the conclusions that the detective can reach on the nature of the two individuals by (abstract) arguments. Hence we use the arguments `a` and `b` introduced earlier plus the arguments `c`: "*Mark is a knight*" and `d`: "*Mark is a knave*". These four arguments and the interaction between them, which we explain below, are shown in the argumentation graph in Figure 1.2. Arguments `a` and `b`, as well as `c` and `d`, attack each other since they represent conflicting information: `a` and `b` can not be both true; similarly for `c` and `d`. Moreover, the attack from `a` to `c` models the fact that, if John is a knight (i.e., we accept argument `a`), then Mark cannot be a knight (i.e., we cannot accept `c`) because knights always tell the truth and John said that only one of them is a knight. Finally, argument `b` attacks `c` since if John is a knave, and thus he is lying (i.e., it holds that not only one of them is a knight), then Mark cannot be a knight; otherwise John was telling the truth, contradicting the behaviour entailed by his nature.

After having defined the structure of an AF as in our example, to figure out which sets of arguments, called *extensions*, collectively represent a plausible conclusion, we rely on the concept of argumentation *semantics*. All the semantics for abstract argumentation frameworks we considered are based on the auxiliary concepts of *conflict-free* and *admissible* sets, which we define next.

We say that a set of arguments is *conflict-free* iff it does not contain two distinct arguments a and b such that a attacks b (i.e., there is an edge between a and b in the argumentation graph). As an example, for the AF in Figure 1.2, we have that both $\{\texttt{a}, \texttt{d}\}$ and $\{\texttt{b}, \texttt{d}\}$ are conflict-free sets.

Let us now generalize the notion of attack from a single argument toward another argument to the case of a set S attacking an argument: we say that a set S of arguments attacks an argument b if there exists an argument a in S such that a attacks b. For instance, for the AF in Figure 1.2, we have that $\{\texttt{a}, \texttt{b}\}$ attacks `c`. Then, we say that a set S *defends* an argument a iff S attacks every argument attacking a. In

our example, we have that $\{\mathtt{a}\}$ defends $\mathtt{d}$ from the attack coming from $\mathtt{c}$. Also, $\{\mathtt{a}\}$ defends itself from the attack from $\mathtt{b}$. An *admissible* set a is conflict-free set that defends all its arguments. In our example, among others, both $\{\mathtt{a}, \mathtt{d}\}$ and $\{\mathtt{b}, \mathtt{d}\}$ are admissible sets (observe that that the empty set is also admissible).

It is worth noting that admissibility does not require to consider all the arguments that are defended by the set under consideration. For instance, $\{\mathtt{a}\}$ and $\{\mathtt{b}\}$ are admissible sets but both could be enlarged to consider argument $\mathtt{d}$ they defend. This behaviour is avoided by the *complete* (`co`) semantics, and by the other argumentation semantics which are also based on completeness. A *complete extension* is an admissible set that contains all the arguments that it defends. In our example, the complete extensions are the empty set, $\{\mathtt{d}\}$, $\{\mathtt{a}, \mathtt{d}\}$, and $\{\mathtt{b}, \mathtt{d}\}$.

Other argumentation semantics prescribe sets of extensions which are subsets of the set of complete extensions, thus considering additional constraints over the set of complete extensions. In particular, a complete extension is said to be: *preferred* (`pr`) iff it is maximal (w.r.t. set inclusion); *stable* (`st`) iff it attacks every argument not included in the extension; *ideal* (`id`) iff it is the biggest (w.r.t the set inclusion) admissible set which is contained in every preferred extension; *grounded* (`gr`) iff it is minimal (w.r.t. set inclusion).

Thus, while grounded semantics captures the most cautious viewpoint across all the possible complete extensions, the preferred semantics aims to accept as many arguments as possible. Moreover, a stable extension ensures that every argument outside the extension is rejected because attacked by the extension. However, a stable extension may not exists. For instance, if there is a self attacking argument, or there is a directed triangle in the argumentation graph, the set of stable extensions is empty. All the other semantics are guaranteed to prescribe at least one extension, and the grounded and ideal semantics prescribes exactly one extension—they are called *deterministic* or *unique-status* semantics. Finally, the ideal semantics provides the biggest deterministic set of arguments that can be collectively accepted.

For a given AF the set of conflict-free (`cf`) sets includes the set of admissible (`ad`) sets, which in turns includes the set of complete extensions. The set of stable extensions is contained in the set of preferred extensions, which is contained in the set of complete extensions. In addition, it is well-known that, for a given AF, the grounded extension is included in the ideal extension, which in turns in included in every preferred extension.

Continuing with our example, for the AF shown in Figure 1.2, different conclusions can be derived under different semantics. In particular, the grounded extension is empty, while the stable and preferred extensions are $\{\mathtt{a}, \mathtt{d}\}$ and $\{\mathtt{b}, \mathtt{d}\}$. Then the ideal extension is $\{\mathtt{d}\}$, telling us that the detective can conclude that Mark is a knave, irrespectively of the nature of John.

Except for the grounded semantics that can be computed by a fix-point polynomial approach, the complexity of computing argumentation semantics is intractable[63, 65, 68, 87]. In this regard, the International Competition on Computational Models

of Argumentation (ICCMA) [1] has been established for nurturing research and development of efficient algorithms for computational models of argumentation [108].

1.3 Dynamic Argumentation and Research Questions

Although most research in formal argumentation have focused on *static* frameworks (i.e. frameworks whose structure does not change over time), AFs are frequently used for modelling dynamic systems [19, 20, 57, 71, 89]. This is not surprising since, as a matter of fact, the argumentation process is inherently dynamic. For instance, think of how many times we change our mind after knowing something new about a situation we are reasoning about. There is clear evidence of that in social networks threads [86], where users frequently post new arguments against or supporting other posts, often made by the same users that change their mind. In general, an update for an AF consists in adding or retracting new arguments and/or attacks. As an example, for the AF shown in Figure 1.2, suppose that a police man arrive on the island and say that he knows that Mark is not a knave. This means adding a new argument h: "The police man is sure that Mark is not a knave" which attacks argument d. Clearly, the set of conclusions (i.e., extensions according to a specific semantics) may change after performing an update on a given AF. In our example, the updated set of stable and preferred extensions is $\{\{\mathtt{a},\mathtt{h}\},\{\mathtt{b},\mathtt{h}\}\}$, while the ideal and the grounded extensions are both equal to $\{\mathtt{h}\}$. Of course we could add further arguments and attacks to model for instance the fact that the detective trust or not the police man; this would entail other changes to the set of extensions for the considered AF after applying new updates.

Therefore, given an AF, the question is: After performing an update, *could we avoid to compute the updated extensions (prescribed by a given semantics) from scratch?*

The Proposed Approach

In the thesis, we show that we can profitably use the information provided by the initial extensions to efficiently and incrementally recompute extensions after performing an update (addition or deletion of arguments and/or attacks). This allow us to substantially reduce the computation effort in many cases, and the improvement is even more significant for the argumentation semantics which suffer from high computational complexity.

Our proposal relies on identifying a (potentially small) part of the input AF, called *influenced set*, that represents the arguments that, intuitively, are involved by a given update—the influenced set depends on both the update and an input initial extension for the AF. Then, the computation of extensions is made on a smaller AF, called *reduced AF*, which is obtained from the initial AF by the arguments and attacks induced by the influenced set plus additional arguments and attacks that take into account the external context needed to perform the computation.

[1] http://argumentationcompetition.org

The ideas behind the incremental approach proposed is applied to different kinds of abstract argumentation frameworks [40]. Particularly, it can also be applied to Bipolar Argumentation Frameworks (BAFs), able to model the notion of supports between arguments. While attacks in Dung's original framework represents conflicting information between arguments, the support relation wants to model a positive relation, that is, an argument is is in favor of other arguments. Thanks to a meta-argumentation approach, a BAF can be converted into a Dung's AF by adding meta-arguments able to model supports by means of Dung's attacks.

Furthermore, other abstract argumentation frameworks have been considered, such as Extended Abstract Argumentation Frameworks (EAFs for short) which are used to model attacks towards an argument or an attack. Similarly to what done for BAFs, an EAF can be converted into a Dung's AF by using additional meta-arguments as well as attacks between them.

1.4 Related Work

Overviews of key concepts in argumentation theory and of formal models of argumentation in the field of Artificial Intelligence can be found in [12, 25, 30, 103]. Further discussion regarding uses of computational argumentation as an Agreement Technology can be found in [97]. Several computational problems of AFs have been studied such as skeptical and credulous reasoning, existence of a non-empty extension, and verifying if a set of arguments is an extension under different argumentation semantics [63, 65, 67, 68]. The complexity of the problem of computing all extensions according to some semantics for AFs has been recently investigated in [87], where it was shown that the enumeration problem is intractable under several argumentation semantics. An approach for dividing the problem of enumerating the preferred extensions into sub-problems is proposed in [56], where a meta-algorithm based on SCC-recursiveness [18] is introduced. However, this kind of approaches provide advantages only if there are many strongly connected components, or in case of sparse AFs (i.e., average degree less than 2) [92]. A comprehensive introduction to (static) abstract argumentation frameworks (AFs) can be found in [14]. Although the idea underlying AFs is simple and intuitive, most of the semantics proposed so far suffer from a high computational complexity [63, 65, 67, 68, 72, 73, 74, 75]. Complexity bounds and evaluation algorithms for AFs have been deeply studied in the literature but most of this research focused on static frameworks, whereas, in practice, AFs are not static systems [19, 20, 57, 71, 89]. [37, 38] have investigated the principles according to which a grounded extension of a Dung's abstract argumentation frameworks does not change when the set of arguments/attacks are changed.

Dung's abstract argumentation framework has been extended along several dimensions (e.g. [15, 96, 113]), and there have been several significant efforts coping with dynamics aspects of abstract argumentation. [48, 49] have addressed the problem of revising the set of extensions of an AF, and studied how the extensions can evolve when a new argument is considered. They focus on adding only one argument interacting with one initial argument (i.e. an argument which is not attacked by any

other argument). [32] have studied the evolution of the set of extensions after performing a change operation (addition/removal of arguments/interaction). Dynamic argumentation has been applied to decision-making of an autonomous agent by [11], where it is studied how the acceptability of arguments evolves when a new argument is added to the decision system. Other relevant works on dynamic aspects of Dung's argumentation frameworks include the following. [20] have proposed an approach exploiting the concept of splitting of logic programs to deal with dynamic argumentation. The technique considers weak expansions of the initial AF, where added arguments never attack previous ones. [23] have investigated whether and how it is possible to modify a given AF so that a desired set of arguments becomes an extension, whereas [99] have studied equivalence between two AFs when further information (another AF) is added to both AFs. [21] have focused on expansions where new arguments and attacks may be added but the attacks among the old arguments remain unchanged, while [22] have characterized update and deletion equivalence, where adding/deleting arguments/attacks is allowed (deletions were not considered by [21, 99]). Approaches for dividing AFs into subgraphs have been explored also in the context of dynamic AFs—AFs which are updated by adding/removing (set of) attacks/arguments— for which the set of extensions evolves. The division-based method, proposed by [89] and then refined by [19], divides the updated framework into two parts: *affected* and *unaffected*, where only the status of affected arguments is recomputed after updates. Using the results in [89], [91] investigated the efficient evaluation of the justification status of a subset of arguments in an AF (instead of the whole set of arguments), and proposed an approach based on answer-set programming for *local* computation. In [90], an AF is decomposed into a set of strongly connected components, yielding sub-AFs located in layers, which are then used for incrementally computing the semantics of the given AF by proceeding layer by layer. Recently, [114] introduced a matrix representation of argumentation frameworks and proposed a matrix reduction that, when applied to dynamic argumentation frameworks, resembles the division-based method in [89].

Bipolarity in argumentation is discussed in [9], where a survey of the use of bipolarity is given, as well as a formal definition of BAF that extends the Dung's concept of argumentation framework by including supports is provided. The notion of support has been found useful in many application domains, including decision making [8]. However, as discussed in [58], different interpretations of the concept of support have been proposed. The acceptability of arguments in the presence of the support relation was first investigated in [50]. Then, to handle bipolarity in argumentation, [51, 52] proposed an approach based on using the concept of *coalition* of arguments, where arguments are grouped together, and defeats occur between coalitions. However, when considering a deductive interpretation of support [36, 113], coalitions may lead to counterintuitive results [58], though they are useful in contexts where support is interpreted differently. Changes in bipolar argumentation frameworks have been studied in [53], where it is shown how the addition of one argument together with one support involving it (and without any attack) impacts the extensions of the updated BAF. Although several related work, no one of the these works

neither tackled the problem of incremental computing argumentation semantics nor experimentally analyzed.

1.5 Organization

After having retraced the role of argumentation in artificial intelligence, as well as discussed the motivations behind the research conducted in the thesis, we now describe the structure of the thesis.

This work is organized in seven chapters that consider results for abstract argumentation (Chapters 3-6) and structured argumentation (Chapter 7). After the introduction given in this Chapter, we will present:

Chapter 2. Preliminary notions for abstract argumentation frameworks are presented. Additionally, key concepts behind argumentation theory as argumentation semantics, extensions and labellings are also presented. Chapter 2 ends with a discussion of different kinds of argumentation frameworks, each of them having interesting properties.

Chapter 3. In this chapter, we propose algorithms for efficiently computing both the set of preferred and semi-stable extensions of a given AF. Our technique relies on first computing the ideal (resp. grounded) extension for the given AF, and then using it to prune some arguments so that a smaller AF is obtained.

Chapter 4. We tackle the problem of incrementally computing extensions for dynamic AFs: given an initial extension and an update (or a set of updates), we devise a technique for computing an extension of the updated AF under several well-known semantics. The idea is to identify a reduced (updated) AF sufficient to compute an extension of the whole AF and use state-of-the-art algorithms to recompute an extension of the reduced AF only.

Chapter 5. In this chapter we first tackle the problem of efficiently recomputing sets of accepted arguments over dynamic Bipolar Argumentation Frameworks (BAFs for short), namely frameworks allowing for both positive and negative interactions between arguments. Focusing on a deductive interpretation of the support relation, we introduce an incremental approach that, given an initial BAF, an initial extension for it, and an update, computes an extension of the updated BAF. This is achieved by introducing a meta-argumentation transformation according to which an initial BAF, as well as its extension and an update, are transformed into a plain argumentation framework with a suitable initial extension and update. Moreover, the proposed approach can be seamlessly applied to a more general form of BAFs, namely Extended Bipolar Argumentation Frameworks (EAFs), where defeasible supports and defeats are modelled by means of second-order attacks (i.e., attacks toward elements of the support or attack relation).

Chapter 6. In this chapter we devise an efficient algorithm for computing the skeptical preferred acceptance in dynamic AFs. More specifically, we investigate

how the skeptical acceptance of an argument (goal) evolves when the given AF is updated and propose an efficient algorithm for solving this problem.

Chapter 7. This chapter introduces key concepts behind structured argumentation and presents one of its famous formalism called Defeasible Logic Programming (DeLP for short). Then, it presents a technique for the incremental computation of warranted arguments of dynamic DeLP programs, namely, programs whose set of rules may contain ”defeasible” rules that may change over time thought updates.

Finally, conclusions and some directions for future work are drawn at the end of the thesis.

All the algorithms that will be presented in the thesis were experimentally analyzed. Particularly, all experiments conducted in Chapters 3-6 were carried out on an Intel Core i7-3770K CPU 3.5GHz with 12GB RAM running Ubuntu 16.04, while experiments conducted in Chapter 7 were carried out on an Intel Core i7-3630QM CPU 2.4GHz with 8GB RAM running Windows 8.1.

2

Fundamental Concepts for Abstract Argumentation

"Don't raise your voice, improve your argument."

– Desmond Tutu

We start by presenting in Section 2.1 the seminal Abstract Argumentation Framework (AF) proposed in [61], whose basic concepts were extended in several ways, by considering for instance other kinds of interaction between arguments. Particularly, for each section, we present and discuss preliminary concepts behind each abstract argumentation framework addressed in the thesis. For each of the proposed frameworks, the notion of argumentation semantics (i.e., ways conclusion can be drawn) and updates (i.e., adding/removing new/old knowledge in the framework) are presented.

2.1 Abstract Argumentation Frameworks

We assume the existence of a set Arg of *arguments*. An *(abstract) argumentation framework* [61] (*AF*) is a pair $\langle A, \Sigma \rangle$, where $A \subseteq Arg$ is a finite set whose elements are referred to as *arguments*, and $\Sigma \subseteq A \times A$ is a binary relation over A whose elements are called *attacks*. Thus, an AF can be viewed as a directed graph where nodes correspond to arguments and edges correspond to attacks. Thus, an argument is an abstract entity whose role is entirely determined by its relationships with other arguments.

Example 2.1. Let $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ be an AF where $A_0 = \{a, b, c, d, e, f, g, h\}$ and $\Sigma_0 = \{(a, b), (b, a), (b, c), (c, c), (d, a), (d, e), (e, d), (b, e), (f, e), (g, d), (g, h), (h, e), (h, f)\}$. The AF $\mathcal{A}_0$ is shown on Figure 2.1. □

Given an AF $\langle A, \Sigma \rangle$ and arguments $a, b \in A$, we say that a *attacks* b iff $(a, b) \in \Sigma$, and that a set $S \subseteq A$ attacks b iff there is $a \in S$ attacking b. We use $S^+ = \{b \mid \exists a \in S : (a, b) \in \Sigma\}$ to denote the set of all arguments that are attacked by S.

Moreover, we say that $S \subseteq A$ *defends* a iff $\forall b \in A$ such that b *attacks* a, there is $c \in S$ such that c *attacks* b. A set $S \subseteq A$ of arguments is said to be:

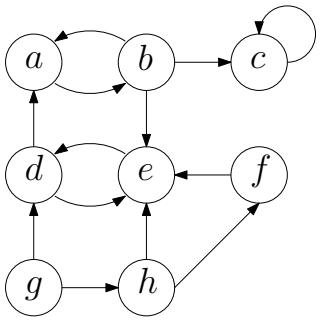


Fig. 2.1: AF $\mathcal{A}_0$

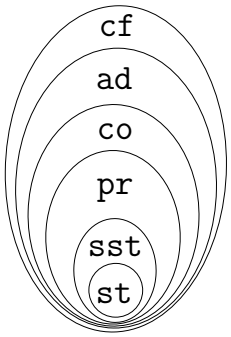


Fig. 2.2: Relations among semantics

- *conflict-free*, if there are no $a, b \in S$ such that a *attacks* b;
- *admissible*, if it is conflict-free and it defends all its arguments.

An argumentation semantics specifies the criteria for identifying a set of arguments that can be considered "reasonable" together, called *extension*.

A *complete extension* (co) is an admissible set that contains all the arguments that it defends.

A complete extension S is said to be:

- *preferred (pr)* iff it is maximal (w.r.t. $\subseteq$);
- *stable (st)* iff it attacks every argument in $A \setminus S$;
- *semi-stable (sst)* iff $S \cup \mathcal{S}^+$ is maximal (w.r.t. $\subseteq$).
- *grounded (gr)* iff it is minimal (w.r.t. $\subseteq$);
- *ideal (id)* iff it is contained in every preferred extension and it is maximal (w.r.t. $\subseteq$).

Given an AF $\mathcal{A}$ and a semantics $\mathcal{S} \in \{\mathtt{co}, \mathtt{pr}, \mathtt{st}, \mathtt{sst}, \mathtt{gr}, \mathtt{id}\}$, we use $\mathcal{E}_{\mathcal{S}}(\mathcal{A})$ to denote the set of $\mathcal{S}$-extensions for $\mathcal{A}$, i.e., the set of extensions for $\mathcal{A}$ according to the given semantics $\mathcal{S}$.

All the above-mentioned semantics except the stable admit at least one extension, and the grounded and ideal admits exactly one extension [41, 61, 62]. Semantics gr and id are called *deterministic* or *unique status* as $|\mathcal{E}_{\mathtt{gr}}(\mathcal{A})| = |\mathcal{E}_{\mathtt{id}}(\mathcal{A})| = 1$, whereas the other above recalled semantics are called *nondeterministic* or *multiple status*. It is well-known that, for any AF $\mathcal{A}$, each grounded extension is also a complete extension (i.e., $\mathcal{E}_{\mathtt{gr}}(\mathcal{A}) \subseteq \mathcal{E}_{\mathtt{co}}(\mathcal{A})$). Moreover, the following relations among different sets of extensions for each semantics holds: $\mathcal{E}_{\mathtt{id}}(\mathcal{A}) \subseteq \mathcal{E}_{\mathtt{co}}(\mathcal{A})$, and $\mathcal{E}_{\mathtt{st}}(\mathcal{A}) \subseteq \mathcal{E}_{\mathtt{sst}}(\mathcal{A}) \subseteq \mathcal{E}_{\mathtt{pr}}(\mathcal{A}) \subseteq \mathcal{E}_{\mathtt{co}}(\mathcal{A})$.

For a given AF, Figure 2.2 shows the relationship between the sets of extensions prescribed by each semantics $\mathcal{S} \in \{\mathtt{co}, \mathtt{pr}, \mathtt{st}, \mathtt{sst}, \mathtt{id}, \mathtt{gr}\}$. Particularly, the set of conflict-free (cf) sets (resp. preferred (pr) sets) includes the set of admissible (ad) sets (resp. semi-stable (sst) sets), which in turn includes the set of complete (resp. stable (st) extensions (co) extension. Finally, the set of grounded and ideal

extensions are included in the set of complete extensions but they are not shown in the figure as they may or not be included in the set of preferred extensions.

Example 2.2. The set of admissible sets for the AF $\mathcal{A}_0$ shown in Fig. 2.1 is $\{\emptyset, \{b\}, \{g\}, \{a, g\}, \{b, g\}, \{f, g\}, \{a, g, f\}, \{b, g, f\}\}$, and $\mathcal{E}_{\mathcal{S}}(\mathcal{A}_0)$ with $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{sst}, \texttt{id}, \texttt{gr}\}$ is as reported in the second column of Table 2.1. □

$\mathcal{S}$	$\mathcal{E}_{\mathcal{S}}(\mathcal{A}_0)$	$\mathcal{E}_{\mathcal{S}}(\mathcal{A})$
co	$\{\{f, g\}, \{a, f, g\}, \{b, f, g\}\}$	$\{\{g\}, \{a, g\}, \{b, f, g\}\}$
pr	$\{\{a, f, g\}, \{b, f, g\}\}$	$\{\{a, g\}, \{b, f, g\}\}$
st	$\{\{b, f, g\}\}$	$\{\{b, f, g\}\}$
id	$\{\{f, g\}\}$	$\{\{g\}\}$
gr	$\{\{f, g\}\}$	$\{\{g\}\}$

Table 2.1: Sets of extensions for $\mathcal{A}_0$ and $\mathcal{A} = +(c, f)(\mathcal{A}_0)$.

2.1.1 Labelling and Status of arguments

The argumentation semantics can be also defined in terms of *labelling* [14]. A labelling for an AF $\mathcal{A} = \langle A, \Sigma \rangle$ is a total function $L : A \to \{\text{IN}, \text{OUT}, \text{UNDECIDED}\}$ assigning to each argument a label. $L(a) = \text{IN}$ means that argument a is accepted, $L(a) = \text{OUT}$ means that a is rejected, while $L(a) = \text{UNDECIDED}$ means that a is undecided.

Let $in(L) = \{a \mid a \in A \wedge L(a) = \text{IN}\}$, $out(L) = \{a \mid a \in A \wedge L(a) = \text{OUT}\}$, and $un(L) = \{a \mid a \in A \wedge L(a) = \text{UNDECIDED}\}$. In the following, we also use the triple $\langle in(L), out(L), un(L)\rangle$ to represent the labelling L.

Given an AF $\mathcal{A} = \langle A, \Sigma \rangle$, a labelling L for $\mathcal{A}$ is said to be *admissible (or legal)* if $\forall a \in in(L) \cup out(L)$ it holds that

(*i*) $L(a) = \text{OUT}$ iff $\exists\, b \in A$ such that $(b, a) \in \Sigma$ and $L(b) = \text{IN}$; and
(*ii*) $L(a) = \text{IN}$ iff $L(b) = \text{OUT}$ for all $b \in A$ such that $(b, a) \in \Sigma$.

Moreover, L is a *complete* labelling iff conditions (*i*) and (*ii*) hold for all $a \in A$.

Between complete extensions and complete labellings there is a bijective mapping defined as follows: for each extension E there is a unique labelling $L = \langle E, E^+, A \setminus (E \cup E^+)\rangle$ and for each labelling L there is a unique extension $in(L)$. We say that L is the labelling *corresponding* to E.

Example 2.3. Continuing Example 2.2, $\langle \{a, f, g\}, \{b, d, e, h\}, \{c\}\rangle$ is the labelling corresponding to the preferred extension $E_{\text{pr}} \in \mathcal{E}_{\text{pr}}(\mathcal{A}_0) = \{a, f, g\}$, as shown in Figure 2.3. □

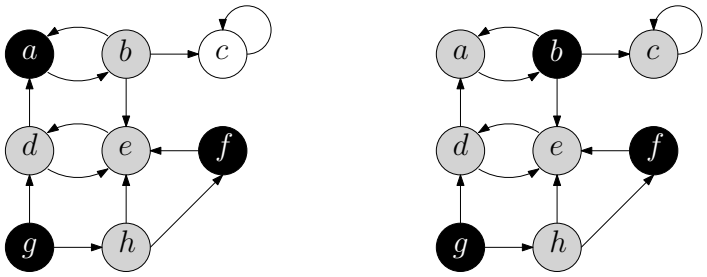


Fig. 2.3: Labelling L according to the preferred extensions $E_{\mathtt{pr}} \in \mathcal{E}_{\mathtt{pr}}(\mathcal{A}_0) = \langle\{a, f, g\}, \{b, d, e, h\}, \{c\}\rangle$ and $E'_{\mathtt{pr}} \in \mathcal{E}_{\mathtt{pr}}(\mathcal{A}_0) = \langle\{b, f, g\}, \{a, d, e, h\}, \{c\}\rangle$. Black (resp., grey and white) nodes x are such that $L(x) =$ IN (resp., OUT and UNDECIDED).

In the following, we say that the *status of an argument* a w.r.t. a labelling L (or its corresponding extension $in(L)$) is IN (resp., OUT, UNDECIDED) iff $L(a) =$ IN (resp., $L(a) =$ OUT, $L(a) =$ UNDECIDED). We will avoid to mention explicitly the labelling (or the extension) whenever it is understood.

2.1.2 Skeptical acceptance

Given an AF $\mathcal{A} = \langle A, \Sigma\rangle$ and an argument $g \in A$, and a semantics $\mathcal{S}$, we say that g is *skeptically accepted* w.r.t. $\mathcal{A}$ under semantics $\mathcal{S}$ iff for each $\mathcal{S}$-extension E in $\mathcal{E}_{\mathcal{S}}(\mathcal{A})$ it holds that $g \in E$.

Throughout this work, we will focus on the skeptical acceptance of a given argument under the preferred semantics, and thus we will avoid to explicitly mention the semantics when referring to skeptical acceptance of an argument.

In the following, we use $SA_{\mathcal{A}}(g)$ to denote the *skeptical acceptance* (i.e. either *true* or *false*) of argument g w.r.t. AF $\mathcal{A}$ (under the preferred semantics).

Example 2.4. For AF $\mathcal{A}_0$ of Example 2.1, we have that the arguments skeptically accepted are f, and g as they belong to the preferred extensions shown in Table 2.1. Thus, $SA_{\mathcal{A}_0}(f)$ is *true*, and so is for $SA_{\mathcal{A}_0}$(g), while for any other argument x, $SA_{\mathcal{A}_0}(x) =$*false* as shown in Figure 2.4. □

2.2 Bipolar Argumentation Frameworks

Bipolar argumentation frameworks (BAFs) extend Dung's argumentation frameworks to explicitly represent the notion of support between arguments, in addition to that of attack. BAFs can be profitably used to model disputes between two or more agents, with the aim of deciding the sets of arguments that should be accepted to support a point of view in a discussion.

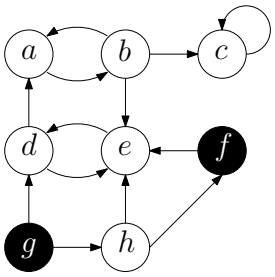


Fig. 2.4: Black (resp. white) arguments are (resp. are not) skeptically accepted under the preferred semantics (i.e., $SA_{\mathcal{A}_0}(f) = SA_{\mathcal{A}_0}(g) = true$).

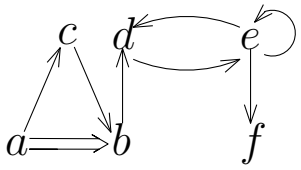


Fig. 2.5: BAFs $\mathcal{B}_0$ of Example 2.5.

We assume again the existence of a set Arg of *arguments*. An *abstract bipolar argumentation framework* (*BAF* for short) [9] is a triple $\langle A, \Sigma, \Pi \rangle$, where (i) $A \subseteq Arg$, (ii) $\Sigma \subseteq A \times A$, (iii) $\Pi \subseteq A \times A$ is a binary relation over A whose elements are called *supports*, and (iv) $\Sigma \cap \Pi = \emptyset$. Thus, a Dung's argumentation framework (AF) [61] is a BAF of the form $\langle A, \Sigma, \emptyset \rangle$.

A BAF can be viewed as a directed graph where each node corresponds to an argument and each edge in the graph corresponds to either an attacks or a support. Given a BAF $\mathcal{B}$, the *bipolar interaction graph* for $\mathcal{B}$ (denoted as $\mathcal{G}_{\mathcal{B}}$) has two kinds of edges: one for the attack relation ($\rightarrow$) and another one for the support relation ($\Rightarrow$), as shown in the following example.

Example 2.5. Consider a BAF $\mathcal{B}_0 = \langle A_0, \Sigma_0, \Pi_0 \rangle$ where:

- $A_0 = \{a, b, c, d, e, f\}$ is the set of arguments;
- $\Sigma_0 = \{(a, c),\ (c, b),\ (b, d),\ (d, e), (e, d),\ (e, e),$
  $(e, f)\}$ is the set of attacks;
- $\Pi_0 = \{(a, b)\}$ is the set of supports;

The bipolar interaction graph $\mathcal{G}_{\mathcal{B}_0}$ is shown in Fig. 2.5.

□

Several interpretations of the notion of support have been proposed in the literature [58]. We focus on *deductive* support [36] which is intended to capture the following intuition: if argument a supports argument b then the acceptance of a implies the acceptance of b, and thus the non-acceptance of b implies the non-acceptance of a. Given this interpretation of support, the coexistence of the support and attack relations in BAFs entails that new kinds of "implicit" attacks should be considered, as explained in what follows.

Given a BAF $\langle A, \Sigma, \Pi \rangle$, a *supported attack* for an argument $b \in A$ by argument $a_1 \in A$ is a sequence $a_1 \Pi a_2 \Pi \ldots \Pi a_n \Sigma\, b$ with $n \geq 1$. Note that a direct attack $a_1 \Sigma\, b$ is a supported attack. Thus a supported attack is a (possibly empty) chain of supports followed by an attack. Moreover, we say that there is a *mediated attack* for

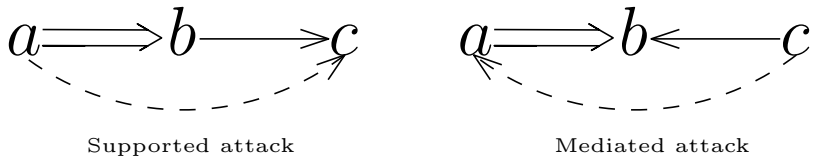


Fig. 2.6: Supported and mediated attacks.

argument a_1 by argument b if there is an attack $b \Sigma a_n$ and a sequence of supports $a_1 \Pi a_2 \Pi \ldots \Pi a_n$ with $n \geq 1$. Thus, for a mediated attack the chain of supports ends in a_n which is attacked by b. Supported and mediated attacks are illustrated in Figure 2.6, where a chain consisting of only one support is considered. It is easy to see that the BAF of Example 2.5 contains a supported attack from argument a to d, and a mediated attack from argument c to a.[1]

2.2.1 Extensions

Given a BAF $\langle A, \Sigma, \Pi \rangle$, we say that a set $S \subseteq A$ *set-attacks* an argument $b \in A$ iff there exists a supported or mediated attack for b by an argument $a \in S$. We use S^+ to denote the set of arguments set-attacked by S. Moreover, we say that a set $S \subseteq A$ *defends* an argument $a \in A$ iff for each $b \in A$ such that $\{b\}$ set-attacks a, it is the case that S set-attacks b (i.e., $b \in S^+$).

Given a BAF $\langle A, \Sigma, \Pi \rangle$, a set $S \subseteq A$ is *conflict-free* iff there are no arguments $a, b \in S$ such that $\{a\}$ set-attacks b. Moreover, a conflict-free set $S \subseteq A$ is said to be *admissible* iff it defends all of its arguments.

Example 2.6. Continuing with Example 2.5, for the BAF $\mathcal{B}_0$ it is easy to see that $\{a\}$ defends argument b (as $\{a\}$ set-attacks c and $\{c\}$ set-attacks b). The set $\{a, b\}$ is conflict-free as neither $\{a\}$ set-attacks b nor $\{b\}$ set-attacks a, while $\{a, d\}$ is not conflict-free as $\{a\}$ set-attacks d (by means of the supported attack (a, d)). Moreover, $S = \{c, d, f\}$ is an admissible set as it is conflict-free and S defends all of its arguments: $\{c\}$ defends itself from a by the mediated attack from c to a; $\{d\}$ is defended by c, and f is defended by d. The set of admissible sets for $\mathcal{B}_0$ is $\{\{\emptyset\}, \{a\}, \{c\}, \{a, b\}, \{c, d\}, \{c, d, f\}\}$. □

Given a BAF $\langle A, \Sigma, \Pi \rangle$, a *preferred extension* (`pr`) for it is an admissible set which is a maximal (w.r.t $\subseteq$). Furthermore, a conflict-free set $S \subseteq A$ is a *stable extension* (`st`), if and only if it set-attacks all the arguments in $A \setminus S$. (Note that this implies that S is admissible).

[1] Another kind of implicit attack which we do not consider because of the deductive interpretation of support is the *secondary attack* [52], which occurs when in a BAF there is a sequence $b \Sigma a_1 \Pi a_2 \Pi \ldots \Pi a_n$ with $n \geq 1$. Considering supported and secondary attacks leads to an alternative interpretation of support [52]. However, when considering a deductive interpretation of support, secondary attacks may lead to counterintuitive results [58], though they are useful in contexts where support is interpreted differently.

Given a BAF $\mathcal{B}$ and a semantics $\mathcal{S} \in \{\texttt{pr}, \texttt{st}\}$, we use $\mathcal{E}_{\mathcal{S}}(\mathcal{B})$ to denote the set of extensions for $\mathcal{B}$ according to $\mathcal{S}$. For the BAF $\mathcal{B}_0$ of Example 2.5, we have that the set of the stable extensions is $\mathcal{E}_{\texttt{st}}(\mathcal{B}_0) = \{\{c, d, f\}\}$, while the set of the preferred extensions is $\mathcal{E}_{\texttt{pr}}(\mathcal{B}_0) = \{\{a, b\}, \{c, d, f\}\}$. □

2.2.2 Labelling and status of arguments

Following the approach of [14], where argumentation semantics have been characterized in terms of *labelling*, we define a labelling function for BAFs. A labelling for a BAF $\mathcal{B} = \langle A, \Sigma, \Pi \rangle$ is a total function $L : A \rightarrow \{\text{IN}, \text{OUT}, \text{UNDECIDED}\}$ assigning to each argument a label: $L(a) = \text{IN}$ means that argument a is accepted, $L(a) = \text{OUT}$ means that a is rejected, while $L(a) = \text{UNDECIDED}$ means that a is undecided.

As for the case of AFs, we denote as $in(L) = \{a \mid a \in A \wedge L(a) = \text{IN}\}$, $out(L) = \{a \mid a \in A \wedge L(a) = \text{OUT}\}$, and $un(L) = \{a \mid a \in A \wedge L(a) = \text{UNDECIDED}\}$. In the following, we also use the triple $\langle in(L), out(L), un(L) \rangle$ to represent the labelling L.

Given a BAF $\mathcal{B} = \langle A, \Sigma, \Pi \rangle$, a labelling L for $\mathcal{B}$ is said to be *admissible (or legal)* if $\forall a \in in(L) \cup out(L)$ it holds that (*i*) $L(a) = \text{OUT}$ iff $\exists\, b \in A$ such that $a \in \{b\}^+$ and $L(b) = \text{IN}$; and (*ii*) $L(a) = \text{IN}$ iff $L(b) = \text{OUT}$ for all $b \in A$ such that $a \in \{b\}^+$. Moreover, L is a *complete* labelling iff conditions (*i*) and (*ii*) hold for all $a \in A$.

Between complete extensions and complete labellings there is a bijective mapping defined as follows: for each extension E there is a unique labelling $L = \langle E, E^+, A \backslash (E \cup E^+) \rangle$ and for each labelling L there is a unique extension $in(L)$. We say that L is the labelling *corresponding* to E. For instance, considering the BAF $\mathcal{B}_0$ of Example 2.5, the labelling corresponding to the preferred extension $E = \{a, b\}$ is $L = \langle \{a, b\}, \{c, d\}, \{e, f\} \rangle$.

2.2.3 Meta-Argumentation Framework

The semantics of BAFs can be also given in terms of meta-argumentation frameworks (i.e., Dung's AFs) where additional (meta-)arguments and attacks are considered to model deductive support. The following construction, introduced in [36], will be used as the basis for defining the meta-argumentation framework of our incremental approach.

Definition 2.7 (Meta-AF [36]). *Given a BAF $\mathcal{B} = \langle A, \Sigma, \Pi \rangle$, the meta-AF for $\mathcal{B}$ is $\mathcal{M} = \langle A^m, \Sigma^m \rangle$ where:*

i) $A^m = A \cup \{X_{a,b}, Y_{a,b} \mid (a, b) \in \Sigma\} \cup \{Z_{a,b} \mid (a, b) \in \Pi\}$
ii) $\Sigma^m = \{(a, X_{a,b}), (X_{a,b}, Y_{a,b}), (Y_{a,b}, b) \mid (a, b) \in \Sigma\} \cup \{(b, Z_{a,b}), (Z_{a,b}, a) \mid (a, b) \in \Pi\}$

The meaning of meta-arguments $X_{a,b}$, $Y_{a,b}$ and $Z_{a,b}$ is as follows. $X_{a,b}$ represents the fact that the corresponding attack (a, b) is "not active" in $\mathcal{B}$—it belongs to an extension for $\mathcal{M}$ iff a does not belong to an extension for $\mathcal{B}$. On the other hand,

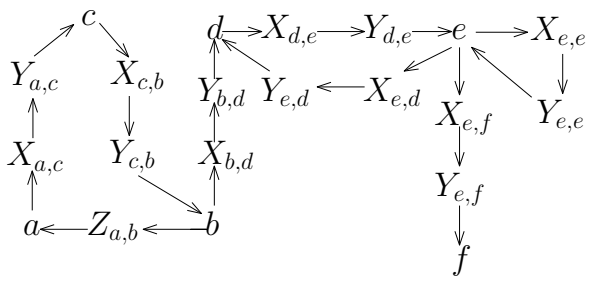


Fig. 2.7: Meta-AF $\mathcal{M}_0$ for the BAF $\mathcal{B}_0$ of Figure 2.5.

$Y_{a,b}$ represents the fact that (a, b) is "active" in $\mathcal{B}$, and it belongs to an extension for $\mathcal{M}$ iff argument b does not. Finally, meta-argument $Z_{a,b}$ represents a support relation between a and b: it does not belong to an extension for $\mathcal{M}$ iff the supported argument b is accepted in the deductive model of support. As an example, the (interaction graph of the) meta-AF $\mathcal{M}_0$ for the BAF $\mathcal{B}_0$ of Example 2.5 is shown in Figure 2.7.

The following proposition characterizes the relationship between the extensions of a given BAF and the extensions of the corresponding meta-AF.

Proposition 2.8 ([5]). *Let $\mathcal{B} = \langle A, \Sigma, \Pi \rangle$ be a BAF, $\mathcal{M}$ the meta-AF for $\mathcal{B}_0$, and $\mathcal{S} \in \{\mathtt{pr}, \mathtt{st}\}$ a semantics. For each extension $E \in \mathcal{E}_{\mathcal{S}}(\mathcal{B})$, there is an extension $E^m \in \mathcal{E}_{\mathcal{S}}(\mathcal{M})$ such that $E = E^m \cap A$, and vice versa.*

Example 2.9. For the meta-AF $\mathcal{M}_0$ of the BAF $\mathcal{B}_0$ in Example 2.5, we have the following preferred extensions (which are also stable extensions): (i) $\{a, b, Y_{a,c}, X_{c,b}, Y_{b,d}, X_{d,e}\}$, which corresponds to the extension $\{a, b\}$ for BAF $\mathcal{B}_0$ of Example 2.5, and (ii) $\{c, d, f, X_{a,c}, Y_{c,b}, X_{b,d}, Y_{d,e}, X_{e,d}, X_{e,e}, X_{e,f}, Z_{a,b}\}$, which corresponds to the extension $\{c, d, f\}$ for $\mathcal{B}_0$. □

2.3 Second Order Attacks

Second-order attacks [36] for BAFs are, (i) attacks from an argument to another attack, and (ii) attacks from an argument to a support. This allows the representation of both attacks towards the attack relation [15, 95] and a kind of *defeasible support* where the support itself can be attacked.[2]

In the following, after presenting the formal definition of BAFs extended with second-order attacks, as well as the formalization of updates for the extended framework, we build on the definition of meta-AF introduced in [36] for encoding second-order attacks.

As discussed before, an argument a deductively support the argument b if the acceptance of a implies the acceptance of b. In this section we extend our technique

[2] The technique can be further extended to consider (second-order) attacks from an attack to another attack [36].

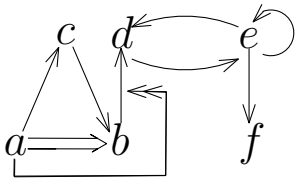


Fig. 2.8: EAF $\mathcal{EB}_0$ of Example 2.10.

by considering also defeasible support, which means that the implication holds only by default and it can be attacked.

An *Extended Bipolar Argumentation Framework* (EAF for short) [36] is a quadruple $\langle A, \Sigma, \Pi, \Delta \rangle$, where $\langle A, \Sigma, \Pi \rangle$ is a BAF and Δ is a binary relation over $A \times (\Sigma \cup \Pi)$ whose elements are called *second-order attacks*.

In the following, a second-order attack from an argument a to an attack (b, c) will be denoted as $(a \twoheadrightarrow (b \rightarrow c))$, while an attack from an argument a to a support (b, c) will be denoted as $(a \twoheadrightarrow (b \Rightarrow c))$.

Example 2.10. $\mathcal{EB}_0 = \langle A_0, \Sigma_0, \Pi_0, \Delta_0 \rangle$ is an EAF where $\Delta_0 = \{(a, (b, d))\}$ is the set of second-order attacks. Its graph is shown in Fig. 2.8, where second-order attacks are drawn using double-headed arrows. □

The semantics of an EAF can be given by means of the following meta-AF, which extends that in Definition 2.7 by taking into account second order attacks.

Definition 2.11 (Meta-AF with Second-Order Attacks [36]). *The meta-AF for* $\mathcal{EB} = \langle A, \Sigma, \Pi, \Delta \rangle$ *is* $\mathcal{M} = \langle A^m, \Sigma^m \rangle$ *where:*

$$
\begin{aligned}
A^m \;=\; & A \cup \{X_{a,b}, Y_{a,b} \mid (a,b) \in \Sigma\} \cup \{Z_{a,b} \mid (a,b) \in \Pi\} \cup \\
& \{X_{a,(b,c)}, Y_{a,(b,c)} \mid (a,(b,c)) \in \Delta, (b,c) \in \Sigma\} \\
\Sigma^m \;=\; & \{(a, X_{a,b}), (X_{a,b}, Y_{a,b}), (Y_{a,b}, b) \mid (a,b) \in \Sigma\} \cup \\
& \{(b, Z_{a,b}), (Z_{a,b}, a) \mid (a,b) \in \Pi\} \cup \\
& \{(a, X_{a,(b,c)}), (X_{a,(b,c)}, Y_{a,(b,c)}), (Y_{a,(b,c)}, Y_{b,c}) \mid \\
& \qquad\qquad (a,(b,c)) \in \Delta, (b,c) \in \Sigma\} \cup \\
& \{(a, X_{a,(b,c)}), (X_{a,(b,c)}, Y_{a,(b,c)}), (Y_{a,(b,c)}, Z_{b,c}) \mid (a,(b,c)) \in \Delta, (b,c) \in \Pi\}
\end{aligned}
$$

Thus, an attack of the form $(a \twoheadrightarrow (b \rightarrow c))$ is encoded as an attack towards the meta-argument $Y_{b,c}$ (that represents the fact that (b, c) is "active"), while an attack of the form $(a \twoheadrightarrow (b \Rightarrow c))$ is encoded as an attack toward the meta-argument $Z_{b,c}$. The meta-AF for the EAF of Example 2.10 is shown in Fig. 2.9.

Analogously to what stated in Proposition 2.8, extensions for an EAF $\mathcal{EB}$ are obtained from extensions for its meta-AF: E is an $\mathcal{S}$-extension for $\mathcal{EB}$ iff $E^m \in \mathcal{E}_{\mathcal{S}}(\mathcal{M})$ and $E = E^m \cap A$. Using this relationship, the notion of labelling can be extended to EAFs as well.

Example 2.12. For the meta-AF $\mathcal{M}$ of Fig. 2.9, we have the following preferred extensions (which are also stable extensions): (i) $\{a,\ b,\ d,\ f,\ Y_{a,c},\ X_{c,b},\ Y_{d,e},$

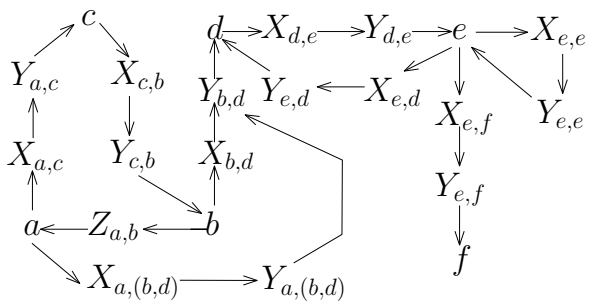


Fig. 2.9: Meta-AF for the the EAF of Example 2.10.

$Y_{a,(b,d)}$, $X_{e,e}$, $X_{e,d}$, $X_{e,f}$, $\}$, which corresponds to the extension $\{a, b, d, f\}$ of the EAF of Example 2.10, and (ii) $\{c, d, f,\ X_{a,c},\ Y_{c,b},\ Z_{a,b},\ X_{b,d},\ Y_{d,e}\ X_{e,e},\ X_{e,d},\ X_{e,f},\ X_{a,(b,d)}\}$, which corresponds to the extension $\{c, d, f\}$ for the EAF of Example 2.10. □

2.4 Dynamic Argumentation Frameworks

Although most research in argumentation focused on static frameworks (i.e., frameworks not changing over the time), argumentation frameworks are often used to model dynamic systems [19, 20, 57, 71, 89]. In fact, usually an AF represents a temporary situation, and new arguments, attacks, and supports can be added/retracted to take into account new available knowledge. For instance, for disputes among users of online social networks [6, 86], arguments, attacks, and supports are continuously added/retracted by users to express their point of view in response to the last move made by the adversaries (often disclosing as few arguments/attacks as possible).

2.4.1 Updates

Performing an update on an AF $\mathcal{A}_0$ means modifying it into an AF $\mathcal{A}$ by adding or removing arguments or attacks.

We use $+(a, b)$, with $a, b \in A_0$ and $(a, b) \notin \Sigma_0$, (resp. $-(a, b)$, with $(a, b) \in \Sigma_0$) to denote the addition (resp. deletion) of an attack (a, b), and $u(\mathcal{A}_0)$ to denote the application of update $u = \pm(a, b)$ to AF $\mathcal{A}_0$ (where $\pm$ means either $+$ or $-$). Applying an update u to an AF implies that its semantics (set of extensions or labellings) changes, as shown by Table 2.1 which reports the sets of extensions for the AFs of Figure 2.1 and Figure 2.10 before and after performing the update $+(c, f)$.

Concerning the addition (resp. deletion) of a set of isolated arguments, it is easy to see that if $\mathcal{A}$ is obtained from $\mathcal{A}_0$ through the addition (resp. deletion) of a set S of isolated argument, then, let E_0 be an extension for $\mathcal{A}_0$, $E = E_0 \cup S$ (resp. $E = E_0 \setminus S$) is an extension for $\mathcal{A}$ that can be trivially computed. Of course, if arguments in S are not isolated, for addition we can first add isolated arguments and

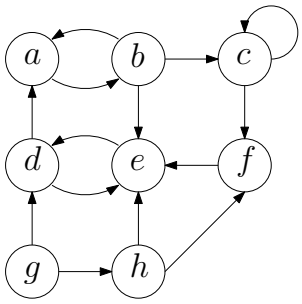


Fig. 2.10: AF $\mathcal{A} = +(c, f)(\mathcal{A}_0)$

then add attacks involving these arguments, while for deletion we can first delete all attacks involving arguments in S. Thus we do not consider these kinds of update in the following.

Multiple Updates

Performing a set of updates $U = \{+(a_1, b_1), \ldots, +(a_n, b_n), -(a'_1, b'_1), \ldots, -(a'_m, b'_m)\}$ on $\mathcal{A}_0$ can be reduced to performing a single update $+(v, w)$ on an AF whose definition depends on both the set of updates U and the initial $\mathcal{S}$-extension E_0, as detailed in what follows.

Given a set U of updates for an AF $\mathcal{A}_0$, and an $\mathcal{S}$-extension E_0 for $\mathcal{A}_0$, we use U^* to denote the subset of (relevant updates) U that does not contain those updates such that the initial extension (under a given semantics) is still an extension for the updated AF. We will see it more in detail on Proposition 4.1.

The AF $\mathcal{A}^U_{E_0}$ for applying a set U^* of relevant updates E_0 is obtained from $\mathcal{A}_0$ by (i) adding arguments x_i, y_i and the chain of attacks between a_i and b_i as shown in Figure 2.11, for each update $+(a_i, b_i) \in U^*$; (ii) replacing each attack (a'_j, b'_j) in $\mathcal{A}_0$ with the chain of attacks between a'_j and b'_j as shown in Figure 2.11, for each update $-(a_j, b_j) \in U^*$; and (iii) adding the new arguments v, w, w' and the attacks involving them as shown in Figure 2.11. The following definition considers a general set of updates which includes also irrelevant updates.

Definition 2.13 (AF for applying a set of updates). Let $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ be an AF, and E_0 an $\mathcal{S}$-extension for $\mathcal{A}$. Let $\Sigma^+ \subseteq (A_0 \times A_0) \setminus \Sigma_0$, and $\Sigma^- \subseteq \Sigma_0$ such that $\Sigma^+ \cap \Sigma^- = \emptyset$ be two sets of attacks. Let $U = \{+(a_i, b_i) \,|(a_i, b_i) \in \Sigma^+\} \cup \{-(a_j, b_j) \,|(a_j, b_j) \in \Sigma^-\}$ be a set of updates, and $U^* \subseteq U$ be the set of relevant updates w.r.t. E_0 and $\mathcal{S}$. Then, $\mathcal{A}^U_{E_0} = \langle A^U, \Sigma^U \rangle$ denotes the AF obtained from $\mathcal{A}$ as follows:

- $A^U = A_0 \cup \{x_i, y_i \mid +(a_i, b_i) \in U^*\} \cup \{x'_j, y'_j \mid -(a_j, b_j) \in U^*\} \cup \{v, w, w'\}$, where all $x_i, y_i, x'_j, y'_j, w, w'$, and v are new arguments not occurring in A, and

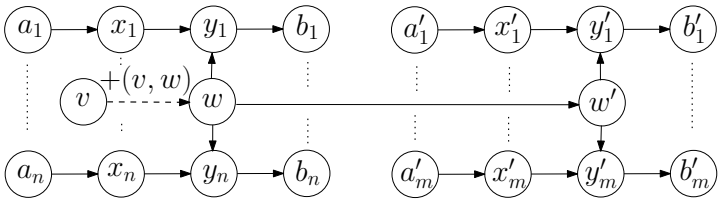


Fig. 2.11: Simulating multiple updates by a single one.

- $\Sigma^U = (\Sigma_0 \setminus \Sigma^-) \cup \{(a_i, b_i) \mid +(a_i, b_i) \in (U \setminus U^*)\} \cup$
  $\{(a_i, x_i), (x_i, y_i), (y_i, b_i) \mid +(a_i, b_i) \in U^*\} \cup$
  $\{(a_j, x'_j), (x'_j, y'_j), (y'_j, b_j) \mid -(a_j, b_j) \in U^*\} \cup$
  $\{(w, y_i) \mid +(a_i, b_i) \in U^*\} \cup$
  $\{(w', y'_j) \mid -(a_j, b_j) \in U^*\} \cup \{(w, w')\}$.

The following theorem states that every extension of the AF obtained by performing on $\mathcal{A}_0$ all the updates in U corresponds to an extension of $+(v,w)(\mathcal{A}^U_{E_0})$, where $+(v,w)$ is a single attack update.

Theorem 2.14. *Let $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ be an AF, E_0 be an $\mathcal{S}$-extension of A_0 with $\mathcal{S} \in \{\mathtt{co}, \mathtt{pr}, \mathtt{st}, \mathtt{gr}\}$, and U a set of updates. Let $\mathcal{A}$ be the AF obtained from $\mathcal{A}_0$ by performing all updates in U on it. Then, for any semantics in $\mathcal{S}$, $E \in \mathcal{E}_\mathcal{S}(\mathcal{A})$ iff there is $E^U \in \mathcal{E}_\mathcal{S}(+(v,w)(\mathcal{A}^U_{E_0}))$ such that $E^U \cap A_0 = E$.*

Proof. Given $E \in \mathcal{E}_\mathcal{S}(\mathcal{A})$, let E^U be as follows.

$E^U = E \cup \{v, w'\} \cup \{x_i | x_i \in A^U_0 \wedge (a_i, x_i) \in \Sigma^U_0 \wedge a_i \in E^+\} \cup$
$\{x'_i | x'_i \in A^U_0 \wedge (a'_i, x'_i) \in \Sigma^U_0 \wedge a'_i \in E^+\} \cup$
$\{y_i | y_i \in A^U_0 \wedge (a_i, x_i), (x_i, y_i) \in \Sigma^U_0 \wedge a_i \in E\}$.

Let L and L^U be the labelling corresponding to E and E^U, respectively. Observe that i) $L^U(y_i) = L^U(a_i)$ since y_i is attacked only by argument w whose status is OUT and argument x_i whose status is IN (resp., OUT, UNDECIDED) if the status of a_i is OUT (resp., IN, UNDECIDED); ii) $L^U(y'_i) = \text{OUT}$. Condition i) means that the presence of attack (a_i, b_i) in $\mathcal{A}$ can be simulated by attack (y_i, b_i) in AF $+(u,v)(\mathcal{A}^U_0)$; Condition ii) means that the absence of attack (a'_i, b'_i) in $\mathcal{A}$ can be simulated by attack (y'_i, b'_i), with y'_i being OUT, in the AF $+(u,v)(\mathcal{A}^U_0)$. Therefore (a) $L^U(a_i) = L(a_i)$ since the attackers of a_i in $+(u,v)(\mathcal{A}^U_0)$ are either already present in $\mathcal{A}$ or replaced in (resp. deleted from) $+(u,v)(\mathcal{A}^U_0)$ by means of arguments y_i whose status is the same as that of the attackers in $\mathcal{A}$ (resp. y'_i whose status is OUT); essentially, the status of the other attackers of a_i is equivalent w.r.t. the two labellings; similarly, we have that (b) $L^U(a'_i) = L(a'_i)$; (c) $L^U(b_i) = L(b_i)$ since $L^U(y_i) = L^U(a_i) = L(a_i)$; (d) $L^U(b'_i) = L(b'_i)$ since $L^U(y'_i) = \text{OUT}$. Given this, it is easy to check that E^U is an extension for $+(u,v)(\mathcal{A}^U_0)$ iff $E^U \cap A_0$ is an extension for $\mathcal{A}$. □

2.4.2 Updates for BAFs

An *update* u for a BAF $\mathcal{B}_0$ allows us to change $\mathcal{B}_0$ into a BAF $\mathcal{B}$ by adding or removing an argument, an attack, or a support. The addition (resp., deletion) of an argument a will be denoted as $+a$ (resp. $-a$), whereas the addition (resp., deletion) of an attack from a to b will be denoted as $+(a \to b)$ (resp., $-(a \to b)$). Moreover, the addition (resp., deletion) of a support from a to b will be denoted as $+(a \Rightarrow b)$ (resp., $-(a \Rightarrow b)$).

We will use $u(\mathcal{B}_0)$ to denote the BAF resulting from the application of update u to the initial BAF $\mathcal{B}_0$. For instance, for the BAF $\mathcal{B}_0 = \langle A_0, \Sigma_0, \Pi_0 \rangle$ of Example 2.5, if $u = +(f \Rightarrow b)$, we have that $u(\mathcal{B}_0) = +(f \Rightarrow b)(\mathcal{B}_0) = \langle A_0, \Sigma_0, \Pi_0 \cup \{(f, b)\} \rangle$; on the other hand, if $u = -(b \to d)$, we have that $u(\mathcal{B}_0) = \langle A_0, \Sigma_0 \setminus \{(b, d)\}, \Pi_0 \rangle$.

Applying an update u to a BAF implies that its semantics (set of extensions or labellings) may change. For the BAF $\mathcal{B}_0$ of Example 2.5 and the update $u = +(f \Rightarrow b)$, we have that the set of the stable extensions for the updated BAF $\mathcal{B} = +(f \Rightarrow b)(\mathcal{B}_0)$ is $\mathcal{E}_{\mathtt{st}}(\mathcal{B}) = \{\{c, d\}\}$, while the set of the preferred extensions is $\mathcal{E}_{\mathtt{pr}}(\mathcal{B}) = \{\{a, b\}, \{c, d\}\}$. In fact, the addition of the support between f and b entails that additional implicit attacks must be considered: a supported attack between f and d, and a mediated one between c and f.

In the following, for the sake of the presentation, we consider only *feasible* updates which are defined as follows. Adding an argument as well as removing an attack/support are feasible updates. The deletion of an argument is feasible only if a is *isolated*, that is there is no argument b attacking/supporting a or being attacked/supported by a, where a is not necessarily distinct from b. The addition of an attack (resp., support) between a and b is feasible only if a and b are arguments of the initial BAF $\mathcal{B}_0$ and there is no already a support (resp. attack) between a and b in $\mathcal{B}_0$.

Clearly, general updates can be simulated by a sequence of feasible updates. For instance, a non isolated argument a can be deleted after deleting all attacks and supports involving a (by performing a sequence of feasible updates). Analogously, adding an attack (resp., a support) between an argument a in $\mathcal{B}_0$ and a fresh argument $b \notin \mathcal{B}_0$ can be simulated as a sequence of updates of the form $+b, +(a \to b)$ (resp., $+b, +(a \Rightarrow b)$).

Finally, observe that if a BAF $\mathcal{B}$ is obtained from $\mathcal{B}_0$ through the addition (resp. deletion) of a set S of isolated argument, then, let E_0 be an extension for $\mathcal{B}_0$, it is the case that $E = E_0 \cup S$ (resp. $E = E_0 \setminus S$) is an extension for $\mathcal{B}$ that can be easily computed. Thus, in the following we do not discuss further updates of the form $+a$ or $-a$. That is, we will focus on updates of the forms $\pm(c \to d)$ and $\pm(e \Rightarrow f)$, where $\pm$ means either $+$ or $-$.

2.4.3 Second-Order Updates

In addition to the kinds of updates introduced in Section 2.4.2, for EAFs we also consider additions and deletions of second-order attacks. Specifically, the addition (resp., deletion) of a second-order attack from an argument a to an attack (b, c) will

be denoted as $+(a \twoheadrightarrow (b \rightarrow c))$ (resp., $-(a \twoheadrightarrow (b \rightarrow c))$). Similarly, if (b, c) is a support, then the update will be denoted as $+(a \twoheadrightarrow (b \Rightarrow c))$ (resp., $-(a \twoheadrightarrow (b \Rightarrow c))$).

3

Extensions Enumeration Problem

"Time spent arguing is, oddly enough, almost never wasted."

– Christopher Hitchens

Enumerating the sets of arguments (i.e., *extensions*) prescribed by an argumentation semantics is arguably one of the most challenging problems for AFs, and this is particularly the case for the well-known *preferred* and *semi-stable* semantics.

In this chapter, we propose an algorithm for efficiently computing the set of preferred and semi-stable extensions of a given AF. The proposed technique relies on first computing the ideal (resp. grounded) extension for the given AF, and then using it to prune some arguments so that a smaller AF is obtained. Finally, we use state-of-the-art solvers for enumerating the preferred (resp. semi-stable) extensions of the pruned AF, and then return the extensions of the input AF after obtaining them from those of the pruned AF.

We experimentally compared the technique with the solvers of the International Competition on Computational Models of Argumentation, and found that the proposed approach is orders of magnitude faster than the computation from scratch.

3.1 Computing Preferred and Semi-Stable Extensions

Although the idea underlying AFs is very simple and intuitive, most of the argumentation semantics proposed so far suffer from a high computational complexity [63, 65, 68, 73, 74]. In particular, the enumeration problem of AFs (i.e., the problem of computing all extensions according to some semantics) is intractable for several argumentation semantics [64, 87], including the well-known *preferred* and *semi-stable* semantics [39, 43].

Complexity bounds and evaluation algorithms for AFs have been deeply investigated in the literature, and the International Competition on Computational Models of Argumentation (ICCMA) [1] has been established for promoting research and

[1] `http://argumentationcompetition.org`

development of efficient algorithms for computational models of AFs. Challenging computational tasks of ICCMA are EE-pr and EE-sst, that is, enumerating all the extensions of a given AF under the preferred and semi-stable semantics, respectively.

In this chapter, we propose an approach for scaling up the computation of the EE-pr and EE-sst problems. The approach enables us to make improvements over the performance of the solvers for EE-pr and EE-sst.

The main contributions of the chapter are as follows:

- We define the concept of *pruned AF* (resp. *cut-AF*) that allows us to compute all the preferred (resp. semi-stable) extensions by focusing only on a smaller AF. In particular, the pruned (resp. cut-) AF is built by removing from the whole AF the arguments (as well as their relationships) belonging to the ideal (resp. grounded) extension, which is an appropriate set of arguments contained in every preferred (resp. semi-stable) extensions [63].
- We introduce an efficient algorithm for computing all the preferred (resp. semi-stable) extensions. The algorithm enables the computation of the preferred (resp. semi-stable) extensions by focusing only on the pruned (resp. cut-) AFs and using state-of-the-art AF solvers.
- We perform a thorough experimental analysis showing the effectiveness of the proposed approach. We compare the technique with the solver that participated at the ICCMA'17 competition for the computational task EE-pr (resp. -sst), and show that the technique is on average two orders of magnitude faster than the computation from scratch.

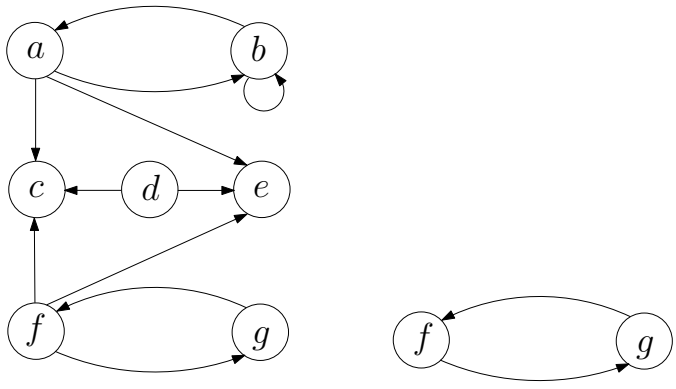
3.2 Enumerating Preferred Extensions

In this section, we provide an approach for efficiently enumerating all the preferred extensions of a given AF. As example consider the following AF.

Example 3.1. The pair $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ where $A_0 = \langle \{a, b, c, d, e, f, g\}$ and $\Sigma_0 = \{(a, b), (b, a), (b, b), (a, c), (a, e), (d, c), (d, e), (f, c), (f, e), (f, g), (g, f)\}\rangle$ is an AF, and the corresponding graph is shown in Figure 3.1(a). Moreover, we have that the grounded extension of $\mathcal{A}_0$ is $E_{gr} = \{d\}$ (i.e., $\mathcal{E}_{\mathtt{gr}}(\mathcal{A}_0) = \{\{d\}\}$), while the ideal extension is $E_{id} = \{a, d\}$ (i.e., $\mathcal{E}_{\mathtt{id}}(\mathcal{A}_0) = \{\{a, d\}\}$). Moreover, the set of preferred extensions is $\mathcal{E}_{\mathtt{pr}}(\mathcal{A}_0) = \{\{a, d, f\}, \{a, d, g\}\}$. □

It is well-known that, for any AF $\mathcal{A}$ and semantics $\mathcal{S} \in \{\mathtt{pr},\mathtt{gr},\mathtt{id}\}$, it is the case that $\mathcal{E}_{\mathcal{S}}(\mathcal{A}) \subseteq \mathcal{E}_{\mathtt{co}}(\mathcal{A})$, and let E_{gr} and E_{id} be the grounded and ideal extensions, for every $E \in \mathcal{E}_{\mathtt{pr}}(\mathcal{A})$, it holds that $E_{gr} \subseteq E_{id} \subseteq E$. Indeed, in the example above, we have that $E_{gr} = \{d\} \subseteq E_{id} = \{a, d\} \subseteq \{a, d, f\}$ and $E_{gr} = \{d\} \subseteq E_{id} = \{a, d\} \subseteq \{a, d, g\}$.

The approach we propose relies on first computing the ideal extension and then using it to define a smaller AF, called *pruned AF*, to be used as the starting point for enumerating the preferred extensions.



(a) (b)

Fig. 3.1: (a) AF $\mathcal{A}_0$, (b) AF *Pruned*$(\mathcal{A}_0)$.

Definition 3.2 (Pruned AF). *Let $\mathcal{A} = \langle A, \Sigma \rangle$ be an AF, and E_{id} the ideal extension for $\mathcal{A}$. The pruned AF for $\mathcal{A}$ is Pruned$(\mathcal{A}) = \langle A_p, \Sigma_p \rangle$ where:*

- $A_p = A \setminus (E_{id} \cup E_{id}^+)$*;*
- $\Sigma_p = \Sigma \setminus \{(a, b) \mid a \in (E_{id} \cup E_{id}^+)$ *or* $b \in (E_{id} \cup E_{id}^+)\}$.

Thus, the pruned AF is obtained by removing from the initial AF all the arguments belonging to the ideal extension as well as the arguments attacked by some argument in the ideal extension. Consistently with this, all the attacks towards or from the arguments removed are deleted as well.

Example 3.3. Continuing with Example 3.1, since $E_{id} = \{a, d\}$, we have that *Pruned*$(\mathcal{A}_0) = \langle A_p, \Sigma_p \rangle$ where:

- $A_p = A_0 \setminus (\{a, d\} \cup \{b, c, e\}) = \{f, g\}$, and
- $\Sigma_p = \Sigma_0 \setminus \{(a, b), (b, a), (b, b), (a, c), (a, e), (d, c), (d, e), (f, c), (f, e)\} = \{(f, g), (g, f)\}$.

The graph corresponding to the pruned AF is shown in Figure 3.1(b).

□

Observe that computing the pruned AF can be accomplished in polynomial time w.r.t. the size (i.e., number of arguments/attacks) of the initial AF.

The following theorem states that every preferred extension E of an AF $\mathcal{A}$ one-to-one corresponds to a preferred extension of the AF *Pruned*$(\mathcal{A})$, and we can obtain a preferred extension of the whole AF by joining a preferred extension of the pruned AF with the ideal extension of $\mathcal{A}$.

Theorem 3.4 (Pruned-AF). *Let $\mathcal{A} = \langle A, \Sigma \rangle$ be an AF, E_{id} the ideal extension for $\mathcal{A}$, and Pruned$(\mathcal{A}) = \langle A_p, \Sigma_p \rangle$ the pruned AF for $\mathcal{A}$. Then, $E \in \mathcal{E}_{pr}(\mathcal{A})$ iff $E = E_{id} \cup E_p$ where $E_p \in \mathcal{E}_{pr}$(Pruned$(\mathcal{A})$).*

Proof. *(Sketch)* The result follows from the facts that (*i*) the ideal extension E_{id} of an AF $\mathcal{A}$ is contained in every preferred extension E for $\mathcal{A}$, that is $E_{id} \subseteq \bigcap_{E \in \mathcal{E}_{\mathtt{pr}}(\mathcal{A})} E$, and (*ii*) removing attacks of the form (a, b) such that $b \in E_{id}$ (for which it holds that $a \notin E_{id}$ and $a \notin E$) preserves any preferred extension E. Let $\mathcal{A}' = \langle A, \Sigma \setminus \{(a, b) \mid b \in E_{id}\}\rangle$. It can be checked that $\mathcal{E}_{\mathtt{pr}}(\mathcal{A}) = \mathcal{E}_{\mathtt{pr}}(\mathcal{A}')$. Indeed, E_{id} is contained in every preferred extension E' for $\mathcal{A}'$, every argument in E^+_{id} does not belong to E', and E^+_{id} does not influence the acceptance status of other arguments. Every preferred extension of $\mathcal{A}'$ consists of E_{id} union a preferred extension of *Pruned*$(\mathcal{A})$, which is the part of $\mathcal{A}'$ (or, equivalently, of $\mathcal{A}$) consisting of the arguments whose acceptance status is not entailed by E_{id}. Finally, the statement follows by observing that $\mathcal{E}_{\mathtt{pr}}(\mathcal{A}) = \mathcal{E}_{\mathtt{pr}}(\mathcal{A}')$. □

Example 3.5. Continuing from Example 3.3, set of preferred extensions of the pruned AF is $\mathcal{E}_{\mathtt{pr}}(\textit{Pruned}(\mathcal{A})) = \{\{f\}, \{g\}\}$. Using the result of Theorem 3.4, we obtain that $\mathcal{E}_{\mathtt{pr}}(\mathcal{A}) = \{\{f\} \cup E_{id}, \{g\} \cup E_{id}\}$, where $E_{id} = \{a, d\}$. Thus, we obtain the preferred extensions $\{a, d, f\}$ and $\{a, d, g\}$. □

It is worth noting that if the ideal extension is empty, then the pruned AF coincides with the whole AF, and the result of Theorem 3.4 becomes trivial. We discuss how to deal with this case in the next section.

3.2.1 Algorithm

The pseudo-code of the algorithm for computing the set of preferred extensions of an AF is shown in Algorithm 1. It takes as input an AF $\mathcal{A}$, and a percentage value k that is a parameter used for deciding when the computation should be carried out by using the pruned AF or not. In fact, in some cases, such as when the ideal extension of the input AF is empty, the overhead of computing the ideal extension of the input AF may not pay off because (*i*) computing the ideal extension is costly [63] and (*ii*) computing the preferred extensions over the pruned AF would cost the same as computing the extension on the initial AF (as discussed earlier, the pruned AF coincides with the whole AF if the ideal extension is empty).

Thus, we use parameter k to decide when computing or not the ideal extension and the pruned AF. In particular, if the grounded extension of the given AF is larger than $k\%$ of the number of arguments in the AF, then the ideal extension (and the pruned AF) is computed; otherwise, the preferred extensions are directly computed on the whole AF from scratch. Here, the grounded extension plays a role for two reasons: first, since the ideal extension is a superset of the grounded extension, the fact that the grounded extension is large enough implies that the ideal extension is large too; second, computing the grounded extension is polynomial-time (while computing the ideal extension is hard), and this suggests that the overhead of computing the grounded extension of the input AF is likely to pay off—in Section 3.2.2 we thoroughly discuss the results of experiments where different values of k are considered, including $k = 0\%$ which means forcing the algorithm to compute the ideal extension and the pruned AF in any case.

Algorithm 1 ScaleEE($\mathcal{A}, k$)

Input: AF $\mathcal{A} = \langle A, \Sigma \rangle$,
A percentage value k.
Output: Set $\mathcal{E}_{\mathrm{pr}}(\mathcal{A})$ of preferred extensions of $\mathcal{A}$.
begin
1: $E_{gr} =$ GR-Solver($\mathcal{A}$)
2: **if** $|E_{gr}| \geq k \cdot |A|$ **then**
3: $E_{id} =$ ID-Solver($\mathcal{A}$)
4: $\mathcal{A}_p = \mathit{Pruned}(\mathcal{A})$
5: $\mathcal{E}_{\mathrm{pr}}(\mathcal{A}_p) =$ PR-Solver($\mathcal{A}_p$)
6: $\mathcal{E}_{\mathrm{pr}}(\mathcal{A}) = \{E \mid E = E_{id} \cup E_p$, where $E_p \in \mathcal{E}_{\mathrm{pr}}(\mathcal{A}_p)\}$
7: **else**
8: $\mathcal{E}_{\mathrm{pr}}(\mathcal{A}) =$ PR-Solver($\mathcal{A}$)
9: **return** $\mathcal{E}_{\mathrm{pr}}(\mathcal{A})$
end.

Algorithm 1 works as follows. It first computes the grounded extension of the given AF $\mathcal{A}$ (Line 1), and then it checks if the size of the grounded extension is bigger than or equal to $k\%$ of the number of the arguments of $\mathcal{A}$ (Line 2). If this holds, the algorithm proceeds by computing the ideal extension of $\mathcal{A}$ (Line 3), which is then used to compute the pruned AF (Line 4). Next, an external AF-solver PR-Solver is called for enumerating the set of extensions of the pruned AF (Line 5), from which the extensions of the whole AF are finally computed at Line 6 using the result of Theorem 3.4. However, if at Line 2 the size of the grounded extension is smaller than $k\%$ of the number of the arguments of $\mathcal{A}$, then the set of extensions of $\mathcal{A}$ is computed from scratch by calling the external solver PR-Solver with input the whole AF (Line 8). Finally, the set of extensions $\mathcal{E}_{\mathrm{pr}}(\mathcal{A})$ computed by using the pruned AF (Lines 3–6) or not (Line 8) is returned.

Example 3.6. Continuing with Example 3.5, if $k = 0\%$ then the condition at Line 2 trivially holds since $|E_{gr}| \geq 0$ for every AF. Therefore, the ideal extension $E_{id} = \{a, d\}$ is computed at Line 3, and the pruned AF $\mathcal{A}_p = \mathit{Pruned}(\mathcal{A}) = \langle \{f, g\}, \{(f, g), (g, f)\} \rangle$ is computed at Line 4. Next, the set of all preferred extensions $\mathcal{E}_{\mathrm{pr}}(\mathcal{A}_p) = \{\{f\}, \{g\}\}$ of the pruned AF is computed (Line 5), and the set of preferred extensions of the whole AF is computed at Line 6 by combining the arguments in the ideal extension with those in the preferred extensions of the pruned AF. Therefore, the output of the algorithm is obtained as follows: $\mathcal{E}_{\mathrm{pr}}(\mathcal{A}) = \{\{\{a, d\} \cup \{f\}\}, \{\{a, d\} \cup \{g\}\}\} = \{\{a, d, f\}, \{a, d, g\}\}$.

Considering now the case that $k = 5\%$, we have again that $|E_{gr}| \geq k \cdot |A|$ (since $1 \geq 0.05 \cdot 7 = 0.35$), and thus the execution of Algorithm 1 is again as above.

Finally, consider the case that $k = 20\%$ for which we have that $|E_{gr}| \not\geq k \cdot |A|$ (since $1 \not\geq 0.2 \cdot 7 = 1.4$). Thus Algorithm 1 directly computes the set $\mathcal{E}_{\mathrm{pr}}(\mathcal{A})$ of preferred extensions by calling the solver PR-Solver with input the whole AF (Line 8). □

	Dataset		
	A1	A2	A3
Number of AFs	23	25	43
Min number of arguments	12	61	40
Max number of arguments	528	1.200	5.700
Min number of attacks	18	97	72
Max number of attacks	3.300	184.000	690.000
Average degree	4	21	22
Average density	0.04	0.05	0.04

Table 3.1: Datasets' properties.

The following theorems states that Algorithm 1 is sound and complete, provided that the external solvers return the correct results.

Theorem 3.7. *Given an AF $\mathcal{A}$, if GR-Solver, ID-Solver, and PR-Solver are sound and complete, then Algorithm 1 computes the set $\mathcal{E}_{\texttt{pr}}(\mathcal{A})$ of preferred extensions of $\mathcal{A}$.*

Proof. *(Sketch)* If $|E_{gr}| < k \cdot |A|$ then the algorithm returns the set of preferred extensions by calling PR-Solver($\mathcal{A}$) (Line 8), which is a sound and complete strategy. Otherwise, $|E_{gr}| \geq k \cdot |A|$ and the algorithm returns the set of preferred extensions by using result of Theorem 3.4, from which the statement follows. □

3.2.2 Implementation and Experiments

We implemented a C++ prototype and compared Algorithm 1 with *ArgSemSAT* [55], the solver that won the ICCMA'17 competition for the task EE-pr which consists in determining all the preferred extensions of a given AF.

Datasets. We used benchmark AFs from the EE-pr track of ICCMA'17. Specifically, we used the AFs in the datasets named $A1$, $A2$, and $A3$ having more than one preferred extension. Table 3.1 reports the number of AFs in each dataset, the range of the number of arguments/attacks in the AFs, as well as the average degree (number of attacks per argument) and average density (number of actual attacks over the number of the attacks in the complete AF).

Methodology. For every AF $\mathcal{A}$ in each dataset, we first computed the set of all the preferred extensions of $\mathcal{A}$ by calling Algorithm 1, where the following external solvers were used:

- GR-Solver: CoQuiAAS [88], the winner of ICCMA'17 track for computing the grounded extension;
- ID-Solver: *pyglaf* [7], the winner of ICCMA'17 track for computing the ideal extension;

	Dataset		
Percentage k	A1	A2	A3
0%	13.43	299	637.28
5%	13.51	286	637.35
10%	13.57	281	572
20%	13.52	205	384

Table 3.2: Average improvement for different values of parameter k over the three datasets.

- PR-Solver: *ArgSemSAT*, the winner of ICCMA'17 track for computing all preferred extensions.

Then, the amount of time required by Algorithm 1 was compared with that required by *ArgSemSAT* to compute all preferred extension over the given AF $\mathcal{A}$ from scratch.

Results. Figure 3.2 reports the average improvement (log scale) obtained by Algorithm 1 over the computation from scratch for the AFs in the datasets $A1$ (left-hand side), $A2$ (middle), and $A3$ (right-hand side), and for $k = 0\%$ (first row), $k = 5\%$ (second row), $k = 10\%$ (third row), and $k = 20\%$ (fourth row).

Specifically, given an AF $\mathcal{A}$ and a percentage value k, we measured the improvement as follows:

$$\textit{improvement}(\mathcal{A}, k) = \frac{\text{running time of } \textit{ArgSemSAT} \text{ with input } \mathcal{A}}{\text{running time of } ScaleEE(\mathcal{A}, k)} \tag{3.1}$$

In Figure 3.2, triangular-shaped data points (coloured green) correspond to AFs having a grounded extension larger than or equal to $k\%$ of the number of the arguments. In these cases, the ideal extension and then the pruned AF is computed by executing Lines 3–6 of Algorithm 1. Squared-shaped data points (coloured red) represent AFs having a grounded extension smaller than $k\%$ of the number of the arguments, and thus the pruned AF is *not* computed as Line 8 of Algorithm 1 is executed.

For each diagram in Figure 3.2, a solid black line representing the average improvement obtained for the considered dataset and value of k is reported. Moreover, to easy readability, we also report a dashed grey line corresponding to average improvement equal to 1. Clearly, an improvement strict less than 1 means that the overall overhead of computing the grounded extension, and eventually the ideal extension, does not pay off. However, when the improvement is close to 1, the overhead is negligible.

From the results in Figure 3.2, we can draw the following conclusions:

- Algorithm 1 significantly outperforms the competitor that computes the preferred extensions from scratch. In fact, the average improvement (see the solid black line in the diagrams in Figure 3.2 and the detailed values in Table 3.2) is greater than 10, 200, and 380 over the datasets $A1$, $A2$, and $A3$, respectively, mean-

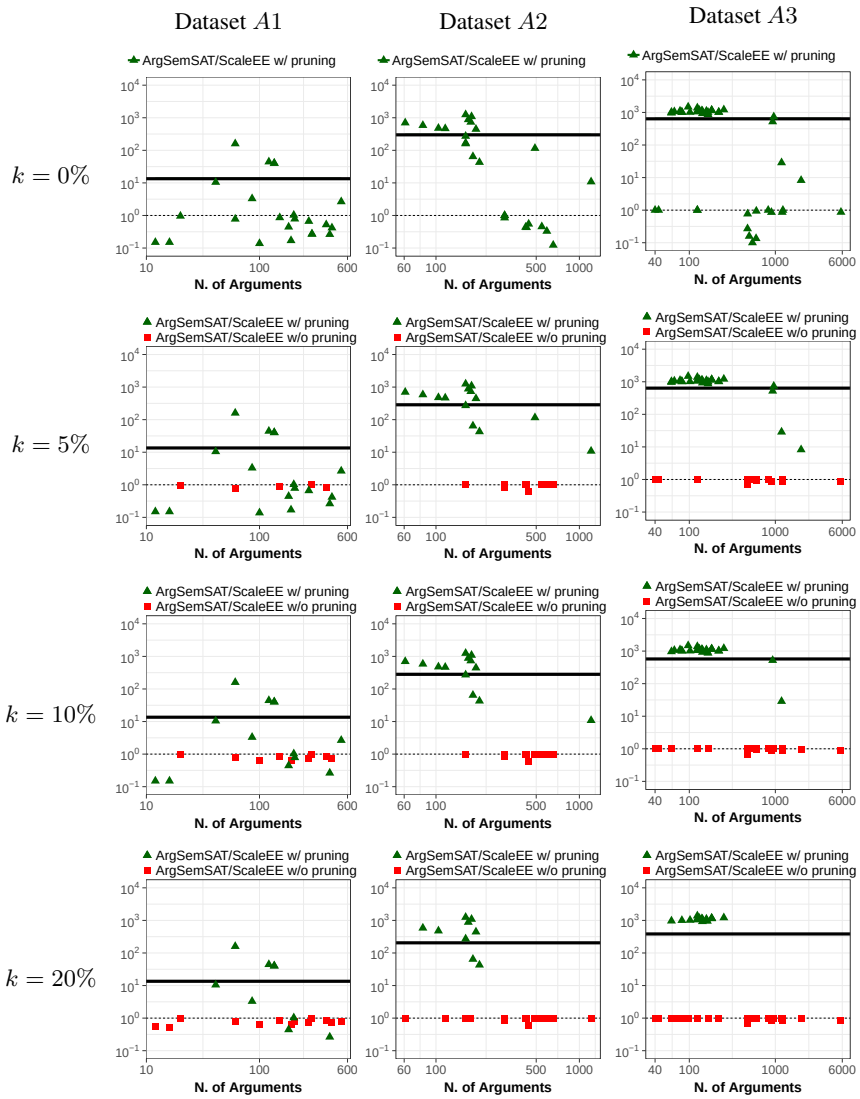


Fig. 3.2: Improvement (i.e. the running time of *ArgSemSAT* over the running time of Algorithm 1) for $k = 0\%$ (first row), $k = 5\%$ (second row), $k = 10\%$ (third row), and $k = 20\%$ (fourth row), over the datasets $A1$ (left-hand side), $A2$ (middle), and $A3$ (right-hand side). Triangular-shaped data points (coloured green) represent AFs having a grounded extension larger than or equal to $k\%$ of the number of the arguments, and thus the pruned AF is computed by executing Lines 3–6 of Algorithm 1. Squared-shaped data points (coloured red) represent AFs having a grounded extension smaller than $k\%$ of the number of arguments, and thus the pruned AF is *not* computed (Line 8 of Algorithm 1 is executed).

ing that Algorithm 1 is on average at least 10, 200, and 380 times faster than *ArgSemSAT*.

- The larger the average degree of the AFs (see Table 3.1), the bigger the (average) improvement obtained. In particular, for the datasets $A2$ and $A3$, this implies that the amount of time required decreases from dozens of minutes (computation from scratch) to a few seconds (Algorithm 1).
- The average improvement remains high for $k = 0\%$, that is, when computing both the ideal extension and the pruned AF irrespectively of the size of the grounded extension. However, the number of AFs for which the improvement is too lower than 1 decreases if $k > 0\%$. In particular, for the datasets $A2$ and $A3$, using $k = 5\%$ is enough for avoiding all the cases for which the improvement is significantly lower than 1, while using $k = 10\%$ avoids many undesirable cases for the datasets $A1$. Thus, using k greater than zero allows us to reduce the overhead due to the computation of the ideal extension and the pruned AF.
- Although increasing the value of k avoids cases where the proposed approach may work worse than the computation from scratch, using too high values of k deteriorates performances on average because the pruned AF is not built even when it would be helpful. In fact, for the datasets $A2$ and $A3$, using $k = 10\%$ (or $k = 20\%$) entails that the pruned AF is not built in vain for the AFs whose improvements are shown as green data points in Figure 3.2 for $k = 5\%$ and become coloured red when passing to $k = 10\%$ (since increasing k entails that pruned AF is no longer computed). A similar behavior can be observed in Figure 3.2 when increasing k from 10% to 20% for the dataset $A1$.
- All in all, the best trade-off between paying the cost of computing the ideal extension along with the pruned AF and risking to have the overhead of the computation of the ideal extension seems to be choosing k greater than zero but no more than 10%.

3.3 Enumerating Semi-Stable Extensions

In this section, we tackle the problem of efficiently enumerating all the semi-stable extensions of a given AF. Analogously to what done for the case of the preferred semantics, where ideal extension was exploited to build the Pruned AF, the approach for semi-stable semantics relies on first computing the grounded extension and then using it to define a smaller AF, called *cut-AF*, to be used as the starting point for enumerating the semi-stable extensions. As running example of this section, we will refer to the following AF.

Example 3.8. The pair $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ where $A_0 = \langle \{a, b, c, d,\ e,\ f, g, h\}$ and $\Sigma_0 = \{(a, b),\ (b, c),\ (c, d),\ (d, a),\ (f, e),\ (g, h),\ (e, a),\ (h, a)\}\rangle$ is an AF, and the corresponding graph is shown in Figure 3.3(a). □

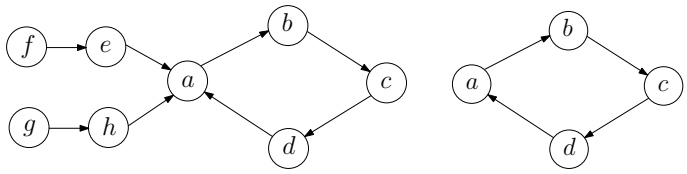


Fig. 3.3: (a) AF $\mathcal{A}_0$, (b) AF *Cut*($\mathcal{A}_0$).

Example 3.9. Continuing with Example 3.8, we have that the grounded extension of $\mathcal{A}_0$ is $E_{gr} = \{f, g\}$ (i.e., $\mathcal{E}_{\texttt{gr}}(\mathcal{A}_0) = \{\{f, g\}\}$). Moreover, the set of semi-stable extensions is $\mathcal{E}_{\texttt{sst}}(\mathcal{A}_0) = \{\{a, c, f, g\}, \{b, d, f, g\}\}$. □

It is well-known that, for any AF $\mathcal{A}$ and semantics $\mathcal{S} \in \{\texttt{gr},\texttt{sst}\}$, it is the case that $\mathcal{E}_{\mathcal{S}}(\mathcal{A}) \subseteq \mathcal{E}_{\texttt{co}}(\mathcal{A})$, and let E_{gr} and E_{sst} be the grounded and semi-stable extensions, for every $E \in \mathcal{E}_{\texttt{sst}}(\mathcal{A})$, it holds that $E_{gr} \subseteq E$. Indeed, in the example above, we have that $E_{gr} = \{f, g\} \subseteq E_{sst} = \{a, c, f, g\} \subseteq E_{co} = \{a, c, f, g\}$ and $E_{gr} = \{f, g\} \subseteq E_{sst} = \{b, d, f, g\} \subseteq E_{co} = \{b, d, f, g\}$

Definition 3.10. *Let $\mathcal{A} = \langle A, \Sigma\rangle$ be an AF, and E_{gr} the grounded extension for $\mathcal{A}$. The cut-AF for $\mathcal{A}$ is Cut($\mathcal{A}$) $= \langle A_{cut}, \Sigma_{cut}\rangle$ where:*

- $A_{cut} = A \setminus (E_{gr} \cup E^{+}_{gr})$;
- $\Sigma_{cut} = \Sigma \setminus \{(a, b) \mid a \in (E_{gr} \cup E^{+}_{gr})$ *or* $b \in (E_{gr} \cup E^{+}_{gr})\}$.

Thus, the cut-AF is obtained by removing from the initial AF all the arguments belonging to the grounded extension as well as the arguments attacked by some argument in the grounded extension. Consistently with this, all the attacks towards or from the arguments removed are deleted as well.

Example 3.11. Continuing with Example 3.9, since $E_{gr} = \{f, g\}$, we have that *Cut*($\mathcal{A}_0$) $= \langle A_{cut}, \Sigma_{cut}\rangle$ where:

- $A_{cut} = A_0 \setminus (\{f, g\} \cup \{h, e\}) = \{a, b, c, d\}$, and
- $\Sigma_{cut} = \Sigma_0 \setminus \{(f, e), (e, a), (g, h), (h, a), \} = \{(a, b), (b, c), (c, d), (d, a)\ \}$.

The graph corresponding to the cut-AF is shown in Figure 3.3(b). □

Observe that computing the cut-AF can be accomplished in polynomial time w.r.t. the size (i.e., number of arguments/attacks) of the initial AF.

The following theorem states that every semi-stable extension E of an AF $\mathcal{A}$ one-to-one corresponds to a semi-stable extension of the AF *Cut*($\mathcal{A}$), and we can obtain a semi-stable extension of the whole AF by joining a semi-stable extension of the cut-AF with the grounded extension of $\mathcal{A}$.

Theorem 3.12. *Let $\mathcal{A} = \langle A, \Sigma\rangle$ be an AF, E_{gr} the grounded extension for $\mathcal{A}$, and Cut($\mathcal{A}$) $= \langle A_{cut}, \Sigma_{cut}\rangle$ the cut-AF for $\mathcal{A}$. Then, $E \in \mathcal{E}_{sst}(\mathcal{A})$ iff $E = E_{gr} \cup E_{cut}$ where $E_{cut} \in \mathcal{E}_{sst}$(Cut($\mathcal{A}$)).*

Example 3.13. Continuing from Example 3.11, the set of semi-stable extensions of the cut-AF is $\mathcal{E}_{\texttt{sst}}(\textit{Cut}(\mathcal{A})) = \{\{a, c\}, \{b, d\}\}$. Using the result of Theorem 3.12, we obtain that $\mathcal{E}_{\texttt{sst}}(\mathcal{A}) = \{\{a, c\} \cup E_{gr}, \{b, d\} \cup E_{gr}\}$, where $E_{gr} = \{f, g\}$. Thus, we obtain the semi-stable extensions $\{f, g, a, c\}$ and $\{f, g, b, d\}$ (c.f. Example 3.9). □

3.3.1 Algorithm

The pseudo-code of the algorithm for computing the set of semi-stable extensions of an AF is shown in Algorithm 2.

Similarly to the case of Algorithm 1, Algorithm 2 takes as input an AF $\mathcal{A}$, and a percentage value k that is a parameter used for deciding when the computation should be carried out by using the cut-AF or not. In fact, in some cases, such as when the grounded extension of the input AF is empty, the overhead of computing the cut-AF does not pay off because it will correspond to be the whole initial framework, and so, computing the semi-stable extensions over the cut-AF would cost the same as computing the extension on the initial AF plus the overhead of computing the cut-AF.

Thus, we use parameter k to decide when computing or not the cut-AF. In particular, if the grounded extension of the given AF is larger than $k\%$ of the number of arguments in the AF, then the cut-AF is computed; otherwise, the semi-stable extensions are directly computed w.r.t the whole AF from scratch. Here, computing the grounded extension is polynomial-time (while computing the semi-stable extension is hard), and this suggests that the overhead of computing the grounded extension of the input AF is likely to pay off—in Section 3.3.2 we discuss the results of experiments where different values of k are considered, including $k = 0\%$ which means forcing the algorithm to compute the cut-AF in any case.

The main difference between Alorithm 1 and Algorithm 2 is that for the case of semi-stable semantics we do not exploit the concept of ideal and grounded semantics as for the case of Algorithm 1. We have seen that for the preferred semantics, computing the ideal extension pays off in many cases. However, as for the semi-stable semantics the number of extensions to be enumerated is generally smaller than that for the preferred semantics, cutting the arguments contained in the grounded extension is enough to obtain good results (computing the ideal extension, which is costly, may not pay off if it is then used to compute only a few semi-stable extensions).

Example 3.14. Continuing with Example 3.13, if $k = 0\%$ then the condition at Line 2 trivially holds since $|E_{gr}| \geq 0$ for every AF. Therefore, the cut-AF $\mathcal{A}_{cut} = \textit{Cut}(\mathcal{A}) = \langle\{a, b, c, d\}, \{(a, b), (b, c), (c, d), (d, a)\}\rangle$ is computed at Line 3. Next, the set of all semi-stable extensions $\mathcal{E}_{\texttt{sst}}(\mathcal{A}_{cut}) = \{\{a, c\}, \{b, d\}\}$ of the cut-AF is computed (Line 4), and the set of semi-stable extensions of the whole AF is computed at Line 5 by combining the arguments in the grounded extension with those in the semi-stable extensions of the cut-AF. Therefore, the output of the algorithm is obtained as follows: $\mathcal{E}_{\texttt{sst}}(\mathcal{A}) = \{\{\{f, g\} \cup \{a, c\}\}, \{\{f, g\} \cup \{b, d\}\}\} = \{\{f, g, a, c\}, \{f, g, b, d\}\}$.

Algorithm 2 CutSST($\mathcal{A}, k$)

Input: AF $\mathcal{A} = \langle A, \Sigma \rangle$,
A percentage value p.
Output: Set $\mathcal{E}_{\texttt{sst}}(\mathcal{A})$ of semi-stable extensions of $\mathcal{A}$.
begin
1: $E_{gr} =$ GR-Solver($\mathcal{A}$)
2: **if** $|E_{gr}| \geq p \cdot |A|$ **then**
3: $\mathcal{A}_{cut} = Cut(\mathcal{A})$
4: $\mathcal{E}_{\texttt{sst}}(\mathcal{A}_{cut}) =$ SST-Solver($\mathcal{A}_{cut}$)
5: $\mathcal{E}_{\texttt{sst}}(\mathcal{A}) = \{E \mid E = E_{gr} \cup E_{cut}$, where $E_{cut} \in \mathcal{E}_{\texttt{sst}}(\mathcal{A}_{cut})\}$
6: **else**
7: $\mathcal{E}_{\texttt{sst}}(\mathcal{A}) =$ SST-Solver($\mathcal{A}$)
8: **return** $\mathcal{E}_{\texttt{sst}}(\mathcal{A})$

Considering now the case that $k = 5\%$, we have again that $|E_{gr}| \geq k \cdot |A|$ (since $2 \geq 0.05 \cdot 8 = 0.4$), and thus the execution of Algorithm 2 is again as above.

Finally, consider the case that $k = 30\%$ for which we have that $|E_{gr}| \not\geq k \cdot |A|$ (since $2 \not\geq 0.3 \cdot 8 = 2.4$). Thus Algorithm 2 directly computes the set $\mathcal{E}_{\texttt{sst}}(\mathcal{A})$ of semi-stable extensions by calling the solver SST-Solver with input the whole AF (Line 7). □

The following theorems states that Algorithm 2 is sound and complete, provided that the external solvers return the correct results.

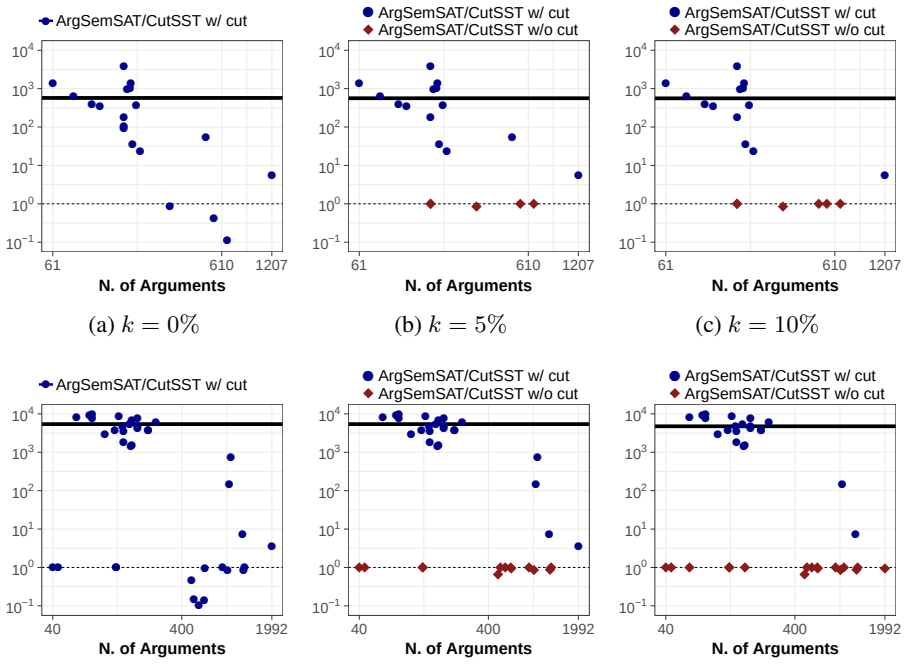
Theorem 3.15. *Given an AF $\mathcal{A}$, if GR-Solver and SST-Solver are sound and complete, then Algorithm 2 returns the set $\mathcal{E}_{sst}(\mathcal{A})$ of semi-stable extensions of $\mathcal{A}$.*

3.3.2 Implementation and Experiments

We implemented a C++ prototype and tested the technique using the same methodology and external solvers for the case of Algorithm 1 over benchmark AFs taken from the EE-`sst` track of ICCMA17, which consists in determining all the semi-stable extensions of a given AF. Specifically, we used the AFs in the datasets named $E2$ and $E3$ having more than one semi-stable extension. Particularly, dataset $E2$ (resp. $E3$) consists of 19 (resp. 41) AFs, and a number of arguments contained in AFs of dataset $E2$ (resp. $E3$) that varies from a minimum value of 61 (resp. 40) to a maximum of $1.2K$ (resp. $1.9K$). Furthermore, the range of the number of attacks in the AFs of dataset $E2$ (resp. $E3$) varies from a minimum of 97 (resp. 72) to a maximum of $10.3K$ (resp. $218K$).

Here we directly report and discuss experimental results.

Figure 3.4 reports (in log scale) the average improvement (obtained by substituting ScaleEE with CutSST in equation (3.1)) of Algorithm 2 over the computation from scratch for the AFs in the datasets $E2$ (first row), and $E3$ (second row), and for $k = 0\%$ (first column), $k = 5\%$ (second column), $k = 10\%$ (third column).



(a) $k = 0\%$ (b) $k = 5\%$ (c) $k = 10\%$

(d) $k = 0\%$ (e) $k = 5\%$ (f) $k = 10\%$

Fig. 3.4: Improvement (i.e. the running time of *ArgSemSAT* over the running time of Algorithm 2) for $k = 0\%$ (Fig. 3.4a and Fig. 3.4d), $k = 5\%$ (Fig. 3.4b and Fig. 3.4e), and $k = 10\%$ (Fig. 3.4c and Fig. 3.4f), over the datasets $E2$ (first row) and $E3$ (second row). Circle-shaped data points (coloured blue) represent AFs having a grounded extension larger than or equal to $k\%$ of the number of the arguments, and thus the cut-AF is computed by executing Lines 3–5 of Algorithm 2. Diamond-shaped data points (coloured red) represent AFs having a grounded extension smaller than $k\%$ of the number of arguments, and thus the cut-AF is *not* computed (Line 7 of Algorithm 2 is executed).

In Figure 3.4, circle-shaped data points (colored blue) correspond to AFs having a grounded extension larger than or equal to $k\%$ of the number of the arguments. In these cases, the cut-AF is computed by executing Lines 3–5 of Algorithm 2. Diamond-shaped data points (colored red) represent AFs having a grounded extension smaller than $k\%$ of the number of the arguments, and in this case, cut-AF is *not* computed and Line 7 of Algorithm 2 is executed.

For each plot in Figure 3.4, a solid black line representing the average improvement obtained for the considered dataset and value of k is reported. Moreover, to easy readability, we also report a dashed grey line corresponding to average improvement equal to 1. Clearly, an improvement strict less than 1 means that the overall overhead

of computing the grounded extension, and eventually the cut-AF, does not pay off. However, when the improvement is close to 1, the overhead is negligible.

From the results in Figure 3.4, we can draw the following conclusions:

- Algorithm 2 significantly outperforms the competitor that computes the semi-stable extensions from scratch. In fact, the average improvement is greater than 410 and 2100 over the datasets $E2$ and $E3$, respectively, meaning that Algorithm 2 is on average at least 410 and 2100 times faster than *ArgSemSAT*.
- The smaller the number of arguments of the AFs, the bigger the (average) improvement obtained. In particular, for the datasets $E2$ and $E3$, this implies that the amount of time required decreases from dozens of minutes (computation from scratch) to a few seconds (Algorithm 2).
- The average improvement remains high for $k = 0\%$, that is, when computing the grounded extension and the cut-AF irrespectively of the size of the grounded extension. However, the number of AFs for which the improvement is too lower than 1 decreases if $k > 0\%$. In particular, for the datasets $E2$ and $E3$, using $k = 5\%$ is enough for avoiding all the cases for which the improvement is significantly lower than 1. Thus, using k greater than zero allows us to reduce the overhead due to the computation of the grounded extension plus the cut-AF.
- Although increasing the value of k avoids cases where the proposed approach may work worse than the computation from scratch, using too high values of k deteriorates performances on average because the cut-AF is not built even when it would be helpful. In fact, for the datasets $E2$ and $E3$, using $k = 10\%$ entails that the cut-AF is not built in vain for the AFs whose improvements are shown as blue data points in Figure 3.4 for $k = 5\%$ and become colored red when passing to $k = 10\%$ (since increasing k entails that cut-AF is no longer computed).
- All in all, the best trade-off between paying the cost of computing the grounded extension along with the cut-AF and risking to have the overhead of the computation of the cut-AF seems to be choosing k greater than zero but no more than 10%.

3.4 Summary

Several computational problems of AFs have been studied such as skeptical and credulous reasoning, existence of a non-empty extension, and verifying if a set of arguments is an extension under different argumentation semantics [63, 65, 67, 68]. The complexity of the problem of computing all extensions according to some semantics for AFs has been recently investigated in [87], where it was shown that the enumeration problem is intractable under the preferred and semi-stable semantics, and, in particular, they are not in *OutputP* ("output-polynomial time", also known as *TotalP* "total polynomial time" [84]) even for bipartite AFs. An approach for dividing the problem of enumerating the preferred extensions into sub-problems is proposed in [56], where a meta-algorithm based on SCC-recursiveness [18] is introduced. However, this kind of approaches provide advantages only if there are many

strongly connected components, or in case of sparse AFs (i.e., average degree less than 2) [92].

This is not case of the proposed technique which seems not taking advantages neither from low average degree (the improvement obtained for preferred semantics over the datasets $A2$ and $A3$ having average degree greater than 20 is even better than that obtained over $A1$ having average degree equal to 4), nor from the number of strongly connected components (we checked that there is no correlation between the improvements obtained and the number of SCCs in the AFs).

An approach to deal with the problem of enumerating the semi-stable extensions is proposed in [35], where a new algorithm for computing semi-stable semantics using dynamic programming on tree decompositions that runs in linear time on AFs of bounded treewidth is presented. However, this kind of approaches provide advantages only in case of bounded treewidth.

To conclude, we introduced a technique for efficiently enumerating the set of preferred and semi-stable extensions of abstract argumentation frameworks. It is modular with respect to the solvers used for computing the grounded (resp. ideal) extensions, as well as the solver used for the enumeration of semi-stable (resp. preferred) extensions on the cut-AF (pruned-AF)—any solver addressing one of these tasks could be plugged-in and exploited for addressing the enumeration problem under the semi-stable (resp. preferred) semantics.

We have experimentally investigated the behavior of the technique, and analyzed the conditions under which building the pruned-AF (resp. cut-AF) and computing the ideal (resp. grounded) extension are convenient for computing the set of preferred (resp. semi-stable). It turned out that it is worth paying the cost of building the cut-AF after looking at the size of the grounded extension as the computation of the semi-stable extensions over the cut-AF yields significant improvements over the computation from scratch. Analogously, for the preferred semantics, it turned out that it is worth paying the cost of computing the ideal extension if is not empty—this can be easily checked by looking at the size of the grounded extension—as the computation of the preferred extensions over the pruned AF yields significant improvements over the computation from scratch.

Future work will be devoted to extending the proposed technique to the enumeration problem in the presence of other argumentation semantics, such as the *stable* semantics [61]. In fact, a stable extension is a complete extension which attacks all the arguments outside the extension, and the set of stable extensions are a subset of the set of preferred and semi-stable extensions; thus, similarly to the preferred and semi-stable semantics, the grounded and ideal extension are contained in every stable extension. For instance, in the proposed examples the set of stable extensions coincides with that of the preferred and semi-stable extensions considering the pruned-AF (resp. cut-AF) and the whole initial one. However, extending the technique to deal with the stable semantics requires to face up with the fact that a stable extension may not exists for an AF, and checking this is computationally hard [65].

4

Efficient Computation of Extensions in Dynamic Argumentation Frameworks

"Logic, it is often said, is the study of valid arguments. It is a systematic attempt to distinguish valid arguments from invalid arguments."

– William H. Newton-Smith

Abstract argumentation frameworks (AFs) are a well-known formalism for modelling and deciding many argumentation problems. Computational issues and evaluation algorithms have been deeply investigated for static AFs, whose structure does not change over the time. However, AFs are often dynamic as a consequence of the fact that argumentation is inherently dynamic.

In this chapter, we tackle the problem of incrementally computing extensions for dynamic AFs: given an initial extension and an update (or a set of updates), we devise a technique for computing an extension of the updated AF under four well-known semantics (i.e., *complete*, *preferred*, *stable*, and *grounded*). The idea is to identify a reduced (updated) AF sufficient to compute an extension of the whole AF and use state-of-the-art algorithms to recompute an extension of the reduced AF only.

The experiments reveal that, for all semantics considered and using different solvers, the incremental technique is on average two orders of magnitude faster than computing the semantics from scratch.

4.1 Arguments Influenced by an Update

In this chapter, as running example, we will refer to the AF $\mathcal{A}_0$ presented in Chapter 2 and reported on Figure 2.1, whose set of admissible sets is $\{\emptyset, \{b\}, \{g\}, \{a, g\}, \{b, g\}, \{f, g\}, \{a, g, f\}, \{b, g, f\}\}$, and $\mathcal{E}_{\mathcal{S}}(\mathcal{A}_0)$ with $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{gr}\}$ is as reported in the second column of Table 4.1.

Applying an update u to an AF implies that its semantics (set of extensions or labellings) changes, as shown by Table 4.1 which reports the sets of extensions for the AFs of Figure 4.1 before and after performing the update $+(c, f)$. The updated AF $\mathcal{A} = +(c, f)(\mathcal{A}_0)$ is shown on Figure 4.1

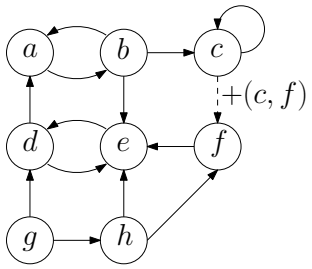


Fig. 4.1: AFs $\mathcal{A}_0$ and $\mathcal{A} = +(c, f)(\mathcal{A}_0)$

$\mathcal{S}$	$\mathcal{E}_{\mathcal{S}}(\mathcal{A}_0)$	$\mathcal{E}_{\mathcal{S}}(\mathcal{A}))$
co	$\{\{f, g\}, \{a, f, g\}, \{b, f, g\}\}$	$\{\{g\}, \{a, g\}, \{b, f, g\}\}$
pr	$\{\{a, f, g\}, \{b, f, g\}\}$	$\{\{a, g\}, \{b, f, g\}\}$
st	$\{\{b, f, g\}\}$	$\{\{b, f, g\}\}$
gr	$\{\{f, g\}\}$	$\{\{g\}\}$

Table 4.1: Sets of extensions for $\mathcal{A}_0$ and $\mathcal{A} = +(c, f)(\mathcal{A}_0)$.

In this section, we first provide some sufficient conditions ensuring that a given $\mathcal{S}$-extension for an AF $\mathcal{A}$ continues to be an $\mathcal{S}$-extension for the updated AF $u(\mathcal{A})$. Then, we introduce the *influenced set* which intuitively consists of the set of arguments whose status may change after performing an update.

Updates preserving a given initial extension. Given an update $\pm(a, b)$ and an initial extension E_0 corresponding to L_0, for each pair of initial statuses $L_0(a)$ and $L_0(b)$ of the arguments involved in the update, Tables 4.2 and 4.3 tell us the semantics for which E_0 is still an extension after the update.

Proposition 4.1. *Let $\mathcal{A}_0$ be an AF, $\mathcal{S}$ a semantics, $E_0 \in \mathcal{E}_{\mathcal{S}}(\mathcal{A}_0)$ an extension of $\mathcal{A}_0$ under semantics $\mathcal{S}$, L_0 the labelling corresponding to E_0, and u an update. If $\mathcal{S}$ is in the cell $\langle L_0(a), L_0(b)\rangle$ of Table 4.2 and $u = +(a, b)$ (resp., of Table 4.3 and $u = -(a, b)$), then $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{A}_0))$.*

The results in Tables 4.2 and 4.3 concerning gr follow from those in [37, 38], where the principles according to which the grounded extension does not change when attacks are added/removed have been studied.

In the following, given an AF $\mathcal{A}_0$ and an $\mathcal{S}$-extension E_0 for it, we say that an update u is *irrelevant* w.r.t. E_0 and $\mathcal{S}$ iff the conditions of Proposition 4.1 hold. Otherwise, u is *relevant*.

Example 4.2. Consider $\mathcal{A}_0$ of Figure 4.1 and its sets of extensions listed in the second column of Table 4.1. $E_0 = \{b, f, g\}$ is an extension according to semantics $\mathcal{S} \in \{$co,pr,st$\}$. Thus, $L_0(c) =$ OUT and $L_0(f) =$ IN, and using Proposition 4.1 it follows that for update $u = +(c, f)$ E_0 is still an extension of $u(\mathcal{A}_0)$ (see the last column of Table 4.1). Thus $+(c, f)$ is irrelevant w.r.t. E_0 and $\mathcal{S}$. However, $+(c, f)$ is relevant w.r.t. $E_0 = \{a, f, g\}$ and all the semantics (in this case $L_0(c) =$ UNDECIDED

update $+(a,b)$		$L_0(b)$ = IN	$L_0(b)$ = UNDECIDED	$L_0(b)$ = OUT
$L_0(a)$	IN			co, pr, st, gr
	UNDECIDED		co, gr	co, pr, gr
	OUT	co, pr, st	co, gr	co, pr, st, gr

Table 4.2: Cases for which $E_0 \in \mathcal{E}_\mathcal{S}(u(\mathcal{A}_0))$ for $u = +(a,b)$.

update $-(a,b)$		$L_0(b)$ = IN	$L_0(b)$ = UNDECIDED	$L_0(b)$ = OUT
$L_0(a)$	IN	NA	NA	
	UNDECIDED	NA		co, pr, gr
	OUT	co, pr, st, gr	co, pr, gr	co, pr, st, gr

Table 4.3: Cases for which $E_0 \in \mathcal{E}_\mathcal{S}(u(\mathcal{A}_0))$ for $u = -(a,b)$.

and $L_0(f) = \text{IN}$, and no semantics is listed in the cell $\langle \text{UNDECIDED}, \text{IN} \rangle$ of Table 4.2). □

It is important to note that Tables 4.2 and 4.3 are not meant to be exhaustive, as more conditions can be found for which an $\mathcal{S}$-extension is preserved after an update. For instance, for the grounded semantics, the initial extension is preserved also if $L_0(a) = \text{OUT}$ and $L_0(b) = \text{IN}$ and argument a of updated $+(a,b)$ is not reachable from b. Here we provided a simple set of conditions that can be easily checked by just looking at the initial labelling L_0. The proposed technique can be trivially extended by considering a more general set of such conditions.

Influenced set. For irrelevant updates, the influenced set will be empty (in this case, the initial extension will be immediately returned as an extension of the updated AF by Algorithm 3). If none of the conditions of Proposition 4.1 holds (i.e., the update is relevant), then the influenced set may turn out to be not empty. In such case, the influenced set will be used to delineate a portion of the argumentation framework, called *reduced AF*, that we will use to recompute (a portion of) an extension for the updated AF.

Given an AF $\mathcal{A} = \langle A, \Sigma \rangle$ and an argument $b \in A$, we use $Reach_\mathcal{A}(b)$ to denote the set of arguments that are reachable from b in the graph $\mathcal{A}$.

Definition 4.3 (Influenced set). Let $\mathcal{A} = \langle A, \Sigma \rangle$ be an AF, $u = \pm(a,b)$, E an extension of $\mathcal{A}$ under semantics $\mathcal{S}$, and let

$$- \mathcal{I}_0(u, \mathcal{A}, E) = \begin{cases} \emptyset \ \textit{if u is irrelevant w.r.t. E and } \mathcal{S} \textit{ or} \\ \quad \exists (z,b) \in \Sigma \ s.t. \ z \neq a \wedge z \in E \wedge \\ \quad z \notin \textit{Reach}_\mathcal{A}(b); \\ \{b\} \ otherwise; \end{cases}$$

$$- \mathcal{I}_{i+1}(u, \mathcal{A}, E) = \mathcal{I}_i(u, \mathcal{A}, E) \cup \{y \mid \exists (x,y) \in \Sigma \ s.t. \ x \in \mathcal{I}_i(u, \mathcal{A}, E) \wedge \nexists (z,y) \in \Sigma \ s.t. \ z \in E \wedge z \notin \textit{Reach}_\mathcal{A}(b)\}.$$

The *influenced set* of u w.r.t. $\mathcal{A}$ and E is $\mathcal{I}(u, \mathcal{A}, E) = \mathcal{I}_n(u, \mathcal{A}, E)$ such that $\mathcal{I}_n(u, \mathcal{A}, E) = \mathcal{I}_{n+1}(u, \mathcal{A}, E)$. □

Example 4.4. Consider the the AF $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ of Figure 4.1 and the update $u = +(c, f)$. We have that $Reach_{\mathcal{A}_0}(f) = A_0 \setminus \{g, h\}$. The influenced set depends on the initial extension chosen. For the extension $\{b, f, g\}$ of Example 4.2, we have that the influenced set is empty as u is irrelevant. For the extension $E_0 = \{a, f, g\}$, the influenced set is $\mathcal{I}(u, \mathcal{A}_0, E_0) = \{f, e\}$. Indeed, $d \notin \mathcal{I}(u, \mathcal{A}_0, E_0)$ since it is attacked by $g \in E_0$ which is not reachable from f. Thus the arguments that can be reached from d do not belong to $\mathcal{I}(u, \mathcal{A}_0, E_0)$. If we consider the initial grounded extension $\{f, g\}$, then $\{f, e\}$ turns out to be the influenced set again. □

4.2 Incremental Computation of Extensions

Given the influenced set, we define a subgraph, called *reduced AF*, that will be used to compute the status of the influenced arguments, thus providing an extension that will be combined with that of initial AF to obtain an extension of the updated AF, for every semantics $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{gr}\}$.

For any AF $\mathcal{A} = \langle A, \Sigma \rangle$ and set $S \subseteq A$ of arguments, we denote with $\Pi(S, \mathcal{A}) = \langle S, \Sigma \cap S \times S \rangle$ the subgraph of $\mathcal{A}$ induced by the nodes in S. Moreover, given two AFs $\mathcal{A}_1 = \langle A_1, \Sigma_1 \rangle$ and $\mathcal{A}_2 = \langle A_2, \Sigma_2 \rangle$, we denote as $\mathcal{A}_1 \sqcup \mathcal{A}_2 = \langle A_1 \cup A_1, \Sigma_1 \cup \Sigma_2 \rangle$ the *union* of the two AFs.

Definition 4.5 (Reduced AF). Let $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ be an AF, $E_0 \in \mathcal{E}_{\mathcal{S}}(\mathcal{A}_0)$ an extension for $\mathcal{A}_0$ under a semantics $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{gr}\}$, and $u = \pm(a, b)$ an update. Let $u(\mathcal{A}_0) = \langle A, \Sigma \rangle$ be the AF updated using u. The reduced AF for $\mathcal{A}_0$ w.r.t. E_0 and u (denoted as $\mathcal{R}(u, \mathcal{A}_0, E_0)$) is as follows.

- $\mathcal{R}(u, \mathcal{A}_0, E_0)$ is empty if $\mathcal{I}(u, \mathcal{A}_0, E_0)$ is empty.
- $\mathcal{R}(u, \mathcal{A}_0, E_0) = \Pi(\mathcal{I}(u, \mathcal{A}_0, E_0), u(\mathcal{A}_0)) \sqcup \mathcal{A}_1 \sqcup \mathcal{A}_2$ where:
  - i) $\mathcal{A}_1$ is the union of the AFs $\langle \{a, b\}, \{(a, b)\} \rangle$ s.t. $(a, b) \in \Sigma$, $a \notin \mathcal{I}(u, \mathcal{A}_0, E_0)$, $a \in E_0$, and $b \in \mathcal{I}(u, \mathcal{A}_0, E_0)$;
  - ii) $\mathcal{A}_2$ is the union of the AFs $\langle \{c\}, \{(c, c)\} \rangle$ s.t. there is $(e, c) \in \Sigma$, $e \notin \mathcal{I}(u, \mathcal{A}_0, E_0)$, $e \notin (E_0 \cup E_0^+)$, and $c \in \mathcal{I}(u, \mathcal{A}_0, E_0)$.

Hence, AF $\mathcal{R}(u, \mathcal{A}_0, E_0)$ contains, in addition to the subgraph of $u(\mathcal{A}_0)$ induced by $\mathcal{I}(u, \mathcal{A}_0, E_0)$, additional nodes and edges containing needed information on the "external context", i.e. information about the status of arguments which are attacking some argument in $\mathcal{I}(u, \mathcal{A}_0, E_0)$. Using fake arguments/attacks to represent external contexts has been exploited in [13] where decomposability properties of argumentation semantics are investigated.

Example 4.6. For our running example, if $E_0 = \{a, f, g\}$ and $u = +(c, f)$, the reduced AF $\mathcal{R}(+(c, f), \mathcal{A}_0, E_0)$ consists of the subgraph induced by $\mathcal{I}(u, \mathcal{A}_0, E_0) = \{f, e\}$ plus the edge (f, f) as there is the attack (c, f) in the updated AF from an uninfluenced argument c labelled as UNDECIDED toward the influenced argument f. Hence, $\mathcal{R}(+(c, f), \mathcal{A}_0, E_0) = \langle \{e, f\}, \{(f, f), (f, e)\} \rangle$. □

Theorem 4.7. *Let $\mathcal{A}_0$ be an AF, and $\mathcal{A} = u(\mathcal{A}_0)$ be the AF resulting from performing update $u = \pm(a, b)$ on $\mathcal{A}_0$. Let $E_0 \in \mathcal{E}_\mathcal{S}(\mathcal{A}_0)$ be an extension for $\mathcal{A}_0$ under a semantics $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{gr}\}$. Then, if $\mathcal{E}_\mathcal{S}(\mathcal{R}(u, \mathcal{A}_0, E_0))$ is not empty, then there is an extension $E \in \mathcal{E}_\mathcal{S}(\mathcal{A})$ for the updated AF $\mathcal{A}$ such that $E = (E_0 \setminus \mathcal{I}(u, \mathcal{A}_0, E_0)) \cup E_d$ where E_d is an S-extension for reduced AF $\mathcal{R}(u, \mathcal{A}_0, E_0)$.*

Example 4.8. Continuing our example, for the preferred semantics, let $E_0 = \{a, f, g\}$ and $u = +(c, f)$, we have that $\mathcal{I}(u, \mathcal{A}_0, E_0) = \{f, e\}$, and $\mathcal{R}(+(c, f), \mathcal{A}_0, E_0) = \langle\{e, f\}, \{(f, f), (f, e)\}\rangle$. Thus, using the theorem, there is an extension E of the updated AF such that $E = (\{a, f, g\} \setminus \{f, e\}) \cup E_d$ where $E_d = \emptyset$ is a preferred extension of the reduced AF. In fact, $E = \{a, g\} \in \mathcal{E}_{\texttt{pr}}(u(\mathcal{A}_0))$. □

It is worth noting that the set of extensions of an AF can be empty only for the stable semantics. Thus, in the case that this happens for the reduced AF (i.e., $\mathcal{E}_\mathcal{S}(\mathcal{R}(u, \mathcal{A}_0, E_0)) = \emptyset$), the theorem does not give a method to determine an extension of the updated AF, as shown in the following example.

Example 4.9. Let $\mathcal{A}_0 = \langle\{a, b, c, d, e\}, \{(a, b), (b, c), (b, d), (c, d), (c, e), (e, c)\}\rangle$. Its stable extensions are $\{a, c\}$ and $\{a, d, e\}$. For update $u = +(d, d)$, depending on the initial extension, the influenced set is either $\mathcal{I}(u, \mathcal{A}, \{a, c\}) = \emptyset$ (as u is irrelevant w.r.t. $\{a, c\}$ and $\texttt{st}$) or $\mathcal{I}(u, \mathcal{A}, \{a, d, e\}) = \{d\}$. Thus, starting from the extension $\{a, c\}$ we directly know $\{a, c\}$ is a stable extension of the updated AF. However, starting from $\{a, d, e\}$, the reduced AF will be $\mathcal{R}(u, \mathcal{A}_0, \{a, d, e\}) = \langle\{d\}, \{(d, d)\}\rangle$, which has no stable extension. In this case, the theorem does not provide a stable extension of the updated AF, thought a stable extension exists: that obtained by starting from the initial extension $\{a, c\}$.

It is worth noting that, if we consider the preferred semantics, for which the starting extensions are again $\{a, c\}$ and $\{a, d, e\}$, a preferred extension of the updated AF can be obtained no matter what starting extension is chosen. In particular, as the preferred extension for reduced AF $\langle\{d\}, \{(d, d)\}\rangle$ is the empty set, it follows that $(\{a, d, e\} \setminus \{d\}) \cup \emptyset = \{a, e\}$ is a preferred extension of the updated AF. □

4.2.1 Incremental Algorithm

Algorithm 3 computes an extension of an updated AF. Besides taking as input an initial AF $\mathcal{A}_0$, an update u, a semantics $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{gr}\}$, and an extension $E_0 \in \mathcal{E}_\mathcal{S}(\mathcal{A}_0)$, it also takes as input a function that computes an $\mathcal{S}$-extension for an AF, if any. In particular, function $\mathsf{Solver}_\mathcal{S}(\mathcal{A})$ will be used to compute an extension of the reduced AF, which will be then combined with the portion of the initial extension that does not change in order to obtain an extension for the updated AF (as stated in Theorem 4.7).

More in detail, Algorithm 3 works as follows. First, the influenced set of $\mathcal{A}_0$ w.r.t. update u and the given initial extension E_0 is computed (Line 1). If it is empty, then E_0 will be still an extension of the updated AF under the given semantics $\mathcal{S}$, and thus it is returned (Line 3). Otherwise, the reduced AF $\mathcal{A}_d$ is computed at Line 4, and function $\mathsf{Solver}_\mathcal{S}$ is invoked to compute an $\mathcal{S}$-extension of $\mathcal{A}_d$, if any. If $\mathcal{S} \in \{\texttt{co}, \texttt{pr},$

Algorithm 3 Incr-Alg($\mathcal{A}_0, u, \mathcal{S}, E_0$, Solver$_\mathcal{S}$)

Input: AF $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$, update $u = \pm(a, b)$,
semantics $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{gr}\}$, extension $E_0 \in \mathcal{E}_\mathcal{S}(\mathcal{A}_0)$,
function Solver$_\mathcal{S}(\mathcal{A})$ returning an $\mathcal{S}$-extension for AF $\mathcal{A}$ if it exists, $\perp$ otherwise;
Output: An $\mathcal{S}$-extension $E \in \mathcal{E}_\mathcal{S}(u(\mathcal{A}_0))$ if it exists, $\perp$ otherwise;
1: $S = \mathcal{I}(u, \mathcal{A}_0, E_0)$;
2: **if** $(S = \emptyset)$ **then**
3: **return** E_0;
4: $\mathcal{A}_d = \mathcal{R}(u, \mathcal{A}_0, E_0)$;
5: Let $E_d =$ Solver$_\mathcal{S}(\mathcal{A}_d)$;
6: **if** $(E_d \neq \perp)$ **then**
7: **return** $E = (E_0 \setminus S) \cup E_d$;
8: **else**
9: **return** Solver$_\mathcal{S}(u(\mathcal{A}_0))$;

$\texttt{gr}\}$, then $\mathcal{A}_d$ will have an extension E_d, which is combined with $E_0 \setminus S$ at Line 7 to get an extension for the updated AF. For the stable semantics, if $\mathcal{E}_{\texttt{st}}(\mathcal{R}(u, \mathcal{A}_0, E_0))$ is not empty, then the algorithm proceeds as for the other semantics (Line 7). Otherwise, function Solver$_\mathcal{S}$ is invoked to compute a stable extension of the whole updated AF $u(\mathcal{A}_0)$, if any.

Theorem 4.10. *Let $\mathcal{A}_0$ be an AF, $u = \pm(a, b)$, and $E_0 \in \mathcal{E}_\mathcal{S}(\mathcal{A}_0)$ an extension for $\mathcal{A}_0$ under $\mathcal{S} \in \{\texttt{co}, \texttt{pr}, \texttt{st}, \texttt{gr}\}$. If Solver$_\mathcal{S}$ is sound and complete then Algorithm 3 computes $E \in \mathcal{E}_\mathcal{S}(u(\mathcal{A}_0))$ if $\mathcal{E}_\mathcal{S}(u(\mathcal{A}_0)) \neq \emptyset$, otherwise it returns $\perp$.*

The proposed approach extends to the case of multiple updates, as initially discussed in Section 2.4. In fact, performing a set of updates can be reduced to performing a single update (see Definition 2.13).

We would also remark that the presented approach does not apply for the case of recomputing the whole set of extensions. As an example, when considering the AF $\mathcal{A} = \langle \{a, b\}, \{(a, b)\} \rangle$, Algorithm 3, having as input the update $+(b, a)$, and the only (initial) preferred extension $\{a\}$, is not able to build the (updated) preferred extension $\{b\}$ for the updated AF $+(b, a)(\mathcal{A})$, as it only returns $\{a\}$ that is a preferred extension for $+(b, a)(\mathcal{A})$.

4.3 Implementation and Experiments

We implemented a C++ prototype and, for each semantics $\mathcal{S}$, compared the performance of the proposed technique with that of the solver that won the ICCMA'15 competition for the computational task $\mathcal{S}$-SE: Given an AF, determine some $\mathcal{S}$-extension.

Datasets. We used the ICCMA'15 benchmarks. The benchmark AFs have three different AFs' structures: (i) `TestSetGr` consists of AFs with a very large grounded extension and many arguments in general; (ii) `TestSetSt` consists of AFs with

many complete/preferred/stable extensions; and (iii) `TestSetSCC` consists of AFs with a rich structure of strongly connected components. In particular, for each of these test sets, three classes of AFs of different sizes were defined: `Small`, `Medium`, and `Large`. However, `TestSetStLarge` was removed from the competition as the majority of the solvers could not solve any of those AFs. For the sake of brevity, we presents the results obtained for `TestSetGrSmall` and `TestSetGrLarge`, and `TestSetStSmall` and `TestSetStMedium` but similar results were obtained also for the other datasets.

Methodology. For each semantics $\mathcal{S}$, for each AF $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ in each dataset, we considered every $\mathcal{S}$-extension E_0 of $\mathcal{A}_0$ as an initial extension. Then, when considering single updates, we randomly selected an update of the form $\pm(a, b)$, while for multiple updates we randomly generated a set U of single updates. Next, we computed an $\mathcal{S}$-extension E for the updated AF $u(\mathcal{A}_0)$ by calling Algorithm 3, where, for each semantics $\mathcal{S}$, we used as $\mathsf{Solver}_{\mathcal{S}}$ the solver that won the ICCMA'15 competition for the task $\mathcal{S}$-SE: *CoQuiAAS* [88] for $\mathcal{S} = \mathtt{co}$ and $\mathcal{S} = \mathtt{gr}$, *Cegartix* [66] for $\mathcal{S} = \mathtt{pr}$, and *ASPARTIX-D* [77] for $\mathcal{S} = \mathtt{st}$. Then, the average run time of Algorithm 3 to compute an $\mathcal{S}$-extension was compared with the average run time of the best ICCMA solver to compute an $\mathcal{S}$-extension for $u(\mathcal{A}_0)$ from scratch.

Results. Figure 7.3 reports the average run times (log scale) of the competitors and *Incr-Alg* for different semantics and datasets. In particular, the graphs also report the run times of *Incr-Alg* for recomputing the extensions after performing sets S of updates simultaneously, with $|S|$ being a percentage of the number of attacks in the initial AFs. The run times of the competitors for these cases were almost equal to their run times for single updates, and are not shown for the sake of readability of the graphs. Besides the data points, for each series, Figure 7.3 also shows the (solid) lines obtained by linear regression. However, some datasets consist of clusters of AFs that differ substantially on the number of arguments/attacks, and thus in these cases the linear regression line would just connect the centroids of the clusters, becoming not useful. For this reason, we only show the largest cluster for `TestSetStSmall` (the other one consists of only 3 data points), and the two clusters of `TestSetStMedium` separately (the former for $\mathcal{S} = \mathtt{pr}$ and the latter for $\mathcal{S} = \mathtt{st}$).

Moreover, the results obtained for the complete semantics are not shown as they were analogous to those obtained for the grounded semantics. Also, considering updates consisting of adding/removing arguments does not affect the efficiency.

The experiments also showed that, on average, the size of the reduced AF w.r.t. that of the input AF is about 9% for single updates and 52% for multiple updates with about 1% of the attacks updated. Moreover, considering also addition/removal of arguments does not affect the efficiency.

From these results, we can draw the following conclusions:

– Algorithm 3 significantly outperforms the competitors that compute the extensions from scratch for single updates. In fact, on average, the proposed technique is two orders of magnitude faster than them. Moreover, the harder the computation

from scratch is, the larger the improvements are. That is, the results show that the improvements obtained for $\mathcal{S} \in \{\texttt{st}, \texttt{pr}\}$ go beyond those for $\mathcal{S} \in \{\texttt{gr}, \texttt{co}\}$.
– Algorithm 3 remains faster than the competitors even when recomputing an extension after performing a quite large number of updates simultaneously. In particular, in the graphs we show the threshold percentages of updated attacks up to which the incremental approach for multiple updates is faster than the computation from scratch. Although the initial extension is never preserved by modifying the framework as in Theorem 2.14 due to the presence of additional arguments in the obtained AF $\mathcal{A}^U_{E_0}$ (e.g., arguments u and w), only relevant updates are taken into account in the construction of $\mathcal{A}^U_{E_0}$—arguments not involved in such updates are not influenced.
– Finally, our experiments have also shown that for sets of updates regarding a relevant portion of the input AF (on average at least 1% of the attacks for $\mathcal{S} \in \{\texttt{st}, \texttt{pr}\}$ and $0,1\%$ of the attacks for $\mathcal{S} \in \{\texttt{gr}, \texttt{co}\}$) recomputing extensions after applying them simultaneously is faster than recomputing extensions after applying them sequentially. Indeed, the green lines in the graphs are mostly below the (dashed) orange lines representing the run times of recomputing extensions after applying the updates sequentially.

4.4 Summary

We introduced a technique enabling any non-incremental algorithm to be used as an incremental one for computing some extension of dynamic AFs. The work proposed in this chapter advanced existing approaches for computing extensions of dynamic AFs from two standpoints: (i) we considered general forms of updates (i.e., not limited forms of updates such as for instance weak expansions [20]) and (ii) we identified a tighter portion of the updated AF to be examined for recomputing the semantics. For instance, the reduced AF is significantly smaller than the *conditioned* AF (CAF) in [89]. Additional experiments showed that the size of CAF is 91% of the size of the initial AF, while the size of the reduced AF is only 9%. Also, using the reduced AF is more efficient than using CAF: the run time of the technique presented in the chapter turned out to be only 1% of the run time of using ICCMA solvers taking as input CAFs, that is, still two orders of magnitude faster.

Future work will be devoted to (i) applying this technique to other argumentation semantics and (ii) extending it to cope with other computational problems, such as enumerating all the extensions and deciding credulous acceptance; the problem of incrementally deciding skeptical acceptance will be treated in Chapter 6.

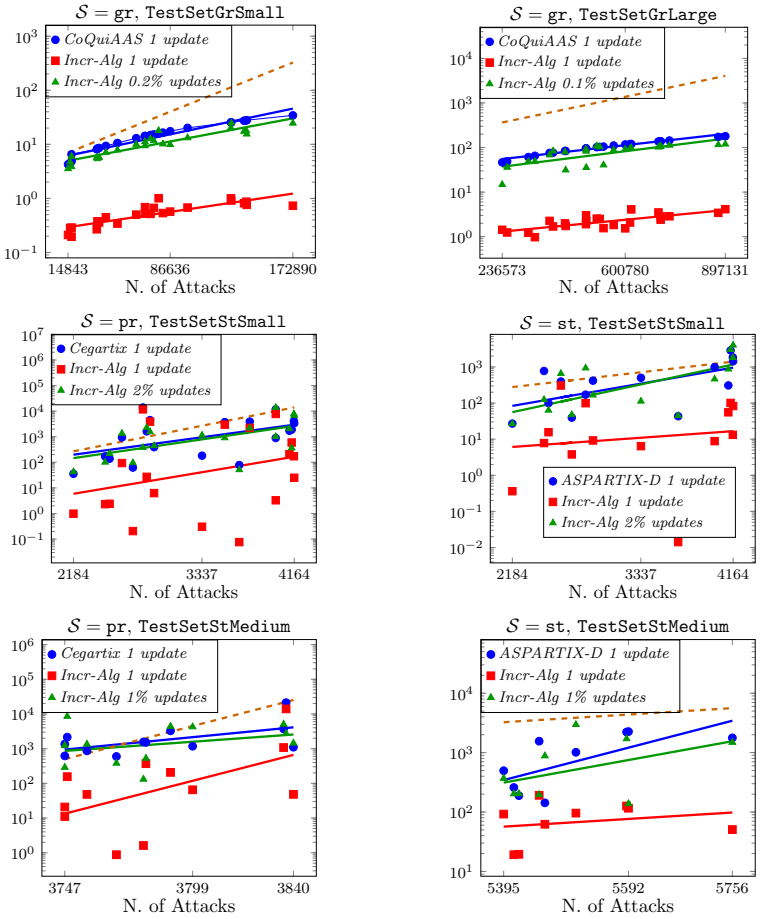


Fig. 4.2: Run times (ms) of ICCMA solvers and *Incr-Alg* for different semantics $\mathcal{S}$ over different datasets (showed on the top of each graph) versus the number of attacks.

5

Efficient Computation of Extensions in Dynamic Bipolar Argumentation Frameworks

"Everything in nature is bipolar, or has a positive and a negative pole."

– Ralph Waldo Emerson

Bipolar argumentation frameworks (BAFs) extend Dung's argumentation frameworks to explicitly represent the notion of support between arguments, in addition to that of attack. BAFs can be profitably used to model disputes between two or more agents, with the aim of deciding the sets of arguments that should be accepted to support a point of view in a discussion. However, since new arguments, attacks, and supports are often introduced to take into account new available knowledge, BAFs as well as the set of accepted arguments (under a given semantics) change over the time.

In this chapter we first tackle the problem of efficiently recomputing sets of accepted arguments of dynamic BAFs (under the preferred and stable semantics). Focusing on a deductive interpretation of the support relation, we introduce an incremental approach that, given an initial BAF, an initial extension for it, and an update, computes an extension of the updated BAF. This is achieved by introducing a meta-argumentation transformation according to which an initial BAF, as well as its extension and an update, are transformed into a plain argumentation framework (AF) with a suitable initial extension and update. Thanks to the use of the meta-argumentation intermediate level, the proposed approach is able to incorporate existing AF-solvers and an incremental technique for plain AFs in order to compute an extension of the updated BAF. Moreover, the proposed approach can be seamlessly applied to a more general form of BAFs, namely *Extended Bipolar Argumentation Frameworks* (EAFs), where defeasible supports and defeats are modelled by means of second-order attacks (i.e., attacks toward elements of the support or attack relation).

We experimentally validated the approach on both BAFs and EAFs. The experiments showed that, on average, the technique is almost 100 times faster than computing extensions of updated BAFs or EAFs from scratch.

5.1 Computing Extensions of Updated BAFs

In this section, given a BAF $\mathcal{B}_0$, a preferred/stable extension E_0 for $\mathcal{B}_0$, and an update u, we address the problem of recomputing a preferred/stable extension E of the updated BAF $u(\mathcal{B}_0)$.

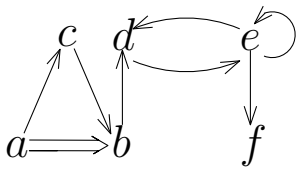


Fig. 5.1: BAFs $\mathcal{B}_0$ of Example 2.5.

Figure 5.1 reports the bipolar interaction graph of BAF $\mathcal{B}_0$ presented in Example 2.5 that will be used as running example of the chapter. We start by presenting a baseline approach and then introduce the incremental technique.

Baseline approach. A first approach to compute an extension of an updated BAF is that of computing it from scratch, without using the information provided by the update and the initial status of arguments. Specifically, a baseline approach to compute an extension E for an updated BAF $\mathcal{B}$ consists of the following steps:

(i) build the meta-AF $\mathcal{M}$ for $\mathcal{B}$ of Definition 2.7;
(ii) compute an extension E^m for $\mathcal{M}$ by using one of the available solvers for AFs [108, 109]; and
(iii) obtain an extension E for $\mathcal{B}_0$ by using the result of Proposition 2.8.

This procedure can be made more efficient by using the following compact meta-AF, which is obtained from that of Definition 2.7 by avoiding to introduce meta-arguments $X_{a,b}$ and $Y_{a,b}$. However, these arguments will turn out to be useful for dealing with second-order attacks in Section 5.2.

Definition 5.1 (Compact Meta-AF). *Given a BAF $\mathcal{B} = \langle A, \Sigma, \Pi \rangle$, the compact meta-AF for $\mathcal{B}$ is $\mathcal{M} = \langle A^m, \Sigma^m \rangle$ where:*

i) $A^m = A \cup \{Z_{a,b} \mid (a,b) \in \Pi\}$
ii) $\Sigma^m = \Sigma \cup \{(b, Z_{a,b}), (Z_{a,b}, a) \mid (a,b) \in \Pi\}$

The following proposition straightforwardly follows from Proposition 2.8.

Proposition 5.2. *Let $\mathcal{B} = \langle A, \Sigma, \Pi \rangle$ be a BAF, $\mathcal{M}$ the compact meta-AF for $\mathcal{B}_0$, and $\mathcal{S} \in \{\mathtt{pr}, \mathtt{st}\}$ a semantics. For each $E \in \mathcal{E}_\mathcal{S}(\mathcal{B})$, there is an extension $E^m \in \mathcal{E}_\mathcal{S}(\mathcal{M})$ such that $E = E^m \cap A$, and vice versa.*

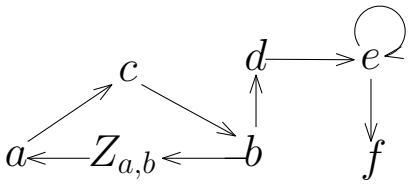


Fig. 5.2: Compact Meta-AF for the BAF of Example 2.5.

Example 5.3. The compact meta-AF for the BAF of Example 2.5 is reported in Figure 5.2. $\{a, b\}$ is the preferred extension for the compact meta-AF, as well as for the BAF, corresponding to the preferred extension given in Example 2.9. The stable extension for the compact meta-AF is $\{c, d, f, Z_{a,b}\}$, which corresponds to the stable extension $\{c, d, f\}$ of Example 2.9. □

Incremental approach. The baseline approach does not make use of the initial extension E_0 for $\mathcal{B}_0$ and of the update u to be performed. On the contrary, the incremental approach we present relies on profitably using this kind of information to improve efficiency and avoid wasted effort during the computation.

The approach to recompute a preferred/stable extension E of an updated BAF $u(\mathcal{B}_0)$ consists of the following three main steps.

1. ***Checking for irrelevant updates.*** We identify conditions ensuring that a given extension E_0 for the initial BAF $\mathcal{B}_0$ (under the preferred or stable semantic) is still an extension for the updated BAF $u(\mathcal{B}_0)$. In such case, the update u does not invalidate the initial extension E_0, and it will be immediately returned by the algorithm.
2. ***Build a suitable (compact) meta-AF***. Given a triple $\langle \mathcal{B}_0, E_0, u \rangle$ consisting of an initial BAF, an extension, and an update, we transform it into a triple $\langle \mathcal{M}_0, E_0^m, u^m \rangle$ consisting of a (compact) meta-AF $\mathcal{M}_0$, an extension E_0^m, and an update u^m for $\mathcal{M}_0$. The transformation we propose is based on the meta-argumentation approach proposed in [36] (cfr. Definitions 2.7 and 5.1), though in this case we need to take into account the initial extension E_0 and the update u.
3. ***Incremental computation on the meta-AF***. Given the AF $\mathcal{M}_0$, its extension E_0^m, and the update u^m for $\mathcal{M}_0$, we use the incremental technique proposed in Chapter 4 for computing an extension E^m of the updated AF $u^m(\mathcal{M}_0)$ w.r.t. E_0^m. The technique identifies a reduced AF sufficient to compute an extension of the whole AF and use state-of-the-art algorithms to recompute an extension of the reduced AF only. Then from extension E^m of the updated AF $u^m(\mathcal{M}_0)$, we derive an extension E of an updated BAF $u(\mathcal{B}_0)$.

In the next sections we describe in detail these three steps.

Table 5.1: Cases for which $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$ *for* $u = +(a \to b)$.

update $+(a \to b)$		$L_0(b)$ IN	$L_0(b)$ UNDECIDED	$L_0(b)$ OUT
$L_0(a)$	IN			pr, st
	UNDECIDED			pr
	OUT	pr,st		pr,st

Table 5.2: Cases for which $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$ *for* $u = +(a \Rightarrow b)$.

update $+(a \Rightarrow b)$		$L_0(b)$ IN	$L_0(b)$ UNDECIDED	$L_0(b)$ OUT
$L_0(a)$	IN	pr,st		
	UNDECIDED			
	OUT	pr,st	pr	pr,st

Table 5.3: Cases for which $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$ *for* $u = -(a \to b)$.

update $-(a \to b)$		$L_0(b)$ IN	$L_0(b)$ UNDECIDED	$L_0(b)$ OUT
$L_0(a)$	IN	NA	NA	
	UNDECIDED	NA		pr
	OUT	pr,st	pr	pr,st

5.1.1 Checking for Irrelevant Updates

Given a BAF $\mathcal{B}_0$, an initial extension E_0 whose corresponding labelling is L_0, and an update u, for each pair of initial statuses $L_0(a)$ and $L_0(b)$ of the arguments involved in the update, Tables 5.1 – 5.4 tell us if E_0 is still an extension after performing the update (under the preferred or stable semantics).

Proposition 5.4 (Extension preservation). *Let $\mathcal{B}_0$ be a BAF, $\mathcal{S} \in \{\texttt{pr}, \texttt{st}\}$ a semantics, $E_0 \in \mathcal{E}_{\mathcal{S}}(\mathcal{B}_0)$ an extension of $\mathcal{B}_0$ under semantics $\mathcal{S}$, L_0 the labelling corresponding to E_0, and u an update.*
If $\mathcal{S}$ is in the cell $\langle L_0(a), L_0(b)\rangle$ of

a) Table 5.1 and $u = +(a \to b)$;
b) Table 5.2 and $u = +(a \Rightarrow b)$;
c) Table 5.3 and $u = -(a \to b)$;
d) Table 5.4 and $u = -(a \Rightarrow b)$;

then $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$, where u is the update specified above.

Thus, if some of the conditions of Proposition 5.4 hold, then the given initial extension of the initial BAF is still an extension of the updated one, and thus Step

Table 5.4: Cases for which $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$ *for* $u = -(a \Rightarrow b)$.

update $-(a \Rightarrow b)$		$L_0(b)$		
		IN	UNDECIDED	OUT
$L_0(a)$	IN	pr,st	NA	NA
	UNDECIDED	pr		NA
	OUT	pr,st	pr	

2) and Step 3) of the incremental approach can be skipped — the algorithm just returns the initial extension which is also an extension for the updated BAF. For instance, considering the BAF $\mathcal{B}_0$ of Example 2.5, the update $u = -(b \rightarrow d)$, and preferred extension $E_0 = \{c, d, f\}$, since $L_0(b) = \text{OUT}$ and $L_0(d) = \text{IN}$, Table 5.3 says that $E_0 = \{c, d, f\}$ is still an extension of the BAF $u(\mathcal{B}_0)$. Similarly, considering $u = +(c \Rightarrow f)$, and again the preferred extension $E_0 = \{c, d, f\}$, since $L_0(c) = L_0(f) = \text{IN}$, Table 5.2 tell us that E_0 is still an extension of the updated BAF.

Conditions similar to those of Proposition 5.4 were identified in Chapter 4 for updates for Dung's AFs, that is, AFs where the support relation is not considered. However, those conditions could be used only at Step 3) when applying the technique of Chapter 4 to the meta-AF $\mathcal{M}_0$, that is, after performing the transformation of Step 2). Therefore, to avoid to uselessly perform Step 2) and to immediately discover irrelevant updates, we provided Proposition 5.4 that extends the results of Chapter 4 to updates for BAFs and allows us to directly check at Step 1) for cases for which initial extensions for BAFs are preserved.

The results of Proposition 5.4 follow from their counterparts for updates of Dung's AFs given in Chapter 4. In fact, since using Step 2—whose details are given in Section 5.1.2— an update u for BAF $\mathcal{B}_0$ with initial extension E_0 corresponds to an update u^m for a meta-AF $\mathcal{M}_0$ with extension E_0^m, the results of Proposition 5.4 can be easily obtained by checking whether u^m is irrelevant for $\mathcal{M}_0$ with respect to E_0^m.

5.1.2 The Meta-Argumentation Framework for Incremental Computation

Given the initial BAF $\mathcal{B}_0$ and an updated u for it, we define the corresponding meta-argumentation framework as follows.

Definition 5.5 ((Compact) Meta-AF for Updates). *Let $\mathcal{B} = \langle A, \Sigma, \Pi \rangle$ be a BAF, and u an update for $\mathcal{B}$ of the form $u = \pm(c \rightarrow d)$ or $u = \pm(e \Rightarrow f)$. Then, the meta-AF for $\mathcal{B}$ w.r.t. u is $\mathcal{CM}(\mathcal{B}, u) = \langle A^m, \Sigma^m \rangle$ where:*

i) $A^m = A \cup \{Z_{a,b} \mid (a,b) \in \Pi\} \cup \{Z_{e,f} \mid u = +(e \Rightarrow f)\}$

ii) $\Sigma^m = \Sigma \cup \{(b, Z_{a,b}), (Z_{a,b}, a) \mid (a,b) \in \Pi\} \cup \{(f, Z_{e,f}) \mid u = +(e \Rightarrow f)\}$

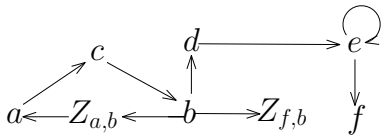


Fig. 5.3: $\mathcal{CM}(\mathcal{B}_0, +(f \Rightarrow b))$: compact meta-AF for the BAF $\mathcal{B}_0$ of Figure 5.1 w.r.t. update $u = +(f \Rightarrow b)$.

For instance, the meta-AF $\mathcal{CM}(\mathcal{B}_0, +(f \Rightarrow b))$ for the BAF $\mathcal{B}_0$ of Figure 5.1 w.r.t the update $u = +(f \Rightarrow b)$ is shown in Figure 5.3 where attack $(b, Z_{f,b})$ is added. It is worth noting that the meta-AF for negative updates (e.g., $\mathcal{CM}(\mathcal{B}_0, -(a \rightarrow c))$) coincides with the compact meta-AF of Figure 5.2.

We have defined a meta-argumentation framework that builds on (the compact version of) that proposed in [36] and considers additional meta-arguments (e.g., $Z_{f,b}$ in Figure 5.3) and attacks (e.g., $(b, Z_{f,b})$ in Figure 5.3) that will allow us to simulate addition updates to be performed on BAF $\mathcal{B}_0$ by means of updates performed on the corresponding the meta-AF $\mathcal{CM}(\mathcal{B}_0, u)$, as follows.

Definition 5.6 (Updates for the (Compact) Meta-AF). *Let $\mathcal{B} = \langle A, \Sigma, \Pi \rangle$ be a BAF, and u an update for $\mathcal{B}$ of the form $u = \pm(c \rightarrow d)$ or $u = \pm(e \Rightarrow f)$. The corresponding update u^m for $\mathcal{CM}(\mathcal{B}, u)$ is as follows:*

$$u^m = \begin{cases} +(Z_{e,f} \rightarrow e) \;\; if \; u = +(e \Rightarrow f) \\ -(Z_{e,f} \rightarrow e) \;\; if \; u = -(e \Rightarrow f)) \\ +(c \rightarrow d) \;\; if \; u = +(c \rightarrow d) \\ -(c \rightarrow d)) \;\; if \; u = -(c \rightarrow d)) \end{cases}$$

Basically, support updates for the given bipolar framework are translated in attacks updates on the corresponding meta-AF of Definition 5.1, while attacks updates can be directly performed on the meta-AF. For instance, given the BAF $\mathcal{B}_0$ of Example 2.5 and $u = +(f \Rightarrow b)$, the update for $\mathcal{CM}(\mathcal{B}_0, +(f \Rightarrow b))$ is $u^m = +(Z_{f,b}, f)$ (see Figure 5.3), while update $u = -(a \rightarrow c)$ for $\mathcal{B}_0$ simply corresponds to the update $u^m = -(a \rightarrow c)$ for $\mathcal{CM}(\mathcal{B}_0, -(a \rightarrow c))$.

The last ingredient we need before being ready to apply the incremental technique of Chapter 4 is the initial extension E_0^m for $\mathcal{CM}(\mathcal{B}_0, u)$. We obtain it from that of initial BAF $\mathcal{B}_0$ by propagating the labels of the arguments in $\mathcal{B}_0$ as follows.

Definition 5.7 (Initial Labelling for the Meta-AF). *Given a BAF $\mathcal{B}_0 = \langle A, \Sigma, \Pi \rangle$ and its initial labelling L_0, the corresponding initial labelling L_0^m for the meta-AF $\mathcal{CM}(\mathcal{B}_0, u) = \langle A^m, \Sigma^m \rangle$ is as follows:*

- $\forall a \in A \cap A^m : \; L_0^m(a) = L_0(a)$;
- $\forall Z_{a,b} \in A^m$:
  $L_0^m(Z_{a,b}) = \text{IN} \; if L_0(b) = \text{OUT}$,
  $L_0^m(Z_{a,b}) = \text{OUT} \; if L_0(b) = \text{IN}$,
  $L_0^m(Z_{a,b}) = \text{UNDECIDED} \; if L_0(b) = \text{UNDECIDED}$.

Algorithm 4 Incr-BAF($\mathcal{B}_0, u, E_0, \mathcal{S}$, Solver$_\mathcal{S}$)

Input: BAF $\mathcal{B}_0 = \langle A_0, \Sigma_0\, \Pi_0 \rangle$,
update u of the form $u = \pm(a \Rightarrow b)$ or $u = \pm(a \rightarrow b)$,
an initial $\mathcal{S}$-extension E_0 for $\mathcal{B}_0$,
semantics $\mathcal{S} \in \{\mathtt{pr}, \mathtt{st}\}$,
function Solver$_\mathcal{S}(\mathcal{A})$ returning an $\mathcal{S}$-extension for AF $\mathcal{A}$ if it exists, $\perp$ otherwise;
Output: An $\mathcal{S}$-extension E for $u(\mathcal{B}_0)$ if it exists, $\perp$ otherwise;

```
if checkProp(B_0, u, E_0, S) then
    return E_0;
Let M_0 = CM(B_0, u) be the meta-AF for B_0 w.r.t. u (cf. Definition 5.5);
Let u^m be the update for M_0 corresponding to u (cf. Definition 5.6);
Let E_0^m be the initial S-extension for M_0 corresponding to E_0 (cf. Definition 5.7);
Let E^m = Incr-Alg(M_0, u^m, S, E_0^m, Solver_S);
if (E^m ≠ ⊥) then
    return E = (E^m ∩ A_0);
else
    return ⊥;
```

For instance, with reference to Figure 5.3, and the preferred extension $E_0 = \{c, d, f\}$ for the BAF $\mathcal{B}_0$ of Example 2.5, we have that the labelling corresponding to E_0 is $L_0 = \langle \{c, d, f\}, \{a, b, e\}, \emptyset \rangle$, and thus $L_0^m(Z_{f,b}) = \text{IN}$ since $L_0^m(b) = L_0(b) = \text{OUT}$. Similarly, $L_0^m(Z_{a,b}) = \text{IN}$ since $L_0^m(b) = \text{OUT}$. In turn, we have that, $E_0^m = \{c, d, Z_{a,b}, Z_{f,b}, f\}$.

The following proposition characterizes the relationship between extensions of updated BAFs and extensions of updated meta-AFs.

Proposition 5.8. *Let $\mathcal{B}_0 = \langle A, \Sigma, \Pi \rangle$ be a BAF, $\mathcal{S} \in \{\mathtt{pr}, \mathtt{st}\}$ a semantics, and $E_0 \in \mathcal{E}_\mathcal{S}(\mathcal{B}_0)$ an extension of $\mathcal{B}_0$ under $\mathcal{S}$.*
Let $\mathcal{M}_0 = \mathcal{CM}(\mathcal{B}_0, u)$ be meta-AF for $\mathcal{B}_0$ w.r.t. u, E_0^m the initial $\mathcal{S}$-extension for $\mathcal{M}_0$ corresponding to E_0, and u^m the update for $\mathcal{M}_0$ corresponding to u.
Then, there is $E \in \mathcal{E}_\mathcal{S}(u(\mathcal{B}_0))$ iff there is $E^m \in \mathcal{E}_\mathcal{S}(u^m(\mathcal{M}_0))$ such that $E = E^m \cap A$.

5.1.3 Incremental Algorithm

Given a BAF $\mathcal{B}_0$, a semantics $\mathcal{S} \in \{\mathtt{pr}, \mathtt{st}\}$, an extension $E_0 \in \mathcal{E}_\mathcal{S}(\mathcal{B}_0)$, and an update u of the form $u = \pm(a \Rightarrow b)$ or $u = \pm(a \rightarrow b)$, we define an incremental algorithm (Algorithm 4) for computing an extension E of the updated BAF $u(\mathcal{B}_0)$, if it exists. [1]

[1] Observe that for the stable semantics, the set of extensions $\mathcal{E}_{\mathtt{st}}(u(\mathcal{B}_0))$ of the updated BAF may be empty; in this case, the algorithm returns $\perp$.

Algorithm 4 works as follows. It first checks if the initial extension E_0 is still an extension of the updated BAF at Line 1, where $checkProp(\mathcal{B}_0, u, E_0, \mathcal{S})$ is a function returning *true* iff some of the conditions of Proposition 5.4 hold. If this is the case, it immediately returns the initial extension. Otherwise, it computes the (meta) AF $\mathcal{M}_0$ (Line 3), the update u^m for $\mathcal{M}_0$ (Line 4), and the initial $\mathcal{S}$-extension E_0^m for $\mathcal{M}_0$ (Line 5). Next, it invokes function Incr-Alg which encodes the incremental algorithm proposed in Chapter 4 for Dung's AFs. As shown in Chapter 4 Incr-Alg (Algorithm 3) takes as input the parameters $\mathcal{M}_0, u^m, \mathcal{S}, E_0^m$, and $\mathsf{Solver}_{\mathcal{S}}$, where $\mathsf{Solver}_{\mathcal{S}}$ is an external solver that can compute an $\mathcal{S}$-extension for the input AF. Finally, the extension of the updated BAF (if any) is obtained by projecting out the extension E^m returned by Incr-Alg over the set of arguments A_0 of the initial BAF (Line 10).

In the next section, we introduce a variant of Algorithm 4 taking as input an extended BAF where second-order attacks are considered.

5.2 Dealing with Second Order Attacks

In this section, we show how to extend the technique to consider *second-order attacks* [36] for BAFs, that is, (i) attacks from an argument to another attack, and (ii) attacks from an argument to a support. This allows the representation of both attacks towards the attack relation [15, 95] and a kind of *defeasible support* where the support itself can be attacked.[2]

In the following, after presenting the formal definition of BAFs extended with second-order attacks, as well as the formalization of updates for the extended framework, analogously to what done in Section 5.1.2 we first identify (additional) early termination conditions for checking whether a second-order update is irrelevant. Then, we build on the definition of meta-AF introduced in [36] for encoding second-order attacks, extend it to deal with updates for such kind of BAFs, and finally discuss how to modify Algorithm 4 to enable the computation of extensions of extended BAFs.

An *Extended Bipolar Argumentation Framework* (EAF for short) [36] is a quadruple $\langle A, \Sigma, \Pi, \Delta \rangle$, where $\langle A, \Sigma, \Pi \rangle$ is a BAF and Δ is a binary relation over $A \times (\Sigma \cup \Pi)$ whose elements are called *second-order attacks*.

In the following, a second-order attack from an argument a to an attack (b, c) will be denoted as $(a \twoheadrightarrow (b \rightarrow c))$, while an attack from an argument a to a support (b, c) will be denoted as $(a \twoheadrightarrow (b \Rightarrow c))$.

Example 5.9. $\mathcal{EB}_0 = \langle A_0, \Sigma_0, \Pi_0, \Delta_0 \rangle$ is an EAF where $\Delta_0 = \{(a, (b, d))\}$ is the set of second-order attacks. Its graph is shown in Fig.5.4, where second-order attacks are drawn using double-headed arrows. □

The semantics of an EAF can be given by means of the following meta-AF, which extends that in Definition 2.7 by taking into account second order attacks.

[2] The technique can be further extended to consider (second-order) attacks from an attack to another attack [36].

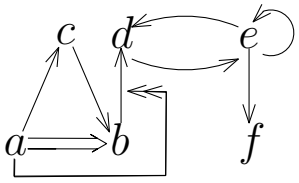


Fig. 5.4: EAF $\mathcal{EB}_0$ of Example 5.9.

Definition 5.10 (Meta-AF with Second-Order Attacks [36]). *The meta-AF for* $\mathcal{EB} = \langle A, \Sigma, \Pi, \Delta \rangle$ *is* $\mathcal{M} = \langle A^m, \Sigma^m \rangle$ *where:*

$$
\begin{aligned}
A^m \;=\; & A \cup \{X_{a,b}, Y_{a,b} \mid (a,b) \in \Sigma\} \cup \\
& \{Z_{a,b} \mid (a,b) \in \Pi\} \cup \\
& \{X_{a,(b,c)}, Y_{a,(b,c)} \mid (a,(b,c)) \in \Delta, (b,c) \in \Sigma\} \\
\Sigma^m \;=\; & \{(a, X_{a,b}), (X_{a,b}, Y_{a,b}), (Y_{a,b}, b) \mid (a,b) \in \Sigma\} \cup \\
& \{(b, Z_{a,b}), (Z_{a,b}, a) \mid (a,b) \in \Pi\} \cup \\
& \{(a, X_{a,(b,c)}), (X_{a,(b,c)}, Y_{a,(b,c)}), (Y_{a,(b,c)}, Y_{b,c}) \mid \\
& \qquad\qquad (a,(b,c)) \in \Delta, (b,c) \in \Sigma\} \cup \\
& \{(a, X_{a,(b,c)}), (X_{a,(b,c)}, Y_{a,(b,c)}), (Y_{a,(b,c)}, Z_{b,c}) \mid \\
& \qquad\qquad (a,(b,c)) \in \Delta, (b,c) \in \Pi\}.
\end{aligned}
$$

Thus, an attack of the form $(a \twoheadrightarrow (b \to c))$ is encoded as an attack towards the meta-argument $Y_{b,c}$ (that represents the fact that (b,c) is "active"), while an attack of the form $(a \twoheadrightarrow (b \Rightarrow c))$ is encoded as an attack toward the meta-argument $Z_{b,c}$. The meta-AF for the EAF of Example 5.9 is shown in Fig. 5.5.

Analogously to what stated in Proposition 2.8, extensions for an EAF $\mathcal{EB}$ are obtained from extensions for its meta-AF: E is an $\mathcal{S}$-extension for $\mathcal{EB}$ iff $E^m \in \mathcal{E}_{\mathcal{S}}(\mathcal{M})$ and $E = E^m \cap A$. Using this relationship, the notion of labelling can be extended to EAFs as well.

Example 5.11. For the meta-AF $\mathcal{M}$ of Fig.5.5, we have the following preferred extension: $\{a, b, d, f, Y_{a,c}, X_{c,b}, Y_{d,e}, Y_{a,(b,d)}, X_{e,e}, X_{e,d}, X_{e,f}, \}$, which corresponds to the extension $\{a, b, d, f\}$ of the EAF of Example 5.9.

A stable extension for the meta-AF is $\{c, d, f, X_{a,c}, Y_{c,b}, Z_{a,b}, X_{b,d}, Y_{d,e}\ X_{e,e}, X_{e,d}, X_{e,f}, X_{a,(b,d)}\}$. It corresponds to the extension $\{c, d, f\}$ for the EAF. □

5.2.1 Second-Order Updates and Early-Termination Conditions

In addition to the kinds of updates introduced in Section 2.4.2, for EAFs we also consider additions and deletions of second-order attacks. Specifically, the addition (resp., deletion) of a second-order attack from an argument a to an attack (b,c) will be denoted as $+(a \twoheadrightarrow (b \to c))$ (resp., $-(a \twoheadrightarrow (b \to c))$). Similarly, if (b,c) is a support, then the update will be denoted as $+(a \twoheadrightarrow (b \Rightarrow c))$ (resp., $-(a \twoheadrightarrow (b \Rightarrow c))$).

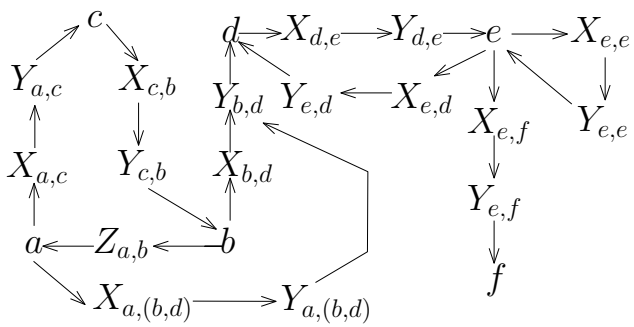


Fig. 5.5: Meta-AF for the EAF of Example 5.9.

We use $u(\mathcal{EB}_0)$ to denote the EAF resulting from the application of update u to an initial EAF $\mathcal{EB}_0$. For instance, given the EAF $\mathcal{EB}_0$ of Example 5.9 and the update $u = -(a \twoheadrightarrow (b \rightarrow d))$, we have that $u(\mathcal{EB}_0)$ is the EAF of Example 2.5.

The following proposition identifies cases for which a given initial extension of an EAF is preserved after performing an update. It is worth noting that conditions *a)–d)* identified in Proposition 5.4 for BAFs still hold for EAFs. The other conditions *e)–h)* are added to deal with irrelevant second-order updates.

Proposition 5.12. *Let $\mathcal{EB}_0$ be an EAF, $\mathcal{S} \in \{\texttt{pr}, \texttt{st}\}$ a semantics, $E_0 \in \mathcal{E}_{\mathcal{S}}(\mathcal{EB}_0)$ an extension of $\mathcal{EB}_0$ under semantics $\mathcal{S}$, L_0 the labelling corresponding to E_0, and u an update.*
If $\mathcal{S}$ is in the cell $\langle L_0(a), L_0(b)\rangle$ of

a) Table 5.1 and $u = +(a \rightarrow b)$;
b) Table 5.2 and $u = +(a \Rightarrow b)$;
c) Table 5.3 and $u = -(a \rightarrow b)$;
d) Table 5.4 and $u = -(a \Rightarrow b)$;
e) Table 5.5 and $u = +(a \twoheadrightarrow (b \rightarrow c))$;
f) Table 5.6 and $u = +(a \twoheadrightarrow (b \Rightarrow c))$;
g) Table 5.7 and $u = -(a \twoheadrightarrow (b \rightarrow c))$;
h) Table 5.8 and $u = -(a \twoheadrightarrow (b \Rightarrow c))$;

then $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{EB}_0))$, where u is the update specified above.

As done in Algorithm 4 for BAFs, the conditions of Proposition 5.12 can be used to recognize that an initial extension for an EAF $\mathcal{EB}_0$ is still valid for the updated EAF $u(\mathcal{EB}_0)$.

5.2.2 Enabling the Incremental Computation with Second-Order Attacks and Updates

To deal with second-order updates we need to extend the meta-AF of Definition 5.5, as well as the notions of update and initial labelling for the meta-AF of Definitions 5.6 and 5.7, respectively.

Table 5.5: Cases for which $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$ *for* $u = +(a \twoheadrightarrow (b \rightarrow c))$.

update $+(a \twoheadrightarrow (b \rightarrow c))$		$L_0(b)$ IN	$L_0(b)$ UNDECIDED	$L_0(b)$ OUT
$L_0(a)$	IN			pr, st
	UNDECIDED			pr
	OUT	pr,st		pr,st

Table 5.6: Cases for which $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$ *for* $u = +(a \twoheadrightarrow (b \Rightarrow c))$.

update $+(a \twoheadrightarrow (b \Rightarrow c))$		$L_0(c)$ IN	$L_0(c)$ UNDECIDED	$L_0(c)$ OUT
$L_0(a)$	IN	pr,st		
	UNDECIDED	pr		
	OUT	pr,st		pr,st

We start by introducing the (compact) meta-AF for dealing with updates in EAFs—it will be used in the variant of Algorithm 4 at Line 3, in place of the meta-AF of Definition 5.5.

Definition 5.13 (Meta-AF for Second-Order Updates). *Let $\mathcal{EB} = \langle A, \Sigma, \Pi, \Delta \rangle$ be an EAF, and u an update of one of the following forms:*

- $u = \pm(e \rightarrow f)$
- $u = \pm(e \Rightarrow f)$
- $u = \pm(e \twoheadrightarrow (g \rightarrow h))$
- $u = \pm(e \twoheadrightarrow (g \Rightarrow h))$.

Then, the (compact) meta-AF for $\mathcal{EB}$ w.r.t. u is
$\mathcal{CM}(\mathcal{EB}, u) = \langle A^m, \Sigma^m \rangle$ where:

$$
\begin{aligned}
A^m \;=\; & A \cup \{Z_{a,b} \mid (a,b) \in \Pi\} \cup \\
& \{X_{c,d}, Y_{c,d} \mid (e,(c,d)) \in \Delta, (c,d) \in \Sigma\} \\
& \{Z_{e,f} \mid u = +(e \Rightarrow f)\} \cup \\
& \{X_{g,h}, Y_{g,h} \mid u = +(e \twoheadrightarrow (g \rightarrow h))\} \\
\Sigma^m \;=\; & \Sigma \setminus \{(g,h) \mid u = +(e \twoheadrightarrow (g \rightarrow h))\} \cup \\
& \{(g, X_{g,h}), (X_{g,h}, Y_{g,h}), (Y_{g,h}, h) \mid u = +(e \twoheadrightarrow (g \rightarrow h))\} \cup \\
& \{(b, Z_{a,b}), (Z_{a,b}, a) \mid (a,b) \in \Pi\} \cup \\
& \{(e, Z_{a,b}) \mid (e,(a,b)) \in \Delta, (a,b) \in \Pi\} \cup \\
& \{(c, X_{c,d}), (X_{c,d}, Y_{c,d}), (Y_{c,d}, d), (e, Y_{c,d}) \mid (e,(c,d)) \in \Delta, (c,d) \in \Sigma\} \cup \\
& \{(f, Z_{e,f}) \mid u = +(e \Rightarrow f)\}.
\end{aligned}
$$

Besides the meta-arguments $Z_{a,b}$ of Definition 5.5, and the attacks involving those arguments, the above meta-AF contains meta-arguments $X_{c,d}, Y_{c,d}$ for encoding second order attacks in Δ toward attacks $(c,d) \in \Sigma$. In fact, an attack $e \twoheadrightarrow (a \Rightarrow b)$ in Δ toward a support is encoded as an attack from e toward $Z_{a,b}$ in the meta-AF, while $e \twoheadrightarrow (c \rightarrow d)$ in Δ is encoded as an attack from e toward $Y_{c,d}$ in the

Table 5.7: Cases for which $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$ for $u = -(a \twoheadrightarrow (b \rightarrow c))$.

update $-(a \twoheadrightarrow (b \rightarrow c))$		$L_0(b)$ IN	$L_0(b)$ UNDECIDED	$L_0(b)$ OUT
$L_0(a)$	IN	NA	NA	
	UNDECIDED	NA		pr
	OUT	pr,st	pr	pr,st

Table 5.8: Cases for which $E_0 \in \mathcal{E}_{\mathcal{S}}(u(\mathcal{B}_0))$ for $u = -(a \twoheadrightarrow (b \Rightarrow c))$.

update $-(a \twoheadrightarrow (b \Rightarrow c))$		$L_0(c)$ IN	$L_0(c)$ UNDECIDED	$L_0(c)$ OUT
$L_0(a)$	IN		NA	NA
	UNDECIDED			NA
	OUT	pr,st	pr	pr,st

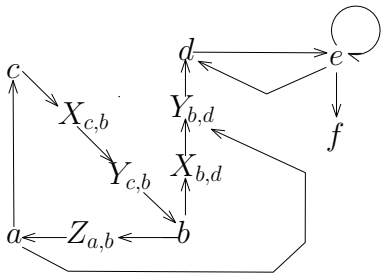


Fig. 5.6: Compact Meta-AF for the EAF $\mathcal{EB}_0$ of Figure 5.4 w.r.t. the update $u = +(d \twoheadrightarrow (c \rightarrow b))$.

meta-AF (which contains also the attacks $(c, X_{c,d}), (X_{c,d}, Y_{c,d}), (Y_{c,d}, d)$). Moreover, meta-arguments $Z_{e,f}$ and $X_{g,h}, Y_{g,h}$, are added to the meta-AF for encoding, respectively, the addition of a second order attack toward a support $(e, f) \in \Pi$ or toward an attack $(g, h) \in \Sigma$. In the latter case, meta-arguments $X_{g,h}$ and $Y_{g,h}$ along with the set of attacks $\{(g, X_{g,h}), (X_{g,h}, Y_{g,h}), (Y_{g,h}, h)\}$ are used to simulate the attack $g \rightarrow h$ which is attacked by e in the EAF. This enables the definition of simple attacks updates (cf. Definition 5.15) to simulate second-order attacks updates.

Example 5.14. The meta-AF for the EAF $\mathcal{EB}_0$ of Figure 5.4 w.r.t. the update $u = +(d \twoheadrightarrow (c \rightarrow b))$ is shown in Figure 5.6. Herein, the attacks involving the meta-arguments $X_{b,d}$ and $Y_{b,d}$ allow us to simulate the second order attack $a \twoheadrightarrow (b \rightarrow d)$. Moreover, the attacks involving the meta-arguments $X_{c,b}$ and $Y_{c,b}$ are added to enable the simulation of the second-order update u by a single attack update on the meta-AF. □

The following definition extends Definitions 5.6 to the case of EAFs and second order updates.

Definition 5.15 (Updates for the Meta-AF). *Let $\mathcal{EB} = \langle A, \Sigma, \Pi, \Delta \rangle$ be an EAF, and u an update for $\mathcal{EB}$. Update u^m for the meta-AF $\mathcal{CM}(\mathcal{EB}, u) = \langle A^m, \Sigma^m \rangle$ is as follows:*

$$u^m = \begin{cases} +(Z_{e,f} \rightarrow e) \ \ if \ u = +(e \Rightarrow f) \\ -(Z_{e,f} \rightarrow e) \ \ if \ u = -(e \Rightarrow f)) \\ +(c \rightarrow d) \ \ if \ u = +(c \rightarrow d) \\ -(c \rightarrow d)) \ \ if \ u = -(c \rightarrow d)) \\ +(e \rightarrow Y_{g,h}) \ if \ u = +(e \twoheadrightarrow (g \rightarrow h)) \\ -(e \rightarrow Y_{g,h}) \ if \ u = -(e \twoheadrightarrow (g \rightarrow h)) \\ +(e \rightarrow Z_{a,b}) \ \ if \ u = +(e \twoheadrightarrow (a \Rightarrow b)) \\ -(e \rightarrow Z_{a,b}) \ \ if \ u = -(e \twoheadrightarrow (a \Rightarrow b)) \end{cases}$$

For instance, continuing with Example 5.14, given the EAF $\mathcal{EB}_0$ of Figure 5.4 and the update $u = +(d \twoheadrightarrow (c \rightarrow b))$, we have that update u^m for the meta-AF $\mathcal{CM}(\mathcal{EB}_0, u)$ shown in Figure 5.6 is $u^m = +(d \rightarrow Y_{c,b})$.

Finally, given an initial extension for an EAF and an update, we define the initial labelling for the corresponding meta-AF as follows.

Definition 5.16 (Initial Labelling for the Meta-AF). *Given an EAF $\mathcal{EB}_0 = \langle A, \Sigma, \Pi, \Delta \rangle$ and a initial labelling L_0, the corresponding initial labelling L_0^m for the meta-AF $\mathcal{CM}(\mathcal{EB}_0, u) = \langle A^m, \Sigma^m \rangle$ is as follows:*

- $\forall a \in A \cap A^m : \ L_0^m(a) = L_0(a)$;
- $\forall \ X_{a,b} \in A^m$:
  - $L_0^m(X_{a,b}) = \text{IN} \ if L_0(a) = \text{OUT}$
  - $L_0^m(X_{a,b}) = \text{OUT} \ if L_0(a) = \text{IN}$
  - $L_0^m(X_{a,b}) = \text{UNDECIDED} \ if L_0(a) = \text{UNDECIDED}$
- $\forall \ Y_{a,b} \in A^m$:
  - $L_0^m(Y_{a,b}) = \text{IN} \ if \ (i) \ L_0^m(X_{a,b}) = \text{OUT} \ and \ \ (ii) \ \forall c \in A \ s.t. \ (c,(a,b)) \in \Delta, \ L_0(c) = \text{OUT}$
  - $L_0^m(Y_{a,b}) = \text{OUT} \ if \ (i) \ L_0^m(X_{a,b}) = \text{IN} \ or \ (ii) \ \exists \ c \in A \mid (c,(a,b)) \in \Delta \ and \ L_0(c) = \text{IN}$
  - $L_0^m(Y_{a,b}) = \text{UNDECIDED}, otherwise.$
- $\forall Z_{a,b} \in A^m$:
  - $L_0^m(Z_{a,b}) = \text{IN} \ if \ (i) \ L_0(b) = \text{OUT} \ and \ \ (ii) \ \forall \ c \in A \ s.t. \ (c,(a,b)) \in \Delta, \ L_0(c) = \text{OUT}$
  - $L_0^m(Z_{a,b}) = \text{OUT} \ if \ (i) \ L_0(b) = \text{IN} \ or \ \ (ii) \ \exists \ c \in A \mid (c,(a,b)) \in \Delta \ and \ L_0(c) = \text{IN}$
  - $L_0^m(Z_{a,b}) = \text{UNDECIDED}, otherwise.$

For instance, given the initial preferred extension $E_0 = \{a, b, d, f\}$ for the EAF $\mathcal{EB}_0$ of Example 5.9, the initial labelling for the meta-AF $\mathcal{CM}(\mathcal{EB}_0, +(d \rightarrow Y_{c,b}))$ of Figure 5.6 is such that $L_0^m(a) = L_0(a) = \text{IN}$, $L_0^m(c) = L_0(c) = \text{OUT}$, $L_0^m(X_{c,b}) = \text{IN}$, and $L_0^m(Y_{c,b}) = \text{OUT}$. Also, we have that $L_0^m(b) = L_0(b) = \text{IN}$, $L_0^m(X_{b,d}) = \text{OUT}$, $L_0^m(Y_{b,d}) = \text{OUT}$ since $L_0^m(a) = L_0(a) = \text{IN}$, and $L_0^m(d) = L_0(d) = \text{IN}$.

5.2.3 Incr-EAF: Incremental Algorithm for EAFs.

We are now ready to define the incremental algorithm for EAFs that we call *Incr-EAF*. In fact, we just need to slightly modify Algorithm 4 as follows.

Incr-EAF takes as input an EAF $\mathcal{EB}_0$, a semantics $\mathcal{S} \in \{\texttt{pr}, \texttt{st}\}$, an extension $E_0 \in \mathcal{E}_\mathcal{S}(\mathcal{EB}_0)$, and an update u of the form $u = \pm(a \Rightarrow b)$, $u = \pm(a \rightarrow b)$, $u = \pm(e \twoheadrightarrow (c \Rightarrow d))$, or $u = \pm(e \twoheadrightarrow (c \rightarrow d))$. To incrementally compute an extension E of the updated EAF $u(\mathcal{EB}_0)$, it suffices to

i) use Proposition 5.12 instead of Proposition 5.4 at Line 1 of Algorithm 4 where function $checkProp$ is called.
ii) compute the (meta) AF $\mathcal{M}_0$, the update u^m for $\mathcal{M}_0$, and the initial $\mathcal{S}$-extension E_0^m for $\mathcal{M}_0$ using the new Definitions 5.13, 5.15, and 5.16 for EAFs, respectively, at Lines 3, 4, and 5.

It is worth noting that if the input EAF of *Incr-EAF* is a BAF, and the update does not concern second-order attacks, then *Incr-EAF* coincides with *Incr-BAF* given in Algorithm 4. In fact, in this case, Proposition 5.12 collapses to Proposition 5.4 as no second order updates are considered, and the meta-AF of Definition 5.13, as well as update u^m of Definition 5.15 and the initial labelling of Definition 5.16, coincide with their counterparts for BAFs introduced in Definitions 5.5, 5.6, and 5.7, respectively.

5.3 Empirical Evaluation

We implemented a C++ prototype and, for each semantics $\mathcal{S} \in \{\texttt{pr}, \texttt{st}\}$, we compared the performance of the incremental algorithm proposed with that of the best ICCMA'17 solver for the computational task $\mathcal{S}$-SE, that is the task of determining some $\mathcal{S}$-extension. More in detail, given an EAF $\mathcal{EB}_0$, a semantics $\mathcal{S} \in \{\texttt{pr}, \texttt{st}\}$, an extension $E_0 \in \mathcal{E}_\mathcal{S}(\mathcal{EB}_0)$, and an update u for the EAF, we compared the following two strategies for computing an extension $E \in \mathcal{E}_\mathcal{S}(u(\mathcal{EB}_0))$ of the updated EAF:

- **Incremental computation**, that is, algorithm *Incr-EAF* with input $\mathcal{EB}_0, u, E_0, \mathcal{S}$, and $\mathsf{Solver}_\mathcal{S}$;
- **Computation from scratch**, where an extension E of the updated EAF $u(\mathcal{EB}_0)$ is computed by running $\mathsf{Solver}_\mathcal{S}$ over the updated (compact) meta-AF.

where $\mathsf{Solver}_\mathcal{S}$ is *ArgSemSAT* [55] for $\mathcal{S} = \texttt{pr}$ and *goDIAMOND* [107] for $\mathcal{S} = \texttt{st}$— these solvers are the winners of the ICCMA'17 competition for the computational task $\mathcal{S}$-SE.

Dataset.

We generated a set of BAFs and a set of EAFs by starting from AFs used as benchmarks at ICCMA'17 [1] for the tracks SE-`pr` and SE-`st`. Specifically, we used the datasets named $B1$ and $B2$, both consisting of 50 AFs whose size is as follows:

- $B1$ consists of AFs with a number of arguments $|A| \in [2, 50K]$ and a number of attacks $|\Sigma| \in [1, 1.6M]$.
- $B2$ consists of AFs with $|A| \in [35, 200K]$ and $|\Sigma| \in [73, 4M]$.

Given the AFs in these datasets, we generated BAFs and EAFs by transforming attacks into supports and then adding second-order attacks as follows.

Given a percentage p of supports, and a percentage $s \leq p$ of second order attacks (i.e., attacks to attacks and attacks to supports), for each AF $\mathcal{A} = \langle A, \Sigma \rangle$ in the ICCMA'17 dataset, we generated EAFs $\mathcal{EB}_0 = \langle A, \Sigma', \Pi, \Delta \rangle$ (or BAFs $\mathcal{B}_0 = \langle A, \Sigma', \Pi \rangle$ if $\Delta = \emptyset$) as follows. First, we selected $p \times |\Sigma|$ attacks in Σ in a random way, and converted them into supports. That is, let $\Sigma^r \subseteq \Sigma$ be the set of the chosen $p \times |\Sigma|$ attacks in Σ, for each $(a, b) \in \Sigma^r$, we added a support randomly chosen in $\{(a, b), (b, a)\}$ to Π, and then set $\Sigma' = \Sigma \setminus \Sigma^r$. Next, we selected $s \times |\Sigma|$ supports in Π and $s \times |\Sigma|$ attacks in Σ' in a random way, and for each support/attack (x, y) selected, we added in Δ a second-order attack from a randomly selected argument in A to (x, y). As an example, if $|\Sigma| = 100$ and $p = 20\%$ and $s = 10\%$, then we have that $|\Sigma'| = 80$ and $|\Pi| = 20$, $|\Delta| = 20$ as there will be 10 second-order attacks toward a support in Π plus 10 second-order attacks toward attack in Σ'.

In the following, we use $\mathrm{B}1(p, s)$ and $\mathrm{B}2(p, s)$ to refer to the datasets consisting of EAFs obtained by starting from AFs in $B1$ and $B2$, respectively, and having a percentage p of supports and a percentage s of second order attacks.

Methodology.

For each semantics $\mathcal{S} \in \{\mathtt{pr}, \mathtt{st}\}$ and EAF $\mathcal{EB}_0 = \langle A_0, \Sigma_0, \Pi_0, \Delta_0 \rangle$ (BAF when $\Delta_0 = \emptyset$) in the dataset, we considered every $\mathcal{S}$-extension E_0 of $\mathcal{EB}_0$ as an initial extension. Then, we randomly selected an update u of the following form (only updates of the types 1)–4) were considered for BAFs):

1) $+(a \rightarrow b)$, with $a, b \in A_0$ and $(a, b) \notin \Sigma_0$;
2) $+(a \Rightarrow b)$, with $a, b \in A_0$ and $(a, b) \notin \Pi_0$;
3) $-(a \rightarrow b)$, with $(a, b) \in \Sigma_0$;
4) $-(a \Rightarrow b)$, with $(a, b) \in \Pi_0$;
5) $+(a \twoheadrightarrow (b \rightarrow c))$, with $a \in A_0$, $(b, c) \in \Sigma_0$, and $(a, (b, c)) \notin \Delta_0$;
6) $+(a \twoheadrightarrow (b \Rightarrow c))$, with $a \in A_0$, $(b, c) \in \Pi_0$, and $(a, (b, c)) \notin \Delta_0$;
7) $-(a \twoheadrightarrow (b \rightarrow c))$, with $(a, (b, c)) \in \Delta_0$;
8) $-(a \twoheadrightarrow (b \Rightarrow c))$, with $(a, (b, c)) \in \Delta_0$.

Next, we computed an $\mathcal{S}$-extension E for the updated EAF $u(\mathcal{EB}_0)$ by calling the incremental algorithm *Incr-EAF*. Finally, the average run time of *Incr-EAF* to compute an $\mathcal{S}$-extension was compared with the average run time of *ArgSemSAT* if $\mathcal{S} = \mathtt{pr}$, or *goDIAMOND* if $\mathcal{S} = \mathtt{st}$, to compute an $\mathcal{S}$-extension for $u^m(\mathcal{CM}(\mathcal{EB}_0, u))$ from scratch.

Results.

Figure 5.7 and 5.8 report the average run times (log scale) of the incremental computation (*Incr-EAF*) and the computation from scratch over the datasets $\mathrm{B}1(p, s)$

and $\mathrm{B2}(p, s)$, respectively, for the preferred semantics (left-hand side) and the stable semantics (right-hand side) and percentages $p \in \{10\%, 20\%\}$ of supports and $s \in \{0\%, 10\%\}$ of second-order attacks. Each data point reported in the figures is an average run time over 20 runs. For the sake of readability, in the figures we also showed the lines obtained by LOESS local regression.

From these results, we can draw the following conclusions:

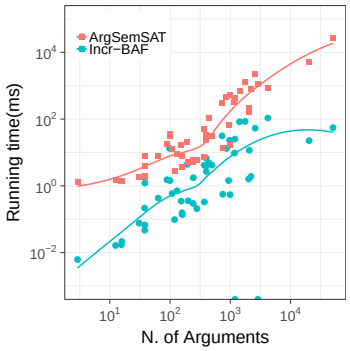
- The incremental algorithm outperforms the competitors that compute extensions from scratch. In fact, on average, the proposed technique is two orders of magnitude faster then the computation from scratch. In particular, the time saved by the incremental computation is higher for the dataset $\mathrm{B2}(p, s)$, where the computation from scratch takes much more time due to the complexer structures of the AFs in $B2$ which are reflected into the generated EAFs. Also, the improvements obtained for the stable semantics are larger than that obtained for the preferred one— this is due to the different external solvers used.
- The improvements obtained for BAFs slightly decrease when increasing the percentage p of supports (see Figures 5.7-5.8 (a-b) and (c-d)). This is due to the fact that the size of the reduced-AF identified by Incr-Alg presented in Section4.2, as well as that of the meta-AF, increases when the number of supports increases. The same behaviour happens when the percentage s of second-order attacks in EAFs increases (see Figures 5.7-5.8 (c-d) and (e-f)). However, worsening in the performance is not dramatic—the incremental technique remains much faster than the computation from scratch in all cases.

5.4 Summary

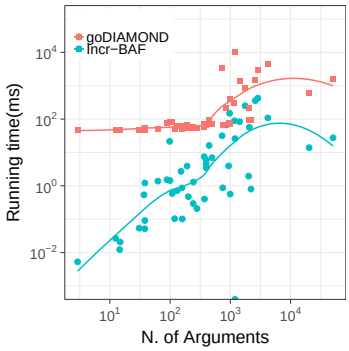
We introduced a technique for the incremental computation of extensions of dynamic EAFs, i.e., BAFs (possibly) incorporating second-order attacks. Following the meta-argumentation approach [58], according to which EAFs are translated into semantically equivalent AFs, we introduced a translation where updates and initial extensions of EAFs are taken into account. Then, we exploited the incremental algorithm proposed in Section 4.2 and computed extensions of the meta-AFs, from which the updated extensions of EAFs are obtained. Experiments showed that the incremental technique outperforms the computation from scratch by two orders of magnitude, on average.

Although in this chapter we focused on updates consisting of adding/removing one attack/support, the technique can be extended to deal with sets of updates performed simultaneously. Indeed, the construction described in Section 2.4 for reducing the application of a set of updates to the application of a single attack update can be easily extended to deal with multiple updates for EAFs. The implementation of such kind of updates for EAFs is left for future work.

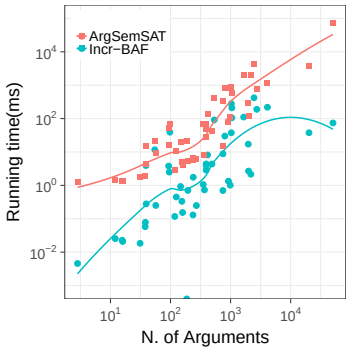
Finally, we envisage the use of the incremental approach in the context of the so called *Argumentation Framework with Recursive Attacks* [16], where also second (and more) order-attacks can be attacked, as well as generalize the algorithm to also



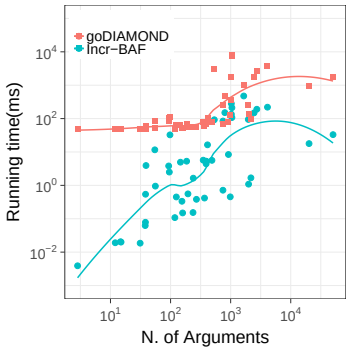
(a) $\mathcal{S} = \texttt{pr},\ p = 10\%,\ s = 0\%$.



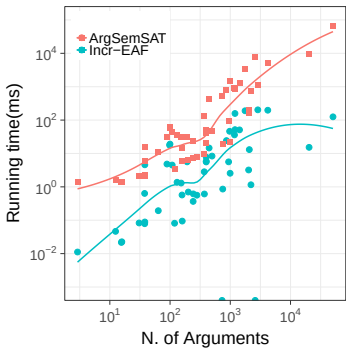
(b) $\mathcal{S} = \texttt{st},\ p = 10\%,\ s = 0\%$.



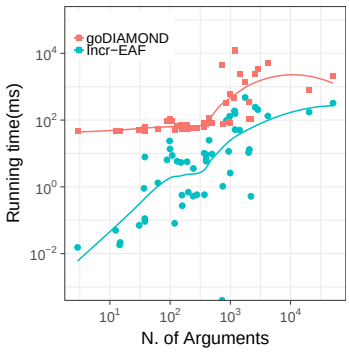
(c) $\mathcal{S} = \texttt{pr},\ p = 20\%,\ s = 0\%$.



(d) $\mathcal{S} = \texttt{st},\ p = 20\%,\ s = 0\%$.

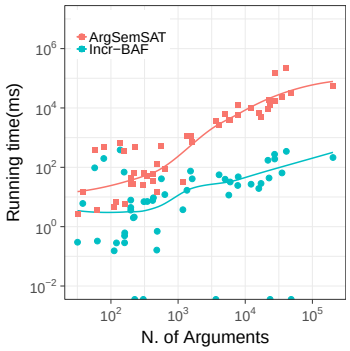


(e) $\mathcal{S} = \texttt{pr},\ p = 20\%,\ s = 10\%$.

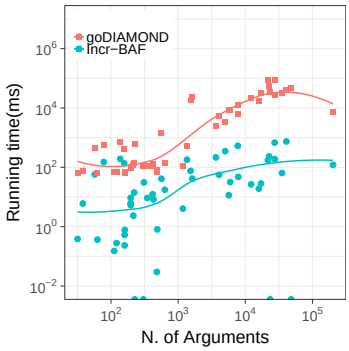


(f) $\mathcal{S} = \texttt{st},\ p = 20\%,\ s = 10\%$.

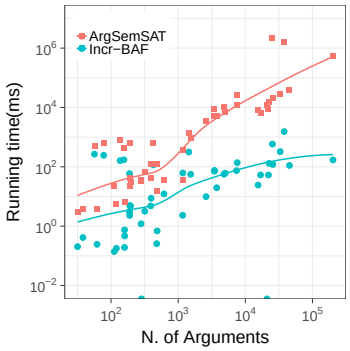
Fig. 5.7: Run times (ms) of ICCMA'17 solvers and *Incr-EAF* over dataset $\mathrm{B1}(p, s)$ versus the number of arguments in the input EAF, for the preferred semantics (left-hand side) and stable semantics (right-hand side). The percentage $p \in \{10\%, 20\%\}$ of supports and the percentage $s \in \{0\%, 10\%\}$ of second-order attacks are shown at the bottom of each graph.



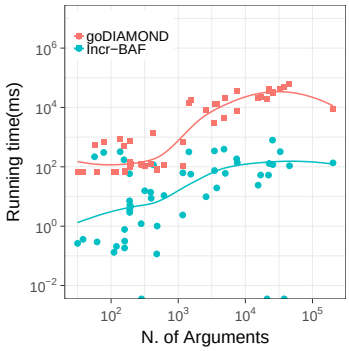
(a) $\mathcal{S} = \texttt{pr},\ p = 10\%,\ s = 0\%$.



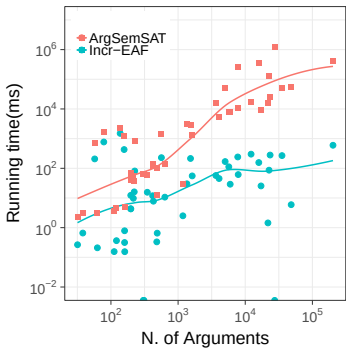
(b) $\mathcal{S} = \texttt{st},\ p = 10\%,\ s = 0\%$.



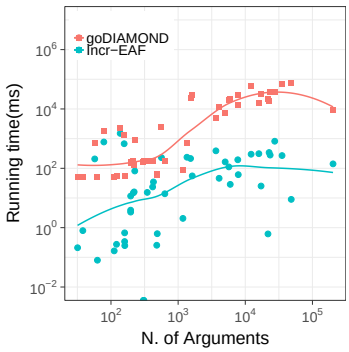
(c) $\mathcal{S} = \texttt{pr},\ p = 20\%,\ s = 0\%$.



(d) $\mathcal{S} = \texttt{st},\ p = 20\%,\ s = 0\%$.



(e) $\mathcal{S} = \texttt{pr},\ p = 20\%,\ s = 10\%$.



(f) $\mathcal{S} = \texttt{st},\ p = 20\%,\ s = 10\%$.

Fig. 5.8: Run times (ms) of ICCMA'17 solvers and *Incr-EAF* over dataset $\mathrm{B2}(p, s)$ versus the number of arguments in the input EAF, for the preferred semantics (left-hand side) and stable semantics (right-hand side). The percentage $p \in \{10\%, 20\%\}$ of supports and the percentage $s \in \{0\%, 10\%\}$ of second-order attacks are shown at the bottom of each graph.

deal with *structured* argumentation [12, 97, 102]. A first step in this direction will be discussed in Chapter 7 where the analysis of the part of DeLP-programs [79]—from which structured arguments are derived—that changes after an update is performed by exploiting the notion of reachability for (hyper-)graphs representing the programs.

6

Efficient Computation of Skeptical Preferred Acceptance in Dynamic Argumentation Frameworks

"The justifications of men who kill should always be heard with skepticism, said the monster."

– Patrick Ness

In this chapter we devise an efficient algorithm for computing the skeptical preferred acceptance in dynamic AFs. More specifically, we investigate how the skeptical acceptance of an argument (goal) evolves when the given AF is updated and propose an efficient algorithm for solving this problem.

The algorithm we propose, called SPA, relies on two main ideas: *i*) computing a small portion of the input AF, called "context-based" AF, which is sufficient to determine the status of the goal in the updated AF, and *ii*) incrementally computing the ideal extension to further restrict the context-based AF.

We experimentally show that SPA significantly outperforms the computation from scratch, and that the overhead of incrementally maintaining the ideal extension pays off as it speeds up the computation.

6.1 Notation for reachability and other useful concepts

We will use as running example of the chapter the AF $\mathtt{AF}_0 = \langle \mathtt{A}_0, \Sigma_0 \rangle$ whose graph is shown on Figure 6.1(a), where $\mathtt{A}_0 = \langle \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{l}\}$ and Σ_0 includes, among others, attacks $(\mathtt{a}, \mathtt{b})$, $(\mathtt{b}, \mathtt{a})$, and $(\mathtt{c}, \mathtt{d})$.

Given an AF $\langle A, \Sigma \rangle$ and arguments $a, b \in A$, we say that a *attacks* b iff $(a, b) \in \Sigma$, and that a set $S \subseteq A$ attacks b iff there is $a \in S$ attacking b. We use $S^+ = \{b \mid \exists a \in S : (a, b) \in \Sigma\}$ to denote the set of arguments attacked by S.

Recalling some preliminaries, it is well-known that every AF admits exactly one ideal extension which is contained in the intersection of the preferred extensions, which are at least one [62]. The preferred extensions of $\mathtt{AF}_0$ are $\mathtt{E}_{\mathtt{pr}} = \{\mathtt{a}, \mathtt{d}, \mathtt{f}, \mathtt{h}, \mathtt{j}, \mathtt{l}\}$ and $\mathtt{E}'_{\mathtt{pr}} = \{\mathtt{b}, \mathtt{d}, \mathtt{f}, \mathtt{h}, \mathtt{k}\}$, while the ideal extension of $\mathtt{AF}_0$ is $E_{id} = \{\mathtt{d}, \mathtt{f}, \mathtt{h}\}$.

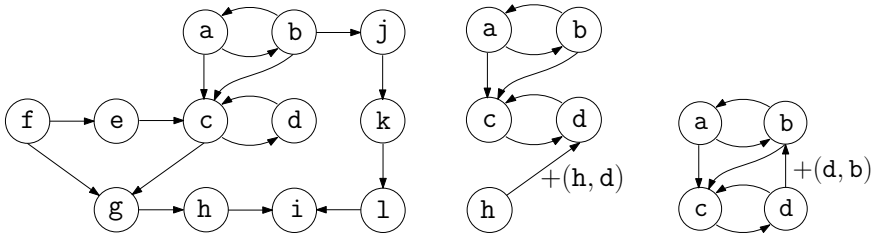


Fig. 6.1: (*a*) AF $\mathtt{AF}_0$, (*b*) context-based AF $\mathcal{R}(+(\mathtt{h},\mathtt{d}), \mathtt{AF}_0, E_{id}, \mathtt{c})$, (*c*) context-based AF $\mathcal{R}(+(\mathtt{d},\mathtt{b}), \mathtt{AF}_0, E_{id}, \mathtt{c})$.

Although the ideal extension of $\mathtt{AF}_0$ coincides with the intersection of the preferred extensions, this is not true in general. For instance, the ideal extension of $\mathtt{AF}_1$ obtained from $\mathtt{AF}_0$ by removing the attack $(\mathtt{d},\mathtt{c})$ is $\{\mathtt{f},\mathtt{h}\}$ which is strictly contained in the intersection of preferred extensions of $\mathtt{AF}_1$, which are the same as the preferred extensions of $\mathtt{AF}_0$.

Given an AF $\mathcal{A} = \langle A, \Sigma\rangle$ and an argument $g \in A$, we say that g is *skeptically accepted* w.r.t. $\mathcal{A}$ under the preferred semantics iff for each preferred extension E of $\mathcal{A}$ it holds that $g \in E$. In the following, we use $SA_{\mathcal{A}}(g)$ to denote the *skeptical acceptance* (either *true* or *false*) of g w.r.t. AF $\mathcal{A}$.

Example 6.1. For AF $\mathtt{AF}_0$ of our running example, we have that the arguments skeptically accepted are $\mathtt{d}$, $\mathtt{f}$, and $\mathtt{h}$. Thus, $SA_{\mathtt{AF}_0}(\mathtt{d})$ is *true*, and so is for $SA_{\mathtt{AF}_0}(\mathtt{f})$ and $SA_{\mathtt{AF}_0}(\mathtt{h})$, while for any other argument x, $SA_{\mathtt{AF}_0}(x)$ =*false*. □

Since every preferred extension contains the ideal extension, the following fact follows.

Fact 1 *Let $\mathcal{A}$ be an AF, E the ideal extension of $\mathcal{A}$, and g an argument of $\mathcal{A}$. If $g \in E$ then $SA_{\mathcal{A}}(g)$ =true. On the other hand, if $g \in E^+$ then $SA_{\mathcal{A}}(g)$ =false.*

As done in previous chapters, we use $+(a,b)$, with $a,b \in A_0$ and $(a,b) \notin \Sigma_0$, (resp. $-(a,b)$, with $(a,b) \in \Sigma_0$) to denote the addition (resp. deletion) of an attack (a,b), and $u(\mathcal{A}_0)$ to denote the application of update $u = \pm(a,b)$ to AF $\mathcal{A}_0$ (where $\pm$ means either $+$ or $-$). Applying an update u to an AF $\mathcal{A}_0$ implies that the extensions prescribed by a given semantics, as well as set of arguments that are skeptically accepted, may change.

Example 6.2. Continuing with our running example, let $u = +(\mathtt{h},\mathtt{d})$. The ideal extension of $u(\mathtt{AF}_0)$ is $\{\mathtt{f},\mathtt{h}\}$, while the preferred extensions are $\{\mathtt{a},\mathtt{f},\mathtt{h},\mathtt{j},\mathtt{l}\}$ and $\{\mathtt{b},\mathtt{f},\mathtt{h},\mathtt{k}\}$. Thus, only $\mathtt{f}$ and $\mathtt{h}$ are skeptically accepted w.r.t. $u(\mathtt{AF}_0)$. □

As for the addition (resp. deletion) of a set of *isolated* arguments (i.e., arguments not adjacent to any other argument in the graph), it is easy to see that if $\mathcal{A}$ is obtained from $\mathcal{A}_0$ through the addition (resp. deletion) of a set S of isolated argument, then

every argument in S is trivially skeptically accepted (resp., not accepted) w.r.t. $\mathcal{A}$. Indeed, if E_0 is an extension for $\mathcal{A}_0$, then $E = E_0 \cup S$ (resp. $E = E_0 \setminus S$) is an extension for $\mathcal{A}$ containing every (resp., none) argument in S. Of course, if arguments in S are not isolated, for addition we can first add isolated arguments and then add attacks involving these arguments, while for deletion we can first delete all attacks involving arguments in S. Thus we do not consider these kinds of updates in the following, and focus on the addition and deletions of attacks.

Given an AF $\mathcal{A} = \langle A, \Sigma \rangle$ and an argument x, we use $Reach_{\mathcal{A}}(x)$ to denote the set of arguments that are reachable from x in the graph $\mathcal{A}$. Moreover, we use $Reach_{\mathcal{A}}^{-1}(x)$ to denote the set of arguments from which x is reachable in $\mathcal{A}$. For instance, for the AF $\mathtt{AF}_0 = \langle \mathtt{A}_0, \Sigma_0 \rangle$ of our running example (see Figure 6.1(a)), we have that $Reach_{\mathtt{AF}_0}(\mathtt{d}) = \{\mathtt{d}, \mathtt{c}, \mathtt{g}, \mathtt{h}, \mathtt{i}\}$, and $Reach_{\mathtt{AF}_0}^{-1}(\mathtt{h}) = \mathtt{A}_0 \setminus \{\mathtt{i}, \mathtt{j}, \mathtt{k}, \mathtt{l}\}$. We write $Reach_{\mathcal{A}}(x) = \emptyset$ and $Reach_{\mathcal{A}}^{-1}(x) = \emptyset$ if x is not in $\mathcal{A}$.

We use $H(\mathcal{A}, u)$ to denote the larger AF between $\mathcal{A}$ and $u(\mathcal{A})$, that is, $H(\mathcal{A}, u)$ is (*i*) the updated AF $u(\mathcal{A})$ if u is an addition update (it includes the attack added through u), (*ii*) the original AF $\mathcal{A}$ if u is a deletion (the removed attack is still considered in $H(\mathcal{A}, u)$). For instance, if $u = +(\mathtt{h}, \mathtt{d})$ then $H(\mathtt{AF}_0, u) = \langle \mathtt{A}_0, \Sigma_0 \cup \{(\mathtt{h}, \mathtt{d})\} \rangle$, while $H(\mathtt{AF}_0, u) = \mathtt{AF}_0$ for any deletion update u.

We use $\Pi(S, \mathcal{A})$ to denote the *restriction* of AF $\mathcal{A} = \langle A, \Sigma \rangle$ to a subset $S \subseteq A$ of its arguments [18], that is $\Pi(S, \mathcal{A}) = \langle S,\ \Sigma \cap (S \times S) \rangle$. For instance, if $S = \{\mathtt{c}, \mathtt{d}\}$ then $\Pi(S, \mathtt{AF}_0) = \langle \{\mathtt{c}, \mathtt{d}\}, \{(\mathtt{c}, \mathtt{d}), (\mathtt{d}, \mathtt{c})\} \rangle$.

Finally, given $\mathcal{A}_1 = \langle A_1, \Sigma_1 \rangle$ and $\mathcal{A}_2 = \langle A_2, \Sigma_2 \rangle$, we denote as $\mathcal{A}_1 \sqcup \mathcal{A}_2 = \langle A_1 \cup A_2, \Sigma_1 \cup \Sigma_2 \rangle$ the *union* of the two AFs.

6.2 Supporting Set

In this section, we introduce the novel concept of *supporting set* which intuitively consists of the set of arguments that needs to be taken into account in order to determine the skeptical acceptance of an argument of interest after performing an update. We provide a parametric definition of supporting set that will enable the characterization of different portions of a given AF, called *context-based AFs*, that will be used for two different purposes: (*i*) recompute the skeptical acceptance of a goal w.r.t. the updated AF, and (*ii*) recompute the ideal extension of the updated AF.

Before defining the supporting set, we introduce the auxiliary notion of *steadiness* of an argument. Given an AF $\mathcal{A} = \langle A, \Sigma \rangle$, the ideal extension E of $\mathcal{A}$, and an update $u = \pm(a, b)$, we first define $E(u)$ as the subset of E consisting of the arguments which are not reachable from b in $\mathcal{A}$, i.e., $E(u) = \{z \mid z \in E, z \notin Reach_{\mathcal{A}}(b)\}$. Intuitively, the acceptance status of the arguments in $E(u)$ is not affected by u as they are not reachable from it. Then, the set of *steady* arguments for $u = \pm(a, b)$ w.r.t. $\mathcal{A}$ is defined as $Std_{\mathcal{A}}(u) = (E(u))^+ \setminus \{b\}$, i.e., the arguments attacked by $E(u)$ in $\mathcal{A}$ and that will be still attacked by $E(u)$ in $u(\mathcal{A})$. Argument b is not included in $Std_{\mathcal{A}}(u)$ as it may be no longer attacked by $a \in E(u)$ after performing $u = -(a, b)$; however, it will be considered for positive updates in Definition 6.3. For the AF $\mathtt{AF}_0$

of our running example, where $E_{id} = \{\mathtt{d}, \mathtt{f}, \mathtt{h}\}$, if $u = +(\mathtt{h}, \mathtt{d})$ then $E_{id}(u) = \{\mathtt{f}\}$ and $Std_{\mathtt{AF}_0}(u) = \{\mathtt{e}, \mathtt{g}\} \subseteq E^+_{id} = \{\mathtt{c}, \mathtt{e}, \mathtt{g}, \mathtt{i}\}$.

Thus, if an argument is steady for update u w.r.t. AF $\mathcal{A}$ then its acceptance status does not depend on u, even in the case that it is reachable from an argument of u. Steady arguments limit the portion of the AF to be examined to define the supporting set.

Definition 6.3 (Supporting set). *Let $\mathcal{A} = \langle A, \Sigma \rangle$ be an AF, $u = \pm(a, b)$ an update, E the ideal extension of $\mathcal{A}$, and g an argument in A. Let*

$$- \; Sup_0(u, \mathcal{A}, E, g) = \begin{cases} \emptyset & \text{if } u = +(a,b) \wedge b \in (E(u))^+; \\ \emptyset & \text{if } b \notin Reach^{-1}_{H(\mathcal{A},u)}(g); \\ \{b\} & otherwise. \end{cases}$$

$$- \; Sup_{i+1}(u, \mathcal{A}, E, g) = Sup_i(u, \mathcal{A}, E, g) \cup \{y \mid \exists (x, y) \in \Sigma \; s.t. \; x \in Sup_i(u, \mathcal{A}, E, g) \; \wedge y \in Reach^{-1}_{H(\mathcal{A},u)}(g) \wedge y \notin Std_{\mathcal{A}}(u)\}.$$

Let n be the natural number such that $Sup_n(u, \mathcal{A}, E, g) = Sup_{n+1}(u, \mathcal{A}, E, g)$. The supporting set $Sup(u, \mathcal{A}, E, g)$ *is:*

$$Sup(u, \mathcal{A}, E, g) = Sup_n(u, \mathcal{A}, E, g) \cap Reach^{-1}_G(g) \tag{6.1}$$

where $G = \Pi(Sup_n(u, \mathcal{A}, E, g), H(\mathcal{A}, u))$ is the restriction of $H(\mathcal{A}, u)$ to $Sup_n(u, \mathcal{A}, E, g)$.

Finally, when g is not specified, the supporting set, denoted as $Sup(u, \mathcal{A}, E, \star)$, is defined as $Sup(u, \mathcal{A}, E, g)$ except that all the checks concerning $Reach^{-1}$ are omitted.

Intuitively, $Sup(u, \mathcal{A}, E, g)$ consists of the arguments whose status may change after performing an update u and such that their change can imply a change of the status of g.

The supporting set is iteratively defined by $n + 1$ steps ($n \leq |A|$), each of them consisting of the addition of at least a non-steady argument attacked by the set built at the previous step and allowing to reach the goal g (if specified). More in detail, $Sup(u, \mathcal{A}, E, g)$ for $u = \pm(a, b)$ and g consists of the arguments that (*i*) can be reached from b without using any steady argument y; and (*ii*) allow to reach g in $H(\mathcal{A}, u)$ by using only the arguments in $Sup_n(u, \mathcal{A}, E, g)$. In fact, Equation (6.1) entails that an argument of $Sup_n(u, \mathcal{A}, E, g)$ will be in $Sup(u, \mathcal{A}, E, g)$ only if it can reach g in the restriction of $H(\mathcal{A}, u)$ to $Sup_n(u, \mathcal{A}, E, g)$—the other arguments in $Sup_n(u, \mathcal{A}, E, g)$ are not needed to determine the acceptance status of g, and thus they are pruned by Equation (6.1).

When no argument g is specified, the set $Sup(u, \mathcal{A}, E, \star)$ is built by ignoring condition (*ii*) above. It is easy to see that, for any argument g, $Sup(u, \mathcal{A}, E, g) \subseteq Sup(u, \mathcal{A}, E, \star) \subseteq Reach_{\mathcal{A}}(b)$, where b is the argument in the update $u = \pm(a, b)$. Moreover, as shown in the following example, $Sup(u, \mathcal{A}, E, g)$ may be empty even if $g \in Reach_{\mathcal{A}}(b)$. Finally, if $Sup(u, \mathcal{A}, E, g) \neq \emptyset$ then the arguments of at least one path from b to g belong to $Sup(u, \mathcal{A}, E, g)$.

Example 6.4 (Supporting set for $u = +(\mathtt{h}, \mathtt{d})$). For the goal $\mathtt{c}$, we have that $Sup_0(u, \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{d}\}$, $Sup_1(u, \mathtt{AF}_0, E_{id}, c) = \{\mathtt{c}, \mathtt{d}\}$, and $Sup_2(u, \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{c}, \mathtt{d}\}$ (the latter does not contain g since $\mathtt{g} \in Std_{\mathtt{AF}_0}(u)$). Thus, $Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{c}, \mathtt{d}\}$ as both $\mathtt{c}$ and $\mathtt{d}$ allow to reach $\mathtt{c}$ in the restriction of the updated AF to $\{\mathtt{c}, \mathtt{d}\}$. Reasoning analogously, we have that $Sup(u, \mathtt{AF}_0, E_{id}, \star) = \{\mathtt{c}, \mathtt{d}\}$.

Consider now what happen for the goal $\mathtt{h}$. Again $Sup_0(u, \mathtt{AF}_0, E_{id}, \mathtt{h}) = \{\mathtt{d}\}$, and $Sup_1(u, \mathtt{AF}_0, E_{id}, \mathtt{h}) = Sup_2(u, \mathtt{AF}_0, E_{id}, \mathtt{h}) = \{\mathtt{c}, \mathtt{d}\}$. However, $Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{h}) = \emptyset$ as $\{\mathtt{c}, \mathtt{d}\} \cap Reach_G^{-1}(\mathtt{h}) = \emptyset$, where $G = \Pi(\{\mathtt{c}, \mathtt{d}\}, u(\mathtt{AF}_0))$.

Also for the goal $\mathtt{a}$, we have that $Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{a}) = \emptyset$. □

Example 6.5 (Supporting set for $u' = +(\mathtt{d}, \mathtt{b})$). For the goal $\mathtt{c}$, if $u' = +(\mathtt{d}, \mathtt{b})$ then $Sup_0(u', \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{b}\}$, $Sup_1(u', \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, $Sup_2(u', \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\} = Sup_3(u', \mathtt{AF}_0, E_{id}, \mathtt{c})$ (the latter does not contain $\mathtt{j}$ since $\mathtt{j} \notin Reach^{-1}_{H(\mathtt{AF}_0, u')}(\mathtt{c})$). Thus, $Sup(u', \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, while $Sup(u', \mathtt{AF}_0, E_{id}, \star) = \mathtt{A}_0 \setminus \{\mathtt{f}, \mathtt{e}, \mathtt{g}, \mathtt{h}\}$. □

As stated next, the skeptical acceptance of an argument does not change if the supporting set for it is empty.

Theorem 6.6. *Let $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ be an AF, E_0 the ideal extension of $\mathcal{A}_0$, $u = \pm(a, b)$ an update, $\mathcal{A} = u(\mathcal{A}_0)$ the updated AF, and x an argument in A_0. Therefore, if $Sup(u, \mathcal{A}_0, E_0, x) = \emptyset$ then $SA_{\mathcal{A}}(x) = SA_{\mathcal{A}_0}(x)$.*

Proof. We consider the cases for which $Sup(u, \mathcal{A}_0, E_0, x)$ turns out to be empty. $Sup(u, \mathcal{A}_0, E_0, x) = \emptyset$ if either (*i*) $Sup_0(u, \mathcal{A}_0, E_0, x) = \emptyset$ or (*ii*) $Sup_n(u, \mathcal{A}_0, E_0, x) \cap Reach_G^{-1}(x) = \emptyset$, where $G = \Pi(Sup_n(u, \mathcal{A}_0, E_0, x), H(\mathcal{A}_0, u))$.

In particular, case (*i*) can occur if either (*i.1*) $b \in Std_{\mathcal{A}}(u)$ or (*i.2*) $b \notin Reach^{-1}_{H(\mathcal{A}, u)}(x)$.

We separately deal with each the aforementioned cases.

(*i.1*) If $b \in Std_{\mathcal{A}}(u)$ then there is an argument $z \in A_0$ such that $z \neq a$, $z \in E_0$, $(z, b) \in \Sigma_0$, and z is not reachable from b in $\mathcal{A}$. This implies that z is IN in every preferred extension of $\mathcal{A}_0$ as well as in every preferred extension of $u(\mathcal{A}_0)$. Thus b is OUT in every preferred extension of $\mathcal{A}_0$ and in every preferred extension of $u(\mathcal{A}_0)$. Therefore, update u does not change the status of b, and consequently it does not change the status of every other argument, as they are reachable from b only using b. It follows that x is skeptically accepted w.r.t. $\mathcal{A}$ iff it is skeptically accepted w.r.t. $\mathcal{A}_0$.

(*i.2*) If $b \notin Reach^{-1}_{H(\mathcal{A}, u)}(x)$ then the status of x w.r.t. any preferred extension of $u(\mathcal{A}_0)$ does not depend on the status of b, from which the statement follows.

(*ii*) If $Sup_n(u, \mathcal{A}_0, E_0, x) \cap Reach_G^{-1}(x) = \emptyset$, then there is no path in the updated AF from argument b to x using the arguments in $Sup_n(u, \mathcal{A}_0, E_0, x)$, which are all the arguments y such that if the status of y changes then the status of the goal x may change. The latter follows by induction on the steps i of the construction of Sup_i (with $i \in [0..n]$) and using for the base case $i = 0$ and the inductive step what is said above for (*i*). Thus, x is not reachable by any argument y which can change status after the update, which implies that the status of x does not change as well, from which the statement follows. □

Example 6.7. Continuing with Example 6.4, since $Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{h}) = \emptyset$ we can conclude that $SA_{u(\mathtt{AF}_0)}(\mathtt{h}) = SA_{\mathtt{AF}_0}(\mathtt{h}) =$*true*. Similarly, since $Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{a}) = \emptyset$ then $SA_{u(\mathtt{AF}_0)}(\mathtt{a}) = SA_{\mathtt{AF}_0}(\mathtt{a}) =$*false*. □

In the next section we address the problem of computing the skeptical acceptance of arguments whose supporting set is not empty, such as argument c of our running example.

6.3 Context-Based Argumentation Frameworks

The supporting set has been used so far to determine whether the status of the goal does not need to be recomputed. In this section, starting from the supporting set, we define a restriction of the AF which will be used to compute the status of the goal after an update. More specifically, given the supporting set $Sup(u, \mathcal{A}, E, g)$ (resp. $Sup(u, \mathcal{A}, E, \star)$), we define the *context-based AF* $\mathcal{R}(u, \mathcal{A}, E, g)$ (resp. $\mathcal{R}(u, \mathcal{A}, E, \star)$). Moreover, while $\mathsf{CBAF}(u, \mathcal{A}, E, \star)$ will be used to incrementally compute the ideal extension of the updated AF (with the aim of checking if one of the conditions of Fact 1 holds), $\mathsf{CBAF}(u, \mathcal{A}, E, g)$ will be used to compute the skeptical acceptance $SA_{u(\mathcal{A})}(g)$ w.r.t. the updated AF.

Given an AF $\mathcal{A} = \langle A, \Sigma \rangle$, its ideal extension E, and a set $S \subseteq A$, we use $Nodes(\mathcal{A}, S, E)$ to denote the set of the nodes $x \in A$ such that there are a node $y \in S$ and a path from x to y in $\mathcal{A}$ such that all nodes except y do not belong to $E \cup E^+$ (i.e., they are *undecided*, using the *labelling* terminology [14]). Analogously, $Edges(\mathcal{A}, S, E)$ is the set of edges $(x, z) \in \Sigma$ such that there are $y \in S$ and a path from x to y in $\mathcal{A}$ containing (x, z) such that all nodes except y do not belong to $E \cup E^+$. Essentially, if S is the supporting set, to determine the status of nodes in S we must also consider all nodes and attacks occurring in paths (of any length) ending in S whose nodes outside S are undecided. The motivation to also consider "undecided" paths is that some of the undecided arguments occurring in such paths could belong to (or be attacked by) some preferred extension and, therefore, together they could determine a change in the status of nodes in S.

Definition 6.8 (Context-Based AF). *Let $\mathcal{A} = \langle A, \Sigma \rangle$ be an AF, $u = \pm(a, b)$, E the ideal extension of $\mathcal{A}$, and x either an argument in A or the symbol $\star$. Let $S = Sup(u, \mathcal{A}, E, x)$. The context-based AF of $\mathcal{A}$ w.r.t. u and x is $\mathsf{CBAF}(u, \mathcal{A}, E, x) = \Pi(Sup(u, \mathcal{A}, E, x), u(\mathcal{A})) \sqcup T_1 \sqcup T_2$ where:*

- *T_1 is the union of the AFs $\langle \{c, d\}, \{(c, d)\} \rangle$ s.t. (c, d) is an attack of $u(\mathcal{A})$ and $c \notin Sup(u, \mathcal{A}, E, x)$, $c \in E$, and $d \in Sup(u, \mathcal{A}, E, x)$;*
- *$T_2 = \langle Nodes(u(\mathcal{A}), S, E),\ Edges(u(\mathcal{A}), S, E) \rangle$.*

Example 6.9. For $\mathtt{AF}_0$, where $E_{id} = \{\mathtt{d}, \mathtt{f}, \mathtt{h}\}$, and $u = +(\mathtt{h}, \mathtt{d})$, we have seen in Example 6.4 that $Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{c}, \mathtt{d}\}$. Thus $\mathsf{CBAF}(u, \mathtt{AF}_0, E_{id}, \mathtt{c}) = \langle \{\mathtt{c}, \mathtt{d}\}, \{(\mathtt{c}, \mathtt{d}), (\mathtt{d}, \mathtt{c})\} \rangle \sqcup T_1 \sqcup T_2$ where: $T_1 = \langle \{\mathtt{h}, \mathtt{d}\}, \{(\mathtt{h}, \mathtt{d})\} \rangle$ since $\mathtt{h} \in E_{id}$ does not belong to $Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{c})$ while $\mathtt{d} \in Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{c})$; and

$T_2 = \langle\{\mathtt{a},\mathtt{b},\mathtt{c}\},\{(\mathtt{a},\mathtt{b}),\ (\mathtt{b},\mathtt{a}),(\mathtt{a},\mathtt{c}),(\mathtt{b},\mathtt{c})\}\rangle$ since there are paths starting from the undecided arguments a and b ($\{\mathtt{a},\mathtt{b}\} \not\subseteq (E_{id} \cup E_{id}^{+})$) and ending in $\mathtt{c} \in Sup(u, \mathtt{AF}_0, E_{id}, \mathtt{c})$. Thus, $\mathsf{CBAF}(u, \mathtt{AF}_0, E_{id}, \mathtt{c})$ is the AF shown in Figure 6.1(b).

Also, $\mathsf{CBAF}(u, \mathtt{AF}_0, E_{id}, \star) = \mathsf{CBAF}(u, \mathtt{AF}_0, E_{id}, \mathtt{c})$. □

In general, $\mathsf{CBAF}(u, \mathcal{A}, E, g)$ is a subgraph of $\mathsf{CBAF}(u, \mathcal{A}, E, \star)$ since $Sup(u, \mathcal{A}, E, g) \subseteq Sup(u, \mathcal{A}, E, \star)$.

Example 6.10. For $\mathtt{AF}_0$, update $u' = +(\mathtt{d},\mathtt{b})$ of Example 6.5, and goal c, $\mathsf{CBAF}(u', \mathtt{AF}_0, E_{id}, \mathtt{c})$ is as shown in Figure 6.1(c), since $Sup(u', \mathtt{AF}_0, E_{id}, \mathtt{c}) = \{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\}$ and both T_1 and T_2 are empty. However, $\mathsf{CBAF}(u', \mathtt{AF}_0, E_{id}, \star)$ is bigger, as it is the restriction of the updated AF to $Sup(u', \mathtt{AF}_0, E_{id}, \star) = \mathtt{A}_0 \setminus \{\mathtt{f},\mathtt{e},\mathtt{g},\mathtt{h}\}$ union $T_1 = \langle\{\mathtt{h},\mathtt{i}\},\{(\mathtt{h},\mathtt{i})\}\rangle$ (T_2 is empty). □

The following theorem states a result which is complementary to that of Theorem 6.6 as it considers the skeptical acceptance of arguments that belong to the supporting set.

Theorem 6.11. *Let $\mathcal{A}_0 = \langle A_0, \Sigma_0\rangle$ be an AF, E_0 the ideal extension of $\mathcal{A}_0$, $u = \pm(a,b)$ an update, $\mathcal{A} = u(\mathcal{A}_0)$ the updated AF, and x an argument in A_0. Thus, if $Sup(u, \mathcal{A}_0, E_0, x) \neq \emptyset$ then x is skeptically accepted w.r.t. $\mathcal{A}$ iff it is skeptically accepted w.r.t. the context-based AF $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, x)$.*

Proof. The statement follows from the fact that the set of preferred extensions of the context-based AF $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, x)$ coincides with the restriction of the set of the preferred extension $\mathcal{E}_{\mathtt{pr}}(\mathcal{A})$ for the updated AF $\mathcal{A}$ to the arguments in $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, x)$. That is, let $\mathcal{A}_c = \langle A_c, \Sigma_c\rangle = \mathsf{CBAF}(u, \mathcal{A}_0, E_0, x)$, it is the case that $\mathcal{E}_{\mathtt{pr}}(\mathcal{A}_c) = \{E \cap A_c \mid E \in \mathcal{E}_{\mathtt{pr}}(\mathcal{A})\}$, which is proved in what follow.

Let L_0 and L be the labelling corresponding to E_0 and E, respectively, where E is a preferred extension of the updated AF. Moreover, let L_c be a preferred labelling for the context-based AF $\mathcal{A}_c$.

Let F be the set of arguments belonging to $\mathcal{A}$ but not to $\mathcal{A}_c$ and attacking one argument in $\mathcal{A}_c$. That is, let $\mathcal{A} = \langle A, \Sigma\rangle$, $F = \{y \mid y \in A, y \notin A_c, \exists z \in A_c \ s.t. (y,z) \in \Sigma\}$.

Then, the way the context-based AF is defined entails that the status of every argument in F is OUT w.r.t. the ideal extension E_0, that is, $\forall z \in F, L_0(z) = \text{OUT}$. Indeed, if one of such arguments z was in E_0 (i.e., $L_0(z) = \text{IN}$) it would belong to the component T_1 of the definition of context-based AF. On the other hand, if it was UNDECIDED then it would belong to T_2, as well as all the UNDECIDED arguments on paths (of any length) ending in that argument. Moreover, the status of every argument in F does not depend on the update, that is $\forall z \in F, L(z) = L_0(z) = \text{OUT}$; otherwise, z would be in $Sup(u, \mathcal{A}_0, E_0, x)$ since x can be reached by any argument in F, and the arguments from which x can be reached does not belong to $Sup(u, \mathcal{A}_0, E_0, x)$ only if they are not reachable from b or they can be reached from b only using steady arguments.

Therefore, for each argument z in F^{+} w.r.t. $\mathcal{A}$, one of the following cases holds.

1) $z^{-} \subseteq F$ (i.e., all the attackers of z are in F) and $L_0(z) = \text{IN}$. Thus, $z \in E_0$ and it is attacked only by arguments in F which are OUT and whose status cannot change

after the update, meaning that $z \in E$, that is $L(z) = L_0(z) = \text{IN}$. By definition of context-based AF, z belongs to the component T_1 of $\mathcal{A}_c$, it is not attacked by any argument in $\mathcal{A}_c$, and its status will be IN for any preferred labelling L_c of $\mathcal{A}_c$.

2) $z^- \subseteq F$ and $L_0(z) = \text{UNDECIDED}$. Thus z is UNDECIDED w.r.t. L_0 and it is attacked only by arguments in F which are OUT and whose status cannot change after the update, meaning that $L(z)$ may be different from $L_0(z)$ but it does not depend on F. By definition of context-based AF, z belongs to the component T_2 of $\mathcal{A}_c$, and its status w.r.t. L and L_c depends only on the status of other arguments in $\mathcal{A}_c$.

3) if $z^- \subseteq F \cup A_c$ (i.e., the attacks to z comes from both F and the arguments of $\mathcal{A}_c$) then the status of z w.r.t. L depends only on the arguments in A_c. Indeed, if $L(z) = \text{OUT}$ then there must be an attacker in $w \in z^-$ such that $L(w) = \text{IN}$ (w cannot belong to F which contains only arguments which are OUT). Similarly, if $L(z) = \text{IN}$ then the status is determined by all the attackers in $w \in z^- \cap A_c$ that must be such that $L(w) = \text{OUT}$. Finally, if $L(z) = \text{UNDECIDED}$ then there must be an attacker in $w \in z^- \cap A_c$ such that $L(w) = \text{UNDECIDED}$.

Therefore, for any extension $E \in \mathcal{E}_{\mathtt{pr}}(\mathcal{A})$, the status of arguments in A_c w.r.t. E either can be decided to be IN (Case 1 above) or it does not depend on the status of arguments in $A \setminus A_c$ (Cases 2 and 3 above). This in turns implies that $\mathcal{E}_{\mathtt{pr}}(\mathcal{A}_c) = \{E \cap A_c \mid E \in \mathcal{E}_{\mathtt{pr}}(\mathcal{A})\}$, from which it follows that the skeptical acceptance of $x \in A_c$ w.r.t. $\mathcal{A}$ is as the skeptical acceptance of x w.r.t. $\mathcal{A}_c$. □

Example 6.12. Continuing from Example 6.9, we can conclude that argument c is not skeptically accepted w.r.t. the updated AF $u(\mathtt{AF}_0)$ because it is not skeptically accepted w.r.t. the context-based AF $\mathsf{CBAF}(u, \mathtt{AF}_0, E_{id}, \mathtt{c})$ of Figure 6.1(b) whose preferred extensions are $\{\mathtt{a}, \mathtt{h}\}$ and $\{\mathtt{b}, \mathtt{h}\}$ (only h is skeptically accepted w.r.t. the context-based AF). □

Finally, the ideal extension of the updated AF can be obtained as the union of the ideal extension of the context-based AF and the projection of the initial ideal extension on the complement of the supporting set.

Theorem 6.13. *Let $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$ be an AF, E_0 the ideal extension of $\mathcal{A}_0$, $u = \pm(a, b)$ an update, and $\mathcal{A} = u(\mathcal{A}_0)$ the updated AF. Then, the ideal extension E of $\mathcal{A}$ is such that $E = (E_0 \setminus Sup(u, \mathcal{A}_0, E_0, \star)) \cup E'$, where E' is the ideal extension of the context-based AF $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, \star)$.*

Proof. Let L_0 and L be the labelling corresponding to E_0 and E, respectively.

We first show that for each argument $x \in A_0 \setminus Sup(u, \mathcal{A}_0, E_0, \star)$ it holds that $L_0(x) = L(x)$.

Reasoning by contradiction, assume that the status $L(x)$ of argument $x \in A_0 \setminus Sup(u, \mathcal{A}_0, E_0, \star)$ is different from its original status $L_0(x)$ in $\mathcal{A}_0$. We have the following two cases: (*i*) $x \notin Reach_{\mathcal{A}_0}(b)$, which implies that $L_0(x) = L(x)$; (*ii*) $x \in Reach_{\mathcal{A}_0}(b)$ and thus $x \notin Sup(u, \mathcal{A}_0, E_0, \star)$ since x is reachable from b only by paths of the form $q_1, \ldots, q_n, y_1, \ldots, y_k$ where $y_k = x$ and such that exists $z \in E_0(u)$ such that $(z, y_1) \in \Sigma_0$ (the subpath $q_1, \ldots, q_n$ may be empty, meaning that

$y_1 = b$). Since $L_0(z) = \text{IN}$ and $z \notin Reach_{\mathcal{A}_0}(b)$, then $L(z) = L_0(z)$ which in turn means that $L(y_1) = L_0(y_1) = \text{OUT}$ and $L(y_i) = L_0(y_i)$ with $i \in [2..k]$ (note that each y_i is reachable only by path having the above-stated property, otherwise the property would not hold for x either). Therefore, for each argument $x \in A_0 \setminus Sup(u, \mathcal{A}_0, E_0, \star)$, we have that $L_0(x) = L(x)$, which in turn implies that all the arguments in $E_0 \setminus Sup(u, \mathcal{A}_0, E_0, \star)$ belongs to E, i.e., their status does not change after the update.

Reasoning as in the proof of Theorem 6.11, it can be shown that the set of preferred extensions of the context-based AF $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, \star)$ coincides with the restriction of the set of the preferred extension $\mathcal{E}_{\texttt{pr}}(\mathcal{A})$ for the updated AF $\mathcal{A}$ to the arguments in $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, \star)$. That is, let $\mathcal{A}_c = \langle A_c, \Sigma_c \rangle = \mathsf{CBAF}(u, \mathcal{A}_0, E_0, \star)$, it is the case that $\mathcal{E}_{\texttt{pr}}(\mathcal{A}_c) = \{E_p \cap A_c \mid E_p \in \mathcal{E}_{\texttt{pr}}(\mathcal{A})\}$. Moreover, all the arguments of $\mathcal{A}$ attacking those in the context-base AF are OUT w.r.t. the initial ideal labelling L_0, as well as w.r.t. any preferred labelling L_p (i.e., those argument are OUT also w.r.t. the ideal labelling L of the updated AF). That is, the context-based AF contains in the component $T2$ all the arguments whose status could change when considering two different preferred labellings. This is sufficient to locally compute the ideal extension E' of $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, \star)$, which is then equal to the restriction of the ideal extension E of the updated AF $\mathcal{A}$ to the arguments in $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, \star)$. □

Example 6.14. Continuing from Example 6.9, the ideal extension $\{\texttt{f}, \texttt{h}\}$ of $u(\texttt{AF}_0)$ is equal to $(\{\texttt{d}, \texttt{f}, \texttt{h}\} \setminus \{\texttt{c}, \texttt{d}\}) \cup \{\texttt{h}\}$ where $\{\texttt{h}\}$ is the ideal extension of $\mathsf{CBAF}(u, \texttt{AF}_0, E_{id}, \star)$. □

6.4 Incremental Computation

The results of Theorems 6.6 and 6.11, along with those of Theorem 6.13 and Fact 1, allow us to define SPA (see Algorithm 5) to decide the skeptical acceptance of a goal g w.r.t. an AF $\mathcal{A}_0$ updated by $u = \pm(a, b)$. Given the initial skeptical acceptance $SA_{\mathcal{A}_0}(g)$ of g and the ideal extension E_0 of $\mathcal{A}_0$, both $SA_{u(\mathcal{A}_0)}(g)$ and the ideal extension E of the updated AF $u(\mathcal{A}_0)$ are incrementally computed, thus enabling consecutive invocations of the algorithm to perform sequences of updates.

Algorithm SPA works as follows. First, the supporting set $S_\star = Sup\,(u, \mathcal{A}_0, E_0, \star)$ is computed at Line 1, and using Theorem 6.13 the ideal extension E of the updated AF is computed by invoking an external solver ID-Solver($\mathcal{A}_{id}$), computing the ideal extension of the context-based AF $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, \star)$ (Line 3). Then, using Fact 1, if g belongs to E, then g is skeptically accepted and the algorithm returns *true* along with the ideal extension of the updated AF (Line 5). Similarly, if g belongs to the set of arguments attacked by an argument in E, then g is not skeptically accepted and the algorithm returns *false* along with E (Line 7). Otherwise, the set $S_g = Sup(u, \mathcal{A}_0, E_0, g)$ is built (it can be efficiently done by starting from $S_\star$), and it is checked if it is empty. If this is the case, using Theorem 6.6, we can conclude that the acceptance status of g does not change after the update (Line 10). Otherwise, the context-based AF is built at Line 11 and, using Theorem 6.11, the skeptical acceptance of g is recomputed by invoking an external solver SA-Solver($\mathcal{A}_{sa}, g$) which

tells us if g is skeptically accepted w.r.t. the context-based AF $\mathsf{CBAF}(u, \mathcal{A}_0, E_0, g)$ (Line 12).

As stated next, Algorithm 5 is sound and complete.

Theorem 6.15. *If ID-Solver and SA-Solver are sound and complete, for any goal g Algorithm 5 computes $SA_{u(\mathcal{A}_0)}(g)$ w.r.t. the updated AF $u(\mathcal{A}_0)$ and the ideal extension of $u(\mathcal{A}_0)$.*

Proof. If for any AF $\mathcal{A}$ and argument g $\mathsf{ID\text{-}Solver}(\mathcal{A})$ and $\mathsf{SA\text{-}Solver}(\mathcal{A}, g)$ correctly return the ideal extension of $\mathcal{A}$ and the skeptical acceptance of g w.r.t. $\mathcal{A}$, respectively, the soundness of Algorithm 5 follows from the fact that pair retuned $\langle SA_{u(\mathcal{A}_0)}(g), E\rangle$, where E is the ideal extension of the updated AF, is correct in all of the four cases. Indeed, the pair returned at either Line 5 or Line 7 is correct due to Fact 1 and Theorem 6.13. Moreover, the pair returned at Line 10 is correct due to Theorem 6.6, while that returned at Line 12 is correct due to Theorem 6.11.

As for completeness, assuming that ID-Solver and SA-Solver are complete w.r.t. the function each of them computes, the completeness of Algorithm 5 follows by observing that the skeptical acceptance $SA_{u(\mathcal{A}_0)}(g)$ of g w.r.t. $u(\mathcal{A}_0)$ and ideal extension E of $u(\mathcal{A}_0)$ is computed whatever is the AF $\mathcal{A}_0$, the goal g and its skeptical acceptance $SA_{\mathcal{A}_0}(g)$, the update u, and the ideal extension E_0 of $\mathcal{A}_0$ taken as input. □

SPA-base: A version of SPA not using the ideal extension.

SPA-base is obtained from SPA by skipping lines 1–7 of Algorithm 5 and assuming $E_0 = \emptyset$ at lines 8 and 11 to compute S_g and $\mathcal{A}_{sa}$ respectively. Also, no ideal extension is returned (i.e., $E = \perp$). Notice that, similarly to SPA-base, SPA does not use the information provided by the initial ideal extension when $E_0 = \emptyset$, though SPA always incrementally computes the ideal extension of the updated AF.

The proposed technique is extended to deal with the case of multiple updates performed simultaneously as follows.

As said before in Section 2.4 the application of a set U of updates can be reduced to performing a single attack, and this holds also for the case of skeptical preferred acceptance.

Proposition 6.16. *Let $\mathcal{A}_0 = \langle A_0, \Sigma_0\rangle$ be an AF, and U a set of updates. Let $\mathcal{A}$ be the AF obtained from $\mathcal{A}_0$ by performing all updates in U on it. Then,*

- *$E = E^U \cap A_0$ is the ideal extension of $\mathcal{A}$ iff E^U is the ideal extension of $+(v, w)(\mathcal{A}^U)$.*
- *Given an argument g of $\mathcal{A}_0$, $SA_{\mathcal{A}}(g) = SA_{+(v,w)(\mathcal{A}^U)}(g)$.*

6.5 Implementation and Experiments

We have implemented a C++ prototype and compared our incremental technique with *ArgSemSAT* [55], the solver that won the ICCMA'17 competition for the task DS-`pr` of determining the skeptical preferred acceptance.

Algorithm 5 SPA($\mathcal{A}_0, g, SA_{\mathcal{A}_0}(g), u, E_0$)

Input: AF $\mathcal{A}_0 = \langle A_0, \Sigma_0 \rangle$, argument $g \in A_0$,
skeptical acceptance $SA_{\mathcal{A}_0}(g)$ of g w.r.t. $\mathcal{A}_0$,
update $u = \pm(a, b)$, ideal extension E_0 of $\mathcal{A}_0$.
Output: skeptical acceptance $SA_{u(\mathcal{A}_0)}(g)$ of g w.r.t. $u(\mathcal{A}_0)$,
ideal extension E of $u(\mathcal{A}_0)$.

Let $S_\star = Sup(u, \mathcal{A}_0, E_0, \star)$
Let $\mathcal{A}_{id} =$ CBAF$(u, \mathcal{A}_0, E_0, \star)$
Let $E = (E_0 \setminus S_\star) \cup$ ID-Solver$(\mathcal{A}_{id})$
if $g \in E$ **then**
    return $\langle$*true*, $E\rangle$
if $g \in E^+$ **then**
    return $\langle$*false*, $E\rangle$
Let $S_g = Sup(u, \mathcal{A}_0, E_0, g)$
if S_g is empty **then**
    return $\langle SA_{\mathcal{A}_0}(g), E\rangle$
Let $\mathcal{A}_{sa} =$ CBAF$(u, \mathcal{A}_0, E_0, g)$
return $\langle$SA-Solver$(\mathcal{A}_{sa}, g)$, $E\rangle$

Datasets.

We used benchmarks from the DS-pr track of ICCMA'17, that is, the dataset $A2$ consisting of 50 AFs with a number of arguments $|A| \in [61,\ 20K]$ and a number of attacks $|\Sigma| \in [97,\ 358K]$, and the dataset $A3$ consisting of 100 AFs with $|A| \in [39,\ 100K]$ and $|\Sigma| \in [72,\ 1.26M]$.

Methodology.

For each AF $\mathcal{A}_0$ in the dataset, we randomly selected an update u (or a set U of updates when considering multiple updates), and an argument g. Then, we computed the skeptical acceptance of g w.r.t. the updated AF $u(\mathcal{A}_0)$ by using 1) SPA, that is Algorithm 5 where ID-Solver is *pyglaf* [7] and SA-Solver is *ArgSemSAT*; 2) SPA-base where only *ArgSemSAT* is used; and 3) *ArgSemSAT* (from scratch).

We measured the efficiency of the proposed technique as follows. For AF $\mathcal{A}_0$, update u, and argument g, let t_A and t_B be the amount of time required by SPA and SPA-base, respectively, to compute $SA_{u(\mathcal{A}_0)}(g)$. Let t_S be the time required by *ArgSemSAT* to compute $SA_{u(\mathcal{A}_0)}(g)$ from scratch. Then, the *improvements* of SPA and SPA-base over *ArgSemSAT* are defined as $\frac{t_S}{t_A}$ and $\frac{t_S}{t_B}$, respectively. Thus, an improvement equal to x means that the incremental computation is x times faster than the computation from scratch.

In the figures, each data-point is the average of 10 runs, and solid lines are obtained by linear regression.

Experiment 1.

Figure 6.2 reports the improvement (log scale) of SPA and SPA-base over *ArgSemSAT* on datasets $A2$ (LHS) and $A3$ (RHS) for single updates versus the size of the

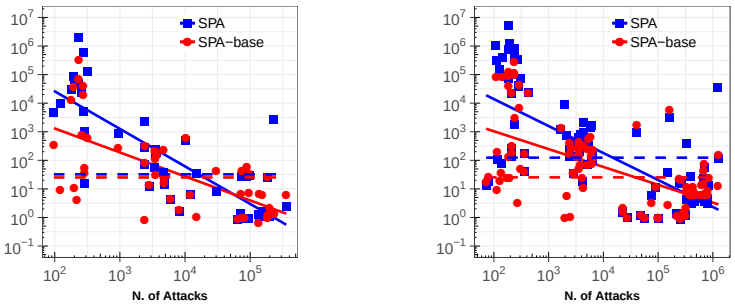
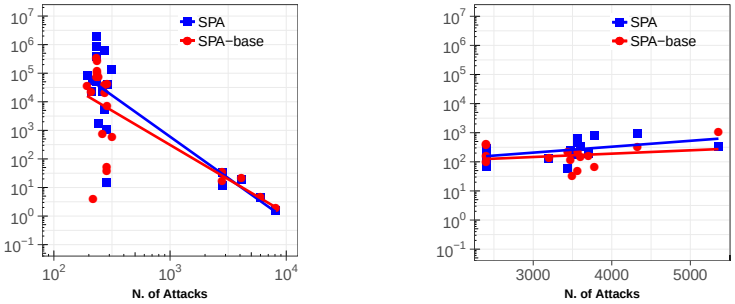


Fig. 6.2: (Experiment 1). Improvement of SPA and SPA-base over *ArgSemSAT* on datasets $A2$ (left-hand side) and $A3$ (right-hand side).



(a) (b)

Fig. 6.3: (Experiment 2). Improvement of SPA and SPA-base over *ArgSemSAT* on AFs from datasets $A2$ and $A3$ having (*a*) a number of arguments in $[160, 200]$; (*b*) an average degree in $[5, 10]$.

AFs, i.e., the number of attacks. Both SPA and SPA-base significantly outperform the computation from scratch, though the improvement decreases as the number of attacks increases—see Experiment 2 for an analysis of this behaviour. Considering the averages of the improvements, SPA and SPA-base turn out to be 5 and 4 orders of magnitude faster than *ArgSemSAT*, respectively. However, as this can be skewed by extremely large values of improvements (e.g. 10^6), we also considered the medians of improvements for SPA (32 on $A2$, 134 on $A3$) and SPA-base (27 on $A2$, 40 on $A3$) (see dashed line in Figure 6.2), which confirm the significance of the gain in efficiency. The experiments show that SPA is generally faster than SPA-base, except for a few AFs whose initial ideal extension is empty.

Experiment 2.

We analysed the performances of SPA and SPA-base by varying the number of attacks and keeping almost constant either the number of arguments or the average degree (i.e., N. of attacks / N. of arguments). To this end, we selected as many AFs as possible from the two datasets having these properties. Figure 6.3 reports the improvement for AFs having (*a*) a number of arguments in $[160, 200]$ and (*b*) an

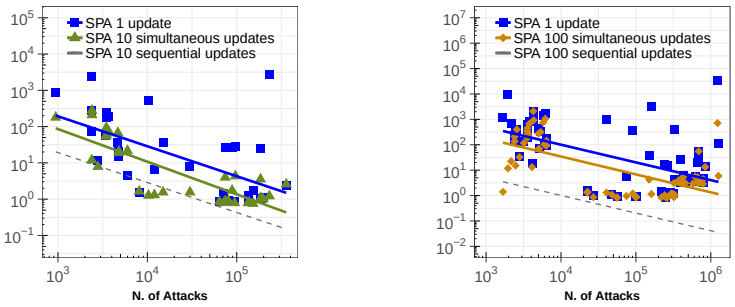


Fig. 6.4: (Experiment 3). Improvement of SPA over *ArgSemSAT* for 10 updates (LHS) and 100 updates (RHS). Dashed gray lines are improvements for 10 and 100 updates applied sequentially.

average degree in $[5, 10]$. Figure 6.3(a) shows that the performance gets worse when the increasing of the number of attacks is mainly due to the increasing of nodes' degree (in experiments it varies from 1.5 to 40) and the AFs become more and more dense. Figure 6.3(b) shows that when the average degree does not change significantly and the increasing of the number of attacks is mainly caused by the increasing of the number of arguments (in experiments it has more than doubled, from $2.4K$ to $5.4K$), the improvement does not decrease. This can be explained by looking at the ratio ρ between the size of the context-based AF and that of the initial AF. Indeed, ρ increases from 4% to 95% when the average degree of the input AFs varies from 2 to 40 in Figure 6.3(a). On the other hand, for Figure 6.3(b), ρ is almost constant (i.e., in $[60\%, 65\%]$). Thus, the performance gets worse when the ratio between the size of the context-based AF and that of the initial AF becomes very large because of the increasing density of the initial AFs.

Experiment 3.

Figure 6.4 reports the improvement of SPA over *ArgSemSAT* on datasets $A2$ (LHS) and $A3$ (RHS) for 10 and 100 updates performed simultaneously. To preserve the structure of the AFs in the datasets, we changed at most 1% (resp., 10%) of the number of attacks for the AFs of $A2$ (resp., $A3$), that is, we considered AFs whose size is greater than $1K$. Figure 6.2 also reports the improvement for 1 update, as well as for 10 and 100 updates applied sequentially, i.e., one after another (see dashed lines). The results show that SPA remains faster than the competitor even when 10 or 100 updates are performed simultaneously. Moreover, despite the overhead of the construction given before Proposition 6.16, applying updates simultaneously is faster than applying them sequentially.

6.6 Summary

Although handling dynamic AFs is becoming central to argumentation in AI, the problem of computing skeptical acceptance within dynamic AF has not been unex-

plored so far. To the best our knowledge, this is the first work proposing an efficient technique for the incremental computation of skeptical acceptance in dynamic AFs.

We have proposed a new algorithm, called SPA, which incrementally computes the skeptical preferred acceptance by maintaining the ideal extension. For a better understanding of the relevance of computing the ideal extension, we have also considered its simpler variant SPA-base which does not rely on the ideal extension. Both algorithms outperform the computation from scratch, and SPA is generally faster than SPA-base. However, as the experiments showed, SPA may be slower than SPA-base when the initial ideal extension is empty. Thus, a first direction for future work is devising heuristics to take advantages of both algorithms (for instance, by avoiding the computation of the ideal extension at each step, when the current ideal extension is empty).

Analogously to classical AF solvers, our approach allows us to determine the skeptical acceptance of a single argument. However, the definition of supporting set and context-based AF can be extended to sets of arguments—a way to do it is using the union of the supporting sets to compute the context-based AF. As a further line for future work, we plan to extend Algorithm 5 to take as input a set of arguments. In this direction, SPA always computes the ideal extension, which being contained in every preferred extension, already provides additional skeptical preferred arguments other than the goal. Other directions for future work include extending our technique to work also in the context of BAFs by using a similar approach as done in Chapter 5.

Finally, another direction for future work is extending the technique to other argumentation semantics. Additionally, the ”base” version of our algorithm can be extended to deal with other argumentation semantics that satisfy directionality principle [17].

7

Incremental Computation of Warranted Arguments over Dynamic Defeasible Logic Programs

"Repetition of an argument proves your determination, not truth."

– Raheel Farooq

Recently there has been an increasing interest in structured argumentation, an extension of the classical argumentation framework (AF) where arguments are not simple simbols (associated to statements), but have a structure defined by a set of (strict and defeasible) rules. Considering the structure of arguments allows users to analyze reasons for and against a conclusion; the *warrant status* of such a conclusion in the context of a knowledge base represents the main output of a dialectical process. A naive approach to computing such statuses is costly, and any update to the knowledge base potentially has a huge impact if done in this manner. 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 an algorithm for the incremental computation of the warrant status of conclusions in DeLP knowledge bases that evolve over time due to the application of updates. We present an experimental evaluation showing that our incremental algorithm yields significantly faster running times in practice, as well as overall fewer recomputations.

7.1 The Defeasible Logic Programming Formalism for Structured Argumentation

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 chapter, we adopt the Defeasible Logic Programming (DeLP) language [79], which is a logic programming-based approach in which knowledge bases, or programs, are composed of facts and rules, which can be either strict or defeasible.

In [27], four frameworks that consider the structure of arguments are presented; two of them—ASPIC$^+$ [94] and ABA [110]—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 [28] and DeLP [78]—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, ASPIC$^+$ and ABA make the analysis rather difficult since Dung's semantics are abstract and therefore not concerned with the internal components of the arguments involved. The last two frameworks mentioned above exhibit several differences [27]—among them is the base logic used as a knowledge representation language: [28] relies on propositional logic, requiring a theorem prover to solve queries, while DeLP [78] adopts an extension of logic programming, which is on the other hand a computational framework. An important distinction between DeLP and the other three frameworks, which significantly affects the resolution of a query, rests on the types of attacks that can be considered among arguments. As we will describe below, DeLP considers two types of defeaters: *proper* and *blocking*; the former is akin to Dung's form of defeat, called *attack* in [61], 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 enriches the reasoning process by forbidding common dialogical fallacies; these characteristics have been reflected in the development of a game-based semantics [112]. These considerations led us to the choice of DeLP as the basis of the 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 [98].

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 dialog—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. Toward this end, in this chapter 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 the problems are NP-hard—this motivates the investigation of incremental/efficient techniques.
- We identify several cases in which updates are guaranteed to have no effect, i.e., updates that are *irrelevant*, as the status of the literals does not change after performing such kinds of updates.
- We propose an approach based on the construction of a labeled hyper-graph 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 the proposed approach, showing that it can lead to significant savings in running time for both rule addition and removal updates.

The area of computational argumentation concerned with actual implementations is in need of more research. There is an effort to develop abstract argumentation solvers [108] that will benefit the systems that use this approach. Several researchers have already examined the dynamics of argumentation following belief revision techniques or abstract argumentation but not looking at the structure of arguments. The proposed work is based on a system that directly implements an answer semantics, i.e., it is concerned with deciding the status of a literal after a particular change in the KB (program), looking to improve performance.

7.2 Basics of Defeasible Logic Programming

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 [79]. 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 "$\sim$" represents strong negation. We may write $\sim\sim\alpha$ 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 $\mathcal{P}$ is a set of *strict* and *defeasible* rules defined using elements of **AT** as follows. A fact is a literal. 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 $\alpha_0, \alpha_1, \ldots, \alpha_n$ ($n > 0$) 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 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 occurring in a fact or a rule of $\mathcal{P}$.

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 are not 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 $\mathcal{P} = \langle \Pi, \Delta \rangle$, we denote as $CL(\mathcal{P})$ 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 7.1, $CL(\mathcal{P}_1) = \{\sim a, b, d, t\}$. 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 obviously contradictory. A set of literals is said to be contradictory if it contains two contradictory literals.

Example 7.1 (Running example). Let $\mathcal{P}_1 = (\Pi_1, \Delta_1)$ be a DeLP-program for supporting doctors in medical diagnoses. Literals in $Lit_{\mathcal{P}_1}$ 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 should take sleeping aids
i	Patient is diagnosed with insomnia
e	Patient shows symptoms of attention deficit disorder
f	Patient suffers from forgetfulness

Let's assume that the strict part of $\mathcal{P}_1$ is $\Pi_1 = \{\sim a,\ t,\ b,\ (d \leftarrow t)\}$, and that the set of defeasible rules is as follows:

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

The (non-contradictory) set of literals that can be derived from Π_1 is $\{\sim a, t, b, d\}$. However, both $\sim i$ and i can be derived from $\mathcal{P}_1$ using the following sets of rules and facts: $\{(\sim i \prec \sim a, s), (s \prec d), (d \leftarrow t), (t)\}$ and $\{(i \prec s), (s \prec d), (d \leftarrow t), (t)\}$, respectively. □

We use $CL(\mathcal{P})$ to denote the set of literals obtained by computing the deductive closure of facts and strict rules of $\mathcal{P}$. For instance, $CL(\mathcal{P}_1) = \{\sim a, b, d, t\}$

7.2.1 The 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 7.2 (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 7.3. Some arguments for $\mathcal{P}_1$ are the following:

$\langle \mathcal{A}_1, i \rangle = \langle \{(i \multimap s),\ (s \multimap h),\ (h \multimap b)\}, i \rangle$;
$\langle \mathcal{A}_2, i \rangle = \langle \{(i \multimap s),\ (s \multimap d)\}, i \rangle$;
$\langle \mathcal{A}_3, h \rangle = \langle \{(h \multimap b)\}, h \rangle$;
$\langle \mathcal{A}_4, \sim i \rangle = \langle \{(\sim i \multimap \sim a, s),\ (s \multimap d)\}, \sim i \rangle$;
$\langle \mathcal{A}_5, \sim h \rangle = \langle \{(\sim h \multimap t, d)\}, \sim h \rangle$;
$\langle \mathcal{A}_6, h \rangle = \langle \{(h \multimap d)\}, h \rangle$. □

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, $\langle \mathcal{A}_7, s \rangle = \langle \{(s \multimap h),\ (h \multimap b)\}, s \rangle$ is a sub-argument of $\langle \mathcal{A}_1, i \rangle$.

A literal α is said to be *warranted* if there exists a non-defeated argument $\mathcal{A}$ supporting α. To establish if the argument $\langle \mathcal{A}, \alpha \rangle$ is a non-defeated argument, *defeaters* for $\langle \mathcal{A}, \alpha \rangle$ are considered, i.e., counter-arguments that by some criterion are preferred to $\langle \mathcal{A}, \alpha \rangle$. The comparison criterion is a modular part of the argumentation inference engine. It is possible to use *priority* among rules, where an explicit preference relation among rules is given; an alternative is *generalized specificity*, where no explicit order among rules or arguments is needed, preferring arguments with greater information content or with less use of rules (w.r.t. subset minimality) [106]. We will abstract away from this criterion and simply assume the existence of a preference relation $\succ$ between arguments: $\langle \mathcal{A}, \alpha \rangle \succ \langle \mathcal{B}, \beta \rangle$ means that argument $\langle \mathcal{A}, \alpha \rangle$ is preferred to argument $\langle \mathcal{B}, \beta \rangle$.

Definition 7.4 (Defeater). *Let $\mathcal{P} = (\Pi, \Delta)$ be a DeLP program, and $\succ$ be a preference relation defined over arguments for $\mathcal{P}$. 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 $\langle \mathcal{B}, \beta \rangle$, and
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 an argument $\langle \mathcal{B}, \beta \rangle$ 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 chapter 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 [106], as defined in what follows.

Definition 7.5 (Generalized Specificity Criterion [79]). *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 $\Pi_s \cup \mathcal{L}' \cup \mathcal{A}_2$, and α_2 does not have a strict derivation from $\Pi_s \cup \mathcal{L}'$, and α_1 does not have a defeasible derivation from $\Pi_s \cup \mathcal{L}' \cup \mathcal{A}_1$.*

From now on we assume that the status of arguments is computed by using Definition 7.5 of generalized specificity, and it is used as preference relation $\succ$ among arguments, which is also the default criterion adopted by the DeLP solver used in the experiments.

Definition 7.6 (Argumentation Line). *An* argumentation line *for $\langle \mathcal{A}_0, \alpha_0 \rangle$ is a sequence Λ of arguments where each argument defeats its predecessor, that is $\Lambda = [\langle \mathcal{A}_0, \alpha_0 \rangle, \langle \mathcal{A}_1, \alpha_1 \rangle, \langle \mathcal{A}_2, \alpha_2 \rangle, \dots]$ where $\langle \mathcal{A}_i, \alpha_i \rangle$ is a defeater for $\langle \mathcal{A}_{i-1}, \alpha_{i-1} \rangle$, for all $i > 0$.*

A line $\Lambda = [\langle \mathcal{A}_0, \alpha_0 \rangle, \langle \mathcal{A}_1, \alpha_1 \rangle, \langle \mathcal{A}_2, \alpha_2 \rangle, \dots]$ can be split in two disjoint sets: $\Lambda_S = \{\langle \mathcal{A}_0, \alpha_0 \rangle, \langle \mathcal{A}_2, \alpha_2 \rangle, \dots\}$ of *supporting* arguments, and $\Lambda_I = \{\langle \mathcal{A}_1, \alpha_1 \rangle, \langle \mathcal{A}_3, \alpha_3 \rangle, \dots\}$ of *interfering* arguments.

To avoid undesirable sequences that may represent circular or fallacious argumentation lines, an argumentation line must be *acceptable*. Given a DeLP program $\mathcal{P} = (\Pi, \Delta)$, two arguments $\langle \mathcal{A}, \alpha \rangle$ and $\langle \mathcal{B}, \beta \rangle$ are said to be *concordant* iff $\Pi \cup \mathcal{A} \cup \mathcal{B}$ is not contradictory. More generally, a set $\{\langle \mathcal{A}_i, \alpha_i \rangle\}_{i=1}^{n}$ of arguments is concordant iff $\Pi \cup \bigcup_{i=1}^{n} \mathcal{A}_i$ is not contradictory.

Definition 7.7 (Acceptable Line). *An argumentation line $\Lambda = [\langle \mathcal{A}_0, \alpha_0 \rangle, \langle \mathcal{A}_1, \alpha_1 \rangle, \langle \mathcal{A}_2, \alpha_2 \rangle, \dots]$ is acceptable iff*

i) it is a finite sequence;

ii) both the set Λ_S of supporting arguments and the set Λ_I of interfering arguments are concordant;
iii) no argument $\langle \mathcal{A}_k, \alpha_k \rangle$ in Λ is a sub-argument of an argument $\langle \mathcal{A}_i, \alpha_i \rangle$ ($i < k$) appearing earlier in Λ;
iv) for all i such that the argument $\langle \mathcal{A}_i, \alpha_i \rangle$ is a blocking defeater for $\langle \mathcal{A}_{i-1}, \alpha_{i-1} \rangle$, if argument $\langle \mathcal{A}_{i+1}, \alpha_{i+1} \rangle$ exists, then $\langle \mathcal{A}_{i+1}, \alpha_{i+1} \rangle$ is a proper defeater for $\langle \mathcal{A}_i, \alpha_i \rangle$.

Example 7.8. Consider the arguments from Example 7.3 and the following preference relations: $\mathcal{A}_5 \succ \mathcal{A}_6$ and $\mathcal{A}_3 \succ \mathcal{A}_5$. Then, $[\langle \mathcal{A}_6, h \rangle, \langle \mathcal{A}_5, \sim h \rangle, \langle \mathcal{A}_3, h \rangle]$ is an acceptable argumentation line. □

It is easy to see that many acceptable argumentation lines could arise from one argument, leading to a tree structure. This tree is called *dialectical* because it represents an exhaustive dialectical analysis for the argument in its root.

Definition 7.9 (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 of the tree is labeled with $\langle \mathcal{A}_0, \alpha_0 \rangle$.*
- *Let N be a non-root node of the tree labeled $\langle \mathcal{A}_n, \alpha_n \rangle$, and $\Lambda = [\langle \mathcal{A}_0, \alpha_0 \rangle, \ldots, \langle \mathcal{A}_n, \alpha_n \rangle]$ the sequence of labels of the path from the root to N. Let $\langle \mathcal{B}_1, \beta_1 \rangle, \ldots, \langle \mathcal{B}_k, \beta_k \rangle$ be all the defeaters for $\langle \mathcal{A}_n, \alpha_n \rangle$. For each defeater $\langle \mathcal{B}_i, \beta_i \rangle$ ($1 \leq i \leq k$), such that the argumentation line $\Lambda' = [\langle \mathcal{A}_0, \alpha_0 \rangle, \ldots, \langle \mathcal{A}_n, \alpha_n \rangle, \langle \mathcal{B}_i, \beta_i \rangle]$ is acceptable, then node N has a child N_i labeled $\langle \mathcal{B}_i, \beta_i \rangle$. If there is no defeater for $\langle \mathcal{A}_n, \alpha_n \rangle$ or there is no $\langle \mathcal{B}_i, \beta_i \rangle$ s.t. Λ' is acceptable, then N is a leaf.*

Dialectical trees can be marked in order to decide the status of the literals at their roots.

Definition 7.10 (Marking of a Dialectical Tree). *Let $\mathcal{T}_{\langle \mathcal{A}, \alpha \rangle}$ be a dialectical tree for $\langle \mathcal{A}, \alpha \rangle$. The corresponding marked dialectical tree, denoted $\mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$, will be obtained marking every node in $\mathcal{T}_{\langle \mathcal{A}, \alpha \rangle}$ as follows:*

- *Leaves in $\mathcal{T}_{\langle \mathcal{A}, \alpha \rangle}$ are marked UNDEFEATED in $\mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$.*
- *Let $\langle \mathcal{B}_i, \beta_i \rangle$ be an inner node of $\mathcal{T}_{\langle \mathcal{A}, \alpha \rangle}$. Then $\langle \mathcal{B}_i, \beta_i \rangle$ will be marked as UNDEFEATED in $\mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$ iff every child of $\langle \mathcal{B}_i, \beta_i \rangle$ is marked as DEFEATED. The node $\langle \mathcal{B}_i, \beta_i \rangle$ will be marked DEFEATED in $\mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$ iff it has at least one child marked as UNDEFEATED.*

The warrant status is then easily determined by checking how the roots are marked.

Definition 7.11 (Warranted Literal). *Let $\langle \mathcal{A}, \alpha \rangle$ be an argument and $\mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$ its associated marked dialectical tree. The literal α is warranted iff the root of $\mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$ is marked as UNDEFEATED. We will say that $\mathcal{A}$ is a warrant for α.*

Example 7.12. In our running example, h is a warranted literal for $\mathcal{P}_1$. In fact, $\mathcal{T}^*_{\langle \mathcal{A}_6, h\rangle}$ for $\langle \mathcal{A}_6, h\rangle$ consists of only the argumentation line $[\langle \mathcal{A}_6, h\rangle, \langle \mathcal{A}_5, \sim h\rangle, \langle \mathcal{A}_3, h\rangle]$ of Example 7.8. Thus, $\langle \mathcal{A}_3, h\rangle$ and $\langle \mathcal{A}_6, h\rangle$ are marked as *UNDEFEATED*, while $\langle \mathcal{A}_5, \sim h\rangle$ is *DEFEATED*. It is easy to see that $t, d, b, h, s, \sim a$ are warranted as well. □

Given a DeLP-program $\mathcal{P}$, we define a total function $S_\mathcal{P} : Lit \to \{\text{IN}, \text{OUT}, \text{UNDECIDED}\}$ 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 $\mathcal{S}_\mathcal{P}(\alpha) = \text{IN}$ 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, $S_{\mathcal{P}_1}(h) = \text{IN}$, $S_{\mathcal{P}_1}(a) = \text{OUT}$, and $S_{\mathcal{P}_1}(i) = \text{UNDECIDED}$.

7.2.2 Updates

An *update* modifies a DeLP-program $\mathcal{P}$ into a new one $\mathcal{P}'$ 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 $\mathcal{P}$, and $u(\mathcal{P})$ to denote the DeLP-program resulting from the application of update $u = \pm(r)$ to $\mathcal{P}$— we use $\mathcal{P}'$ instead of $u(\mathcal{P})$ if the update is understood or we do not need to specify u explicitly.

Given a DeLP-program $\mathcal{P} = (\Pi, \Delta)$, and a strict or defeasible rule r, the *updated* program $\mathcal{P}' = u(\mathcal{P})$ is as follows.

– If r is a strict rule and $u = +r$, then $\mathcal{P}' = ((\Pi \cup \{r\}), \Delta)$ if $(\Pi \cup \{r\})$ is not contradictory, otherwise $\mathcal{P}' = \mathcal{P}$ (i.e., it has no effect if it would yield a contradictory program).

– If r is a strict rule and $u = -r$, then $\mathcal{P}' = ((\Pi \setminus \{r\}), \Delta)$; note that $(\Pi \setminus \{r\})$ is guaranteed to be not contradictory.

– If r is defeasible and $u = +r$, then $\mathcal{P}' = (\Pi, (\Delta \cup \{r\}))$.

– Finally, if $r \in \Delta$ and $u = -r$, then $\mathcal{P}' = (\Pi, (\Delta \setminus \{r\}))$.

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

Example 7.13. Consider the DeLP-program $\mathcal{P}_1 = (\Pi_1, \Delta_1)$ of Example 7.1, and the update $u = +r$ where $r = (\sim i \multimap h)$. The updated DeLP-program is $(\Pi_1, (\Delta_1 \cup \{r\}))$. Note that new arguments may arise after performing the update; for instance, $\langle \mathcal{A}_8, \sim i\rangle = \langle \{\sim i \multimap h,\ h \multimap b\}, \sim i\rangle$ exists for the updated program but it did not exist for $\mathcal{P}_1$. □

We will use $CL(u, \mathcal{P}) = CL(\mathcal{P}) \cap CL(u(\mathcal{P}))$ to denote the set of literals which are in the deductive closure of facts and strict rules both before and after an update.

7.2.3 Hyper-graphs for DeLP Programs

A *directed hyper-graph* is a pair $\langle N, H \rangle$, where N is the set of nodes and $H \subseteq 2^N \times N$ is the set of hyper-edges. For hyper-edge (S, t), S is a possibly empty set and it is called the *source set*, while t is called the *target node*. The directed, labeled hyper-graph $G(\mathcal{P})$ is recursively defined as follows.[1]

Definition 7.14 (Labeled hyper-graph for a DeLP program). *Let $\mathcal{P}$ be a program and $L = \{\texttt{def}, \texttt{str}, \texttt{cfl}\}$ be an alphabet of labels. $G(\mathcal{P}) = \langle N, H \rangle$, where $H \subseteq 2^N \times N \times L$, 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 $(\{\alpha_1, \ldots, \alpha_n\}, \alpha_0, \texttt{str}) \in H$ (resp. $(\{\alpha_1, \ldots, \alpha_n\}, \alpha_0, \texttt{def}) \in H$);*
- *(final step) For each pair of nodes in N representing complementary literals α and $\sim\alpha$, both $(\{\alpha\}, \sim\alpha, \texttt{cfl}) \in H$ and $(\{\sim\alpha\},\ \alpha, \texttt{cfl}) \in H$.*

Thus, the literals for which there exists a strict derivation 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.

The hyper-graph $G(\mathcal{P}_1)$ for the DeLP-program $\mathcal{P}_1$ of Example 7.1 is shown in Figure 7.1(a) where $\leftrightarrow$ (resp. $\leftarrow$ and $\blacktriangleleft$) denotes hyper-edges labelled as `cfl` (resp. `def` and `str`). By definition of $G(\mathcal{P})$, if a literal $\alpha \in Lit_{\mathcal{P}}$ does not belong to $G(\mathcal{P})$, then there is no argument for it w.r.t. $\mathcal{P}$. However, there may be literals in $G(\mathcal{P})$ that have no argument. As shown later, deciding whether there is an argument for a given literal is NP-hard.

Example 7.15. Consider program $\mathcal{P}'_1$ obtained by update $u = -(d \leftarrow t)$ on $\mathcal{P}_1$. $G(\mathcal{P}'_1)$ is shown in Figure 7.1(b). Since d is not in $G(\mathcal{P}'_1)$, there is no argument for it w.r.t. $\mathcal{P}'_1$. □

Given $G(\mathcal{P}) = \langle N, H \rangle$, we say that there is a path from a literal β to a literal α (and write $path_{G(\mathcal{P})}(\beta, \alpha)$, or simply $path(\beta, \alpha)$, whenever the graph is understood), if either *i*) there exists an hyper-edge whose source set contains β and whose target is α, or *ii*) there exist 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 $G(\mathcal{P})$. We also say that a node y is *reachable* from a set X of

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

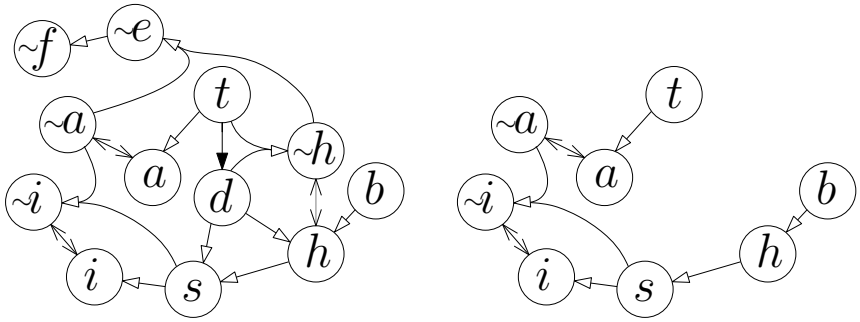


Fig. 7.1: (a) $G(\mathcal{P}_1)$ for the program $\mathcal{P}_1$ of Example 7.1; (b) $G(\mathcal{P}'_1)$ for the program $\mathcal{P}'_1 = -(d \leftarrow t)(\mathcal{P}_1)$ of Example 7.15.

nodes if there exists a path 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 of Example 7.1 and its hypergraph shown in Figure 7.1(a), we have that $Reach_{G(\mathcal{P}_1)}(\{d\}) = \{d, h, \sim h, s, \sim i, i, \sim e, \sim f\}$. Moreover, we also use the notations $ReachStr_{G(\mathcal{P})}(X)$ to denote the set of nodes reachable from X using only strict hyper-edges, and $ReachStr^{-1}_{G(\mathcal{P})}(X)$ to denote the set of nodes that are reachable from X by navigating backward strict hyper-edges only.

Structure $G(\mathcal{P})$ can be built in polynomial time in the size of $\mathcal{P}$; thus, both deciding whether a literal depends on another one, and computing $Reach_{G(\mathcal{P})}(X)$, can be done in PTIME.

7.3 Complexity Analysis

The status of a literal is computed by the dialectical process that builds a (marked) dialectical tree where each node is an argument. Thus, an important step of this process is deciding whether there exists an argument for a given literal to build counterarguments. A first result states that deciding whether there exists an argument for a literal is costly.

Theorem 7.16. *Given $\mathcal{P}$ and a literal $\alpha \in Lit_{\mathcal{P}}$, deciding whether there is an argument for α w.r.t. $\mathcal{P}$ is NP-complete.*

Proof. Membership in NP follows from the results in [54], 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—that is, 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 [100], whose definition is recalled next. An

instance of 3SAT is a pair $\langle U, \Phi \rangle$, where $U = \{x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_k, \overline{x}_k\}$ 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 $C_1 \wedge \cdots \wedge C_n$ defined over U. Specifically, each 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 U. A *truth assignment* for the variables in U is a function $\tau : U \rightarrow \{\textit{true, false}\}$ such that, for each pair of variables $x, \overline{x}$ in U, 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 $\langle U, \Phi \rangle$ is true iff there is a truth assignment τ for U such that $\tau(\Phi)$ is true.

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

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

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

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

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

- $\phi \prec\!\!\!- C_1 \wedge \cdots \wedge C_n$;
- $C_i \prec\!\!\!- x_{i,j}$ (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 C_i (resp., $\tau(\overline{x}_{i,j})$ is true and $\overline{x}_{i,j}$ is in 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.

Basically, $\mathcal{B}$ contains (at least) a rule of the form $C \prec\!\!\!- x$ for each clause in $C \in \Phi$ 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 is $\mathcal{A} \subseteq \mathcal{B}$ 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 U such that $\tau(\Phi)$ is true as follows.

1) For each defeasible rule $x_i \prec\!\!\!- f_i$ in $\mathcal{A}$, we define $\tau(x_i) = \textit{true}$ and $\tau(\overline{x}_i) = \textit{false}$.

2) For each defeasible rule $\sim x_i \prec f_i$ in $\mathcal{A}$, we define $\tau(x_i) = \textit{false}$ and $\tau(\overline{x}_i) = \textit{true}$.
3) Let $U' \subseteq U$ 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 $x \in U'$ either *true* or *false*, provided that $\tau(x) = \neg\tau(\overline{x})$. Let us assume that $\tau(x) = \textit{true}$ and $\tau(\overline{x}) = \textit{false}$ for each $x, \overline{x} \in U'$.

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 U, $\tau(x) = \neg\tau(\overline{x})$—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 are satisfied by τ, and thus that $\tau(\Phi)$ is true. □

As the DeLP-program defined in the hardness proof of Theorem 7.16 can be reduced to an equivalent one where all rules have at most two literals in the body, the result holds even for this kind of programs.

Corollary 7.17. *Let $\mathcal{P} = (\Pi, \Delta)$ be a DeLP-program such that for all $r \in (\Pi \cup \Delta)$, $|body(r)| \leq 2$. Deciding whether there is an argument for $\alpha \in Lit_{\mathcal{P}}$ w.r.t. $\mathcal{P}$ is NP-complete.*

Proof. Membership follows from Theorem 7.16. Hardness can be proved by slightly modifying the construction in the proof of Theorem 7.16 so that all rules have at most two literals in the body. Specifically, we only need to rewrite the defeasible rule $\phi \prec C_1 \wedge \cdots \wedge C_n$, as the others have at most one literal in the body. It is easy to see that $\phi \prec C_1 \wedge \cdots \wedge C_n$ 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 7.16 and such that each rule has at most two literals in the body. The statement follows. □

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

Proposition 7.18. *Given $\mathcal{P} = (\Pi, \Delta)$ and a literal $\alpha \in Lit_{\mathcal{P}}$, deciding whether there is an argument for α w.r.t. $\mathcal{P}$ 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 the definition of $G(\mathcal{P})$, according to which a node is added to the hyper-graph only if there is a derivation for it starting from the facts of $\mathcal{P}$. □

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 the definition of $G(\mathcal{P})$ and the fact that it can be built in PTIME.

Since in order to decide the status of a literal we must decide whether there is an argument for at least one literal, Theorem 7.16 and Corollary 7.17 allow us to conclude that computing the status of an argument is NP-hard.

Corollary 7.19. *Let $\mathcal{P} = (\Pi, \Delta)$ be a DeLP-program such that for all $r \in (\Pi \cup \Delta)$, $|body(r)| \leq 2$. Deciding whether $S_{\mathcal{P}}(\alpha) =$* IN*, $S_{\mathcal{P}}(\alpha) =$* OUT*, or $S_{\mathcal{P}}(\alpha) =$* UNDECIDED*, for $\alpha \in Lit_{\mathcal{P}}$ is NP-hard.*

Proof. We show that deciding whether $S_{\mathcal{P}}(\alpha) =$ IN is NP-hard, from which the other two results easily follow.

We provide a reduction from 3SAT by providing a construction obtained by augmenting that in the proof of Theorem 7.16. In the following, we use the notation introduced in the proof of Theorem 7.16. Given an instance $\langle U, \Phi \rangle$ of 3SAT, we define an instance $\langle \mathcal{P}', \alpha \rangle$ of our problem as follows. Let $\mathcal{P} = (\Pi, \Delta)$ be the DeLP program defined in the proof of Theorem 7.16. Then, $\mathcal{P}'$ is as follows: $\mathcal{P}' = (\Pi', \Delta')$ where:

- Π' consists of all the facts in Π plus a fact f'_i for each pair of variables $x_i, \overline{x}_i \in U$;
- Δ' consists of the defeasible rules in Δ plus the following defeasible rules: for each pair of variables $x_i, \overline{x}_i \in U$, the defeasible rules $x_i \multimap f'_i$ and $\sim x_i \multimap f'_i$.

Finally, we have a preference relation $\succ$ between arguments such that, for each propositional variable x_i, argument $\langle \{x_i \multimap f'_i\}, x_i \rangle$ is preferred to argument $\langle \{x_i \multimap f_i\}, x_i \rangle$, and argument $\langle \{\sim x_i \multimap f'_i\}, \sim x_i \rangle$ is preferred to argument $\langle \{\sim x_i \multimap f_i\}, \sim x_i \rangle$. Thus, arguments for x_i (resp. or $\sim x_i$) using primed facts f'_i are preferred over those using not primed arguments f_i.

We now show that $\langle U, \Phi \rangle$ is true iff $S_{\mathcal{P}}(\phi) = \text{IN}$, where ϕ is the literal in (Π, Δ), and thus in (Π', Δ'), corresponding to the 3SAT formula.

($\Rightarrow$) We show that if Φ is satisfiable then there exists a dialectical tree $\mathcal{T}_{\langle \mathcal{A}, \phi \rangle}$ w.r.t. $\mathcal{P}'$ that is marked UNDEFEATED. As shown in the proof of Theorem 7.16, if Φ is satisfiable then there is an argument $\langle \mathcal{A}, \phi \rangle$ for ϕ w.r.t. $\mathcal{P}$. Then, there is the same argument $\langle \mathcal{A}, \phi \rangle$ for ϕ w.r.t. $\mathcal{P}'$—the fact that we added some rules in $\mathcal{P}'$ does not invalidate argument $\mathcal{A}$ for ϕ.

Argument $\mathcal{A}$ contains (sub)arguments of the form $\mathcal{X}_i = \langle \{x_i \multimap f_i\}, x_i \rangle$ and $\overline{\mathcal{X}}_i = \langle \{\sim x_i \multimap f_i\}, \sim x_i \rangle$ that are defeated, respectively, by arguments of the form $\overline{\mathcal{X}}_i = \langle \{\sim x_i \multimap f_i\}, \sim x_i \rangle$, and $\mathcal{X}_i = \langle \{x_i \multimap f_i\}, x_i \rangle$; the latter arguments form the second level of the dialectical tree $\mathcal{T}_{\langle \mathcal{A}, \phi \rangle}$. The defeaters between these arguments and the sub-arguments of $\mathcal{A}$ are all blocking defeaters; thus, only proper defeaters can be added in the (acceptable) argumentation lines starting in the root of $\mathcal{T}_{\langle \mathcal{A}, \phi \rangle}$.

On the third level of $\mathcal{T}_{\langle \mathcal{A}, \phi \rangle}$, we have arguments of the form $\mathcal{X}'_i = \langle \{x_i \multimap f'_i\}, x_i \rangle$ and $\overline{\mathcal{X}}'_i = \langle \{\sim x_i \multimap f'_i\}, \sim x_i \rangle$, which properly defeat, respectively, the arguments $\mathcal{X}_i = \langle \{x_i \multimap f_i\}, x_i \rangle$ and $\overline{\mathcal{X}}_i = \langle \{\sim x_i \multimap f_i\}, \sim x_i \rangle$ belonging to the second level of $\mathcal{T}_{\langle \mathcal{A}, \phi \rangle}$.

Therefore, a marked dialectical tree $\mathcal{T}^*_{\langle \mathcal{A}, \phi \rangle}$ such that the root is marked UNDEFEATED exists, since all the arguments on the third level of $\mathcal{T}^*_{\langle \mathcal{A}, \phi \rangle}$ can be marked UNDEFEATED, those on the second level can be marked DEFEATED, and the root $\mathcal{A}$ turns out to be UNDEFEATED. This means that ϕ is warranted and $S_{\mathcal{P}}(\phi) = \text{IN}$.

($\Leftarrow$) If $S_{\mathcal{P}}(\alpha) = \text{IN}$ then there is a marked dialectical tree $\mathcal{T}^*_{\langle \mathcal{A}, \phi \rangle}$ such that the root is marked UNDEFEATED. This means that there is an argument $\mathcal{A}$ for ϕ. As shown in the proof of Theorem 7.16, if there is an argument $\langle \mathcal{A}, \phi \rangle$ for ϕ then Φ is satisfiable, which suffices to complete the proof for the IN case.

Concerning the problem of deciding whether $S_{\mathcal{P}}(\alpha) = \text{OUT}$, its NP-hardness follows from the fact that $S_{\mathcal{P}}(\alpha) = \text{OUT}$ iff $S_{\mathcal{P}}(\sim \alpha) = \text{IN}$, which is NP-hard as shown above. Reasoning analogously, the hardness for the problem of deciding whether $S_{\mathcal{P}}(\alpha) = \text{UNDECIDED}$ can be proved.

Finally, reasoning as in the proof of Corollary 7.19, it can be shown that the statement holds even if the DeLP program contains at most two literals in each rule's body. □

The fact that computing the status of arguments is hard motivated the investigation of incremental techniques.

7.4 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 $S_{\mathcal{P}}$. Our approach consists of two main steps:

(1) Check if the update is *irrelevant*, i.e., $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$ for all literals $\alpha \in Lit$. In such a case we say that all the literals are *preserved*, and return the initial status $S_{\mathcal{P}}$.

(2) To deal with relevant updates, we identify the set of literals that are "influenced" by an update and then define *core literals* whose status needs to be recomputed after performing an update, and only recompute their status. The remaining literals are either guaranteed to be *preserved* or their status can be *inferred* using the core literals.

In the following, given a DeLP-program $\mathcal{P}$, an update $u = \pm r$ for $\mathcal{P}$, and the updated program $\mathcal{P}' = u(\mathcal{P})$, we will use $G(u, \mathcal{P})$ to denote the hyper-graph of the updated or original program depending on whether $u = +r$ or $u = -r$, respectively. That is,

$$G(u, \mathcal{P}) = \begin{cases} G(\mathcal{P}'), \text{ if } u = +r \\ G(\mathcal{P}), \text{ if } u = -r \end{cases}$$

As it will become clearer in what follows, the reason for considering $G(\mathcal{P})$ in rule removal cases—instead of $G(\mathcal{P}')$—is that the head *head*(r) of a removed rule r may not belong to the updated hyper-graph, and this would result in reaching no literal from *head*(r) (and thus having no related literals, see for instance Example 7.15).

The next definitions characterizes the set of literal that are related to an update u and a DeLP-program $\mathcal{P}$ by exploiting concepts of reachable literals both forward and backward in $G(u, \mathcal{P})$.

Definition 7.20 (Related literals). *Let $\mathcal{P}$ be a DeLP-program, $G(\mathcal{P}) = \langle N, H \rangle$ and $u = \pm r$ be an update. Then, let*

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

the set of nodes that are related w.r.t. u and $\mathcal{P}$ is: $\mathcal{R}(u, \mathcal{P}) = \mathcal{R}^n(u, \mathcal{P})$ such that $\mathcal{R}^n(u, \mathcal{P}) = \mathcal{R}^{n-1}(u, \mathcal{P})$.

In the following we use $\mathcal{R}^*(u, \mathcal{P})$ to denote the set of literals computed as in Definition 7.20 but explicitly looking at the initial labelled hyper-graph $G(\mathcal{P})$ (instead of $G(u, \mathcal{P})$) also for the case of rule addition update.

The following lemma provides a sufficient condition under which the status of a specific literal does not change; it will be useful both for checking whether an update is irrelevant and dealing with relevant updates.

Lemma 7.21 (Related and Preserved Literals). *Let $\mathcal{P}$ be a DeLP-program, $G(\mathcal{P}) = \langle N, H \rangle$ and $u = \pm r$ be an update, and let $\mathcal{R}(u, \mathcal{P})$ be the set of nodes that are related w.r.t. u and $\mathcal{P}$. Then, a literal α occurring as a node of $G(u, \mathcal{P})$ is preserved (i.e., $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$) if $\alpha \notin \mathcal{R}(u, \mathcal{P})$.*

Proof. Let $h = head(r)$, and assume $u = +r$. Hence, $G(u, \mathcal{P})$ is $G(u(\mathcal{P}))$. First of all, we observe that if $\alpha \notin \mathcal{R}(u, \mathcal{P})$ then $\alpha \notin \mathcal{R}^*(u, \mathcal{P})$, from which it follows that h cannot belong to any dialectical tree for $\langle \mathcal{A}, \alpha \rangle$, for any $\mathcal{A}$ (i.e., $h \notin \mathcal{T}^*_{\langle \mathcal{A}, \alpha \rangle}$ for all $\mathcal{A}$). We 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 $\langle \mathcal{B}, \beta\rangle$, then it follows that the marked dialectical tree continues to exist for the updated DeLP program $\mathcal{P}'$. 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 point (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.

Similar arguments can be used to prove the claim for update $u = -r$, for which we need to consider $G(\mathcal{P})$. □

Thus, if a literal is not related w.r.t u and $\mathcal{P}$ then its status does not change after a rule addition/removal update.

7.4.1 Irrelevant Updates

We provide conditions under which the status of all the literals in Lit are guaranteed to remain unchanged w.r.t. their previous status, and thus need not be recomputed. The first proposition provided 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.

Proposition 7.22. *Let $\mathcal{P}$ be a DeLP-program and $r = \alpha_0 \prec \alpha_1, \ldots, \alpha_n$ a defeasible rule such that $\{\alpha_0, \ldots, \alpha_n\} \subseteq (Lit_{\mathcal{P}} \cap Lit_{\mathcal{P}'})$.*

(1) If $S_{\mathcal{P}}(\alpha_0) =$ IN then $+r$ is irrelevant for $\mathcal{P}$.

(2) If $S_{\mathcal{P}}(\alpha_0) =$ OUT then $-r$ is irrelevant for $\mathcal{P}$.

Proof. 1) Since $S_{\mathcal{P}}(\alpha_0) =$ IN, there exists a marked dialectical tree $\mathcal{T}^*_{\langle \mathcal{A},\alpha_0\rangle}$ for $\mathcal{P}$ whose root is marked as UNDEFEATED; the addition of rule r cannot interfere with this tree, nor any other tree determining the warrant status of the rest of the literals in $\mathcal{P}$.

2) Since $S_{\mathcal{P}}(\alpha_0) =$ OUT, then $S_{\mathcal{P}}(\sim\alpha_0) =$ IN and there exists a marked dialectical tree for $\mathcal{P}$ $\mathcal{T}^*_{\langle \mathcal{A},\sim\alpha_0\rangle}$ whose root is marked as UNDEFEATED; thus, reasoning as above, the removal of rule r cannot interfere with this tree, nor any other tree determining the warrant status of the rest of the literals in $\mathcal{P}$. □

Thus, if the update consists of adding (resp., removing) a defeasible rule involving literals whose initial status is IN (resp., OUT), then nothing changes in the status of any of the literals of the (updated) program.

However, the result of Proposition 7.22 does not hold for updates concerning strict rules. For instance, the following example shows that a strict rule update may be relevant (the status of at least one literal changes) even if the status of the head of the rule update was IN.

Example 7.23. Consider the DeLP program $\mathcal{P}_2 = \langle \Pi_2, \Delta_2\rangle$ where $\Pi_2 = \{\mathtt{d},\ \mathtt{e},\ \mathtt{f}\}$ and $\Delta_2 = \{(\mathtt{a} \prec \mathtt{d}),\ (\mathtt{a} \prec \mathtt{e}),\ (\sim\mathtt{a} \prec \mathtt{d}),\ (\mathtt{b} \prec \mathtt{a}),\ (\mathtt{b} \prec \mathtt{d}),\ (\sim\mathtt{c} \prec \mathtt{f}, \mathtt{b}),\ (\mathtt{c} \prec \mathtt{b})\}$

and let $u = +(\mathtt{b} \leftarrow \mathtt{f})$. Although $S_{\mathcal{P}_2}(\mathtt{b}) = \text{IN}$ (i.e. the status of the head of u is IN), after performing the update we have that $S_{u(\mathcal{P}_2)}(\sim\mathtt{c}) = \text{IN}$, though it was UNDECIDED before performing the update, meaning that the status of at least one literal (e.g. $\sim\mathtt{c}$) changed, and thus the update is not irrelevant. Particularly, before performing the update we have (among others) the following arguments:

- $\langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle = \langle\{(\sim\mathtt{c} \prec \mathtt{f}, \mathtt{b}),\ (\mathtt{b} \prec \mathtt{a}),\ (\mathtt{a} \prec \mathtt{d})\}, \sim\mathtt{c}\rangle$
- $\langle \mathcal{A}_2^{ex}, \sim\mathtt{c}\rangle = \langle\{(\sim\mathtt{c} \prec \mathtt{f}, \mathtt{b}),\ (\mathtt{b} \prec \mathtt{a}), (\mathtt{a} \prec \mathtt{e})\}, \sim\mathtt{c}\rangle$
- $\langle \mathcal{A}_3^{ex}, \sim\mathtt{c}\rangle = \langle\{(\sim\mathtt{c} \prec \mathtt{f}, \mathtt{b}),\ (\mathtt{b} \prec \mathtt{d})\}, \sim\mathtt{c}\rangle$
- $\langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle = \langle\{(\mathtt{c} \prec \mathtt{b}),\ (\mathtt{b} \prec \mathtt{a}), (\mathtt{a} \prec \mathtt{d})\}, \sim\mathtt{c}\rangle$
- $\langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle = \langle\{(\mathtt{c} \prec \mathtt{b}),\ (\mathtt{b} \prec \mathtt{a}), (\mathtt{a} \prec \mathtt{e})\}, \sim\mathtt{c}\rangle$
- $\langle \mathcal{A}_6^{ex}, \mathtt{c}\rangle = \langle\{(\mathtt{c} \prec \mathtt{b}),\ (\mathtt{b} \prec \mathtt{d})\}, \sim\mathtt{c}\rangle$

with $\langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle \succ \langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle$, $\langle \mathcal{A}_2^{ex}, \sim\mathtt{c}\rangle \succ \langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle$, $\langle \mathcal{A}_3^{ex}, \sim\mathtt{c}\rangle \succ \langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle$, $\langle \mathcal{A}_3^{ex}, \sim\mathtt{c}\rangle \succ \langle \mathcal{A}_6^{ex}, \mathtt{c}\rangle$ as preference relations between them.

Some acceptable argumentation lines are:

- $[\langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_6^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_3^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_2^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle]$
- $[\langle \mathcal{A}_2^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle]$
- $[\langle \mathcal{A}_2^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_3^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle]$
- $[\langle \mathcal{A}_3^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_2^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_6^{ex}, \mathtt{c}\rangle]$
- $[\langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_2^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_6^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_3^{ex}, \sim\mathtt{c}\rangle]$
- $[\langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_6^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_3^{ex}, \sim\mathtt{c}\rangle, \langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_2^{ex}, \sim\mathtt{c}\rangle]$
- $[\langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle]$
- $[\langle \mathcal{A}_6^{ex}, \mathtt{c}\rangle, \langle \mathcal{A}_1^{ex}, \sim\mathtt{c}\rangle]$

The change in the status is due to the fact that there exist two new arguments $\langle \mathcal{A}_7^{ex}, \sim\mathtt{c}\rangle = \langle\{(\sim\mathtt{c} \prec \mathtt{b}, \mathtt{f})\}, \sim\mathtt{c}\rangle$ and $\langle \mathcal{A}_8^{ex}, \mathtt{c}\rangle = \langle\{(\mathtt{c} \prec \mathtt{b})\}, \mathtt{c}\rangle$ for $u(\mathcal{P}_2)$ having $\mathtt{b}$ as strict part of the program and with the preference relation (among others) $\langle \mathcal{A}_7^{ex}, \sim\mathtt{c}\rangle \succ \langle \mathcal{A}_4^{ex}, \mathtt{c}\rangle$, $\langle \mathcal{A}_7^{ex}, \sim\mathtt{c}\rangle \succ \langle \mathcal{A}_5^{ex}, \mathtt{c}\rangle$, $\langle \mathcal{A}_7^{ex}, \sim\mathtt{c}\rangle \succ \langle \mathcal{A}_6^{ex}, \mathtt{c}\rangle$, and $\langle \mathcal{A}_7^{ex}, \sim\mathtt{c}\rangle \succ \langle \mathcal{A}_8^{ex}, \mathtt{c}\rangle$.

Then, $[\langle \mathcal{A}_7^{ex}, \sim\mathtt{c}\rangle]$ is the only acceptable argumentation line contained in the dialectical three $\mathcal{T}_{\langle \mathcal{A}_7^{ex}, \sim\mathtt{c}\rangle}$. □

Using $G(u, \mathcal{P})$ and Lemma 7.21, additional irrelevant updates for both defeasible and strict rules can be identified.

Proposition 7.24. *Let $\mathcal{P}$ be a DeLP-program and r a strict rule $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n$ or defeasible rule $\alpha_0 \prec \alpha_1, \ldots, \alpha_n$ such that $\{\alpha_0, \ldots \alpha_n\} \subseteq (Lit_{\mathcal{P}} \cap Lit_{\mathcal{P}'})$. Update $u = \pm r$ is irrelevant for $\mathcal{P}$ if α_0 does not belong to $G(u, \mathcal{P})$.*

Proof. If $u = +r$ and α_0 does not belong to $G(\mathcal{P}')$, then by Definition 7.14 it holds that $\{\alpha_1, \ldots, \alpha_n\} \not\subseteq N'$ and thus also α_0 does not belong to N'. Therefore, no argument for α_0 exists w.r.t. $\mathcal{P}'$, from which it follows that no argument exists for α_0 w.r.t. $\mathcal{P}$. Thus, every dialectical tree w.r.t. $\mathcal{P}$ is still a dialectical tree w.r.t. $\mathcal{P}'$—arguments for $\mathcal{P}$ are the same as those for $\mathcal{P}'$. Hence, the status of all the literals remains unchanged, meaning that u is irrelevant.

If $u = -r$ and α_0 does not belong to $G(\mathcal{P})$, then for each strict or defeasible rule r such that $head(r) = \alpha_0$ it holds that $\{\alpha_1, \ldots, \alpha_n\} \not\subseteq N$. Then, after removing one of these rules, α_0 will continue to not belong to N'. Therefore, dialectical trees of the program will not change after performing the update, which entails that the status of all literals is preserved. □

Proposition 7.25. *Let $\mathcal{P}$ be a DeLP-program and r a strict rule $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n$ or defeasible rule $\alpha_0 \prec \alpha_1, \ldots, \alpha_n$ such that $\{\alpha_0, \ldots \alpha_n\} \subseteq (Lit_{\mathcal{P}} \cap Lit_{\mathcal{P}'})$. Update $u = \pm r$ is irrelevant for $\mathcal{P}$ if there is α_i (with $i \in [1..n]$) such that $S_{\mathcal{P}}(\alpha_i) = \text{OUT}$ and $\alpha_i \notin \mathcal{R}(u, \mathcal{P})$.*

Proof. We now consider the case of $u = +r$ consisting of a positive update and show that the update has no effect on the way the roots of each dialectical tree are marked w.r.t. $\mathcal{P}$ and $\mathcal{P}' = u(\mathcal{P})$, which entails that the status of each literal does not change. Observe that if r does not appear in a dialectical tree for a literal α w.r.t. $\mathcal{P}'$, then the status of α does not change. Thus, in the following, we focus on those dialectical trees of $\mathcal{P}'$ where r appears in at least one of its arguments.

Let $\mathcal{T}_{\langle \mathcal{A}_j, \alpha_j \rangle}$ be a dialectical tree for determining the status of α_j w.r.t. $\mathcal{P}'$. Suppose that at level k of the tree there is an argument $\langle \mathcal{A}_k, \alpha_k \rangle$ such that r appears in $\mathcal{A}_k$, that is, β appears in $\mathcal{A}_k$. Since $S_{\mathcal{P}}(\beta) = \text{OUT}$ w.r.t. $\mathcal{P}$ and $\beta \notin \mathcal{R}(U^+, \mathcal{P})$, which in turn implies that $\beta \notin \mathcal{R}(\{u\}, \mathcal{P})$, then $S_{\mathcal{P}}(\beta) = \text{OUT}$ w.r.t. $\mathcal{P}'$ since it is preserved. Therefore, the node labeled with argument $\langle \mathcal{A}_k, \alpha_k \rangle$ is marked as DEFEATED w.r.t. $\mathcal{P}'$. Thus, the parent node of the argument $\langle \mathcal{A}_k, \alpha_k \rangle$ at level $k-1$ will continue to have the same marking assignment as in $\mathcal{P}$. In fact, if it is UNDEFEATED w.r.t. $\mathcal{P}$, it remains UNDEFEATED w.r.t. $\mathcal{P}'$ because the only change in the tree is the addition of a sub-tree rooted in $\langle \mathcal{A}_k, \alpha_k \rangle$ marked as DEFEATED. On the other hand, if the parent node of the argument $\langle \mathcal{A}_k, \alpha_k \rangle$ is marked as DEFEATED w.r.t. $\mathcal{P}$, it remains so in $\mathcal{P}'$ since it must still have a child marked as UNDEFEATED.

Assume now that we have a negative update $u = -r$. Analogously to the previous case, we prove that update $u = -r$ has no effect on the way the roots of each dialectical tree are marked w.r.t. $\mathcal{P}$ and $\mathcal{P}'$. Again, if r does not appear in a dialectical tree for a literal α w.r.t. $\mathcal{P}$, then the status of α does not change. Thus, we focus on those dialectical trees for $\mathcal{P}$ where r appears in at least one of its arguments. Let $\mathcal{T}_{\langle \mathcal{A}_j, \alpha_j \rangle}$ be a dialectical tree for determining the status of α_j in $\mathcal{P}$. Suppose that at level k of the tree there is an argument $\langle \mathcal{A}_k, \alpha_k \rangle$ such that r appears in $\mathcal{A}_k$, that is, β appears in $\mathcal{A}_k$. Since $S_{\mathcal{P}}(\beta) = \text{OUT}$ and $S_{\mathcal{P}'}(\beta) = \text{OUT}$ (as $\beta \notin \mathcal{R}(U^-, \mathcal{P})$, and thus $\beta \notin \mathcal{R}(\{u\}, \mathcal{P})$), then the node labeled with argument $\langle \mathcal{A}_k, \alpha_k \rangle$ is marked as DEFEATED w.r.t. $\mathcal{P}$, and continues to be so in $\mathcal{P}'$. In fact, the parent node of the argument $\langle \mathcal{A}_k, \alpha_k \rangle$ at level $k-1$ will continue to have the same marking assignment as in $\mathcal{P}$ since it has at least one other argument $\langle \mathcal{A}'_k, \alpha_k \rangle$ such that there exists $r' \neq r$ such that $head(r') = head(r)$ that appears in $\mathcal{A}'_k$. In particular, if the marking assignment of the parent node at level $k-1$ is UNDEFEATED it remains UNDEFEATED because the only change in the tree is the removal of a sub-tree rooted in $\langle \mathcal{A}_k, \alpha_k \rangle$ marked as DEFEATED. On the other hand, if the parent node of the argument $\langle \mathcal{A}_k, \alpha_k \rangle$ is marked as DEFEATED, it remains so since it must still have a child marked as UNDEFEATED. □

However, in many cases updates are not irrelevant, as shown in the following example.

Example 7.26. Consider again the DeLP-program $\mathcal{P}_1$ from our running example, where we have that $S_{\mathcal{P}_1}(s) = S_{\mathcal{P}_1}(t) = \text{IN}$. For update $u = +(s \leftarrow t)$, we have that $S_{u(\mathcal{P}_1)}(\sim i) = \text{IN}$, though it was UNDECIDED before performing the update. The change in the status of s is caused by the new argument $\langle \mathcal{A}_{10}, \sim i \rangle = \langle \{(\sim i \prec\!\!\!- \sim a, s)\}, \sim i \rangle$ for $u(\mathcal{P}_1)$ and $\mathcal{A}_{10}$ is preferred to all the other arguments of the form $\langle \mathcal{A}, i \rangle$. □

7.4.2 Dealing with Relevant Updates

An update is not irrelevant (or *relevant*) whenever it causes the status of at least one literal to change. To avoid wasted effort, it is in our best interest to determine—as precisely as possible—the subset of literals whose status needs to be recomputed after an update. Towards this end, we propose the concept of *influenced set*, which consists of a subset of literals that are in $\mathcal{R}(u, \mathcal{P})$ computed by using only the hyper-edges whose body does not contain an unrelated literal whose status is OUT—intuitively, the other hyper-edges can be ignored as they correspond to rules whose head does not change status.

Let $\mathit{OUT}(u, \mathcal{P}, S_\mathcal{P}) = \{\alpha \mid S_\mathcal{P}(\alpha) = \text{OUT} \wedge \alpha \notin \mathcal{R}(u, \mathcal{P})\}$ be the set of literals whose status is OUT w.r.t. $\mathcal{P}$ and such that they are *not* related w.r.t u and $\mathcal{P}$, where $u = \pm r$. The influenced set is iteratively defined as follows.

Definition 7.27 (Influenced Set). *Let $\mathcal{P}$ be a DeLP-program, $u = \pm r$, and $S_\mathcal{P}$ the status of literals w.r.t. $\mathcal{P}$, and $G(u, \mathcal{P}) = \langle N^u, H^u \rangle$.*

– $\mathcal{I}_0(u, \mathcal{P}, S_\mathcal{P}) = \begin{cases} \emptyset & \textit{if u is irrelevant for } \mathcal{P} \\ \{\textit{head}(r)\} & \textit{otherwise;} \end{cases}$

– $\mathcal{I}_{i+1}(u, \mathcal{P}, S_\mathcal{P}) = \mathcal{I}_i(u, \mathcal{P}, S_\mathcal{P}) \cup$
$\{\sim\alpha \mid \exists(\{\alpha\}, \sim\alpha, \texttt{cfl}) \in H^u \; s.t. \; \alpha \in \mathcal{I}_i(u, \mathcal{P}, S_\mathcal{P})\} \cup$
$\{\alpha \mid \exists(X, \alpha, \ell) \in H^u \; s.t. \; \ell \in \{\texttt{str}, \texttt{def}\} \wedge (X \cap \mathcal{I}_i(u, \mathcal{P}, S_\mathcal{P}) \neq \emptyset) \wedge$
$(\{\alpha, \sim\alpha\} \cap \mathit{CL}(u, \mathcal{P}) = \emptyset) \wedge (X \cap \mathit{OUT}(u, \mathcal{P}, S_\mathcal{P}) = \emptyset)\} \cup$
$\{\alpha \mid \exists(X, \beta, \texttt{str}) \in H^u \; s.t. \; \beta \in \mathcal{I}_i(u, \mathcal{P}, S_\mathcal{P}) \wedge (\alpha \in X) \wedge$
$(\{\alpha, \sim\alpha\} \cap \mathit{CL}(u, \mathcal{P}) = \emptyset) \wedge (X \cap \mathit{OUT}(u, \mathcal{P}, S_\mathcal{P}) = \emptyset)\}.$

The influenced set for u w.r.t. $\mathcal{P}$ and $S_\mathcal{P}$ is then defined as $\mathcal{I}(u, \mathcal{P}, S_\mathcal{P}) = \mathcal{I}_n(u, \mathcal{P}, S_\mathcal{P})$ such that $\mathcal{I}_n(u, \mathcal{P}, S_\mathcal{P}) = \mathcal{I}_{n+1}(u, \mathcal{P}, S_\mathcal{P})$.

Example 7.28. Consider the DeLP-program $\mathcal{P}_1$ of Example 7.1 and the update $u = +(a \prec\!\!\!- s)$, which yields the DeLP-program $u(\mathcal{P}_1)$. The rule addition update u is not irrelevant: Proposition 7.22 does not apply, as the $S_{\mathcal{P}_1}(a) = \text{OUT}$, Proposition 7.24 does not apply, as literal a belongs to $G(u, \mathcal{P}_1)$, while Proposition 7.25 does not apply since $S_{\mathcal{P}_1}(s) = \text{IN}$.

Thus, we have $\mathcal{I}_0(u, \mathcal{P}_1, S_{\mathcal{P}_1}) = \{a\}$. Moreover, $\mathit{OUT}(+(a \prec\!\!\!- s), \mathcal{P}_1, S_{\mathcal{P}_1}) = \{\sim h\}$ Hence, since $\sim a \in \mathit{CL}(u, \mathcal{P})$, $\mathcal{I}_1(u, \mathcal{P}_2, S_{\mathcal{P}_1}) = \mathcal{I}_0(u, \mathcal{P}_1, S_{\mathcal{P}_1}) = \{a\}$, and $\mathcal{I}(u, \mathcal{P}_1, S_{\mathcal{P}_1}) = \{a\}$. Supposing that $\sim a \notin \mathit{CL}(u, \mathcal{P})$ yielding $\mathcal{I}_1(u, \mathcal{P}_2, S_{\mathcal{P}_1}) =$

$\{a, \sim a\}$, observe that, although both $\sim e$ and $\sim f$ are reachable from $\sim a$ in $G(u, \mathcal{P}_1)$ (and thus they are related literals too) through the (hyper-)edge $(\{\sim h, \sim a\}, \sim e)$, $\sim e$ does not belong to $\mathcal{I}_2(u, \mathcal{P}_1, S_{\mathcal{P}_1})$ because $\sim h \in \mathit{OUT}(+(a \multimap\!\prec s), \mathcal{P}_1, S_{\mathcal{P}_1})$. □

The notion of influenced set is conceptually similar to the influenced set introduced in Chapter 4 in the context of abstract argumentation [61]—where arguments have no internal structure—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 incremental computation of the status of *structured* arguments, and devised a notion of influenced set w.r.t. an update for a DeLP-program and its status that we then apply to (hyper-)graphs representing DeLP-programs, from which structured arguments are derived.

The following theorem strengthens the result of Lemma 7.21 by identifying a possibly smaller subset of literals whose status does not change after performing an update.

Theorem 7.29 (Influenced and Preserved Literals). *Let $\mathcal{P}$ be a DeLP-program, $u = \pm r$, $\mathcal{P}' = u(\mathcal{P})$, and $G(\mathcal{P}') = \langle N', H' \rangle$ the updated hyper-graph. Then, a literal $\alpha \in N'$ is preserved (i.e., $S_{\mathcal{P}}(\alpha) = S_{\mathcal{P}'}(\alpha)$) if $\alpha \notin \mathcal{I}(u, \mathcal{P}, S_{\mathcal{P}})$.*

Proof. Let $\mathcal{P} = (\Pi, \Delta)$, and let $\mathcal{P}^*$ be the DeLP program obtained from $\mathcal{P}$ by applying all the updates $u = -r'$ where $r' = \alpha_0 \multimap\!\prec \alpha_1, \ldots, \alpha_n \in \Delta$ and r' is such that (i) $\alpha_0 \in \mathcal{R}(u, \mathcal{P})$, and (ii) there is α_i (with $i \in [1..n]$) such that $S_{\mathcal{P}}(\alpha_i) = \text{OUT}$ and $\alpha_i \notin \mathcal{R}(u, \mathcal{P})$. Since $\alpha_0 \in \mathcal{R}(u, \mathcal{P})$ and $\alpha_i \notin \mathcal{R}(u, \mathcal{P})$, then $\alpha_i \notin \mathcal{R}_{[\alpha_0]}(u, \mathcal{P})$ where $\mathcal{R}_{[\alpha_0]}(u, \mathcal{P})$ is computed as in Definition 7.20 but replacing $head(r)$ with α_0. Therefore, $\alpha_i \notin \mathcal{R}^*_{[\alpha_0]}(u, \mathcal{P})$ and the same holds for any sub-graph $G(\mathcal{P}')$ of $G(\mathcal{P})$ where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by removing some of the rules having the properties of r'. Hence, updates $-r'$ are removal updates of the form of Proposition 7.25, and thus all of them are irrelevant. Therefore, $\forall \alpha \in Lit_{\mathcal{P}^*}$ it holds that $S_{\mathcal{P}^*}(\alpha) = S_{\mathcal{P}}(\alpha)$.

Let $\mathcal{P}^{**}$ be the DeLP program obtained from $\mathcal{P}^*$ by applying all the updates $u = r''$ where $r'' = \alpha_0 \multimap\!\prec \alpha_1, \ldots, \alpha_n \in \Delta^*$ and r'' is such that $\{\alpha_0, \sim \alpha_0\} \cap CL(u, \mathcal{P}^*) \neq \emptyset$. We obtain that the status of all literals $\alpha \in Lit_{\mathcal{P}^*}$ is the same w.r.t $\mathcal{P}^{**}$ (i.e., $\forall \alpha \in Lit_{\mathcal{P}^{**}}, S_{\mathcal{P}^{**}}(\alpha) = S_{\mathcal{P}^*}(\alpha)$). This is proved by noting that if $\alpha_0 \in CL(u, \mathcal{P}^*)$ then i) no argument for $\sim \alpha_0$ exists and ii) it means that $S_{\mathcal{P}^*}(\alpha_0) = S_{\mathcal{P}^{**}}(\alpha_0) = \text{IN}$. Any argument $\langle \mathcal{A}, \alpha \rangle$ in $\mathcal{P}^*$ s.t. α_0 is used in at least one defeasible rule of $\mathcal{A}$, will continue to be also an argument for $\mathcal{P}^{**}$. Obviously, any argument $\langle \mathcal{A}, \alpha \rangle$ in $\mathcal{P}^*$ s.t. α_0 is not used in at least one defeasible rule of $\mathcal{A}$, will continue to be also an argument for $\mathcal{P}^{**}$. Thus, no dialectical tree will change, meaning that $S_{\mathcal{P}^*}(\alpha) = S_{\mathcal{P}^{**}}(\alpha) \forall \alpha \in Lit_{\mathcal{P}^{**}}$. The same holds for the case where $\sim\alpha_0 \in CL(u, \mathcal{P}^*)$. Now consider the hyper-graph $G(u, \mathcal{P}^{**})$. Using Definition 7.27, with a little effort, it can be checked that $\mathcal{I}(u, \mathcal{P}, S_{\mathcal{P}})$ coincides with $\mathcal{R}(u, \mathcal{P}^{**})$. Therefore, by applying Lemma 7.21, the statement follows. □

Using this result, we can determine a set of literals whose status need not be recomputed after performing a relevant update—though this is not guaranteed to be a comprehensive set, it will certainly save us from a non-trivial amount of wasted effort, as we will show empirically below.

Given the updated hyper-graph $G(\mathcal{P}') = \langle N', H' \rangle$, we use $PR = \{\alpha \in N' \setminus \mathcal{I}(u, \mathcal{P}, S_{\mathcal{P}})\}$ to denote the set of literals of $\mathcal{P}'$ that are guaranteed to be preserved; the set of literals whose status needs to be recomputed is thus $S = Lit \setminus PR$. However, as we show next, the status of some literals in S can be derived from that of (complementary) literals in S, and thus we only recompute the status of the latter, which we call *core literals*.

Inferable and Core Literals. 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 hyper-graph of updated programs, we can define inferable literals as follows.

Definition 7.30 (Set of Inferable Literals). *Let $\mathcal{P}$ be a DeLP-program, $u = \pm r$, $\mathcal{P}' = u(\mathcal{P})$, and $G(\mathcal{P}') = \langle N', H' \rangle$. The set of inferable literals for u w.r.t. $\mathcal{P}$ is* ***Infer***$(u, \mathcal{P}) = Lit_{\mathcal{P}'} \setminus N'$.

Example 7.31. Consider $\mathcal{P}_1$ from Example 7.1, whose hyper-graph is shown in Figure 7.1(a) where $\leftrightarrow$ (resp. $\lhd\!\!-$ and $\blacktriangleleft\!\!-$) denotes hyper-edges labelled as `cfl` (resp. `def` and `str`), and the update $u = -(d \leftarrow t)$ that yields program $u(\mathcal{P}_1)$, whose hyper-graph is shown in Figure 7.1(b). Then, ***Infer***$(u, \mathcal{P}_1) = \{d, {\sim}h, {\sim}e, {\sim}f\}$. □

The core literals for a relevant update $u = \pm r$ w.r.t. $\mathcal{P}$ are those in $Lit_{\mathcal{P}'}$ that are influenced but are not inferable.

Definition 7.32 (Set of Core Literals). *Let $\mathcal{P}$ be a DeLP-program, $u = \pm r$, and $S_{\mathcal{P}}$ the status of the literals of $\mathcal{P}$. The set* ***Core***$(u, \mathcal{P})$ *of core literals for u w.r.t. $\mathcal{P}$ is* ***Core***$(u, \mathcal{P})) = (\mathcal{I}(u, \mathcal{P}, S_{\mathcal{P}}) \setminus$ ***Infer***$(u, \mathcal{P})) \cap Lit_{\mathcal{P}'}$.

Observe that if the update is irrelevant, then ***Core***$(u, \mathcal{P}) = \emptyset$, and so is the influenced set.

Example 7.33. Continuing from Example 7.31, we have that ***Core***$(u, \mathcal{P}_1) = (\{h, s, {\sim}i, i, d, {\sim}h, {\sim}e, {\sim}f\} \setminus \{d, {\sim}h, {\sim}e, {\sim}f\}) \cap Lit_{\mathcal{P}'} = \{h, s, {\sim}i, i\}$. □

The relationship between inferable and core 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 literal that is IN.

Theorem 7.34 (Status of Inferable Literals). *Let $\mathcal{P}$ be a DeLP-program, $u = \pm(r)$ an update, and $\mathcal{P}' = u(\mathcal{P})$. For each literal $\alpha \in$* ***Infer***$(u, \mathcal{P})$*, it holds that $S_{\mathcal{P}'}(\alpha) =$ OUT iff ${\sim}\alpha \in$ ***Core***$(u, \mathcal{P}, S_{\mathcal{P}})$ and $S_{\mathcal{P}'}({\sim}\alpha) =$ IN; otherwise $S_{\mathcal{P}'}(\alpha) =$ UNDECIDED.*

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 consider separately the cases where ${\sim}\alpha$ either belongs to ***Core***$(u, \mathcal{P})$ or not, and whether ${\sim}\alpha$ is preserved.

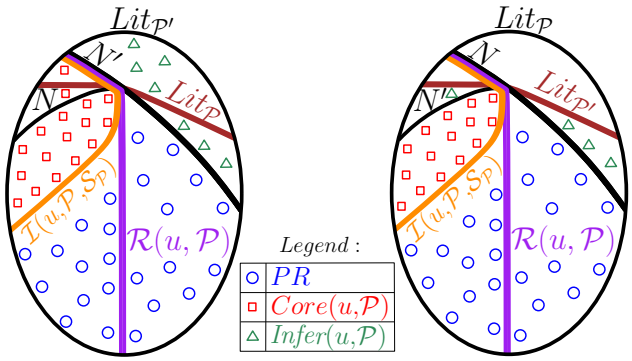


Fig. 7.2: Relationship between sets of literals Lit_P, Lit'_P, N, N', $\mathcal{R}$, *PR*, *Infer*, $\mathcal{I}$ and *Core* for addition (a) and removal (b) case.

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 $S_{\mathcal{P}'}(\sim\alpha)$: $S_{\mathcal{P}'}(\alpha) =$ OUT if $S_{\mathcal{P}'}(\sim\alpha) =$ IN, while $S_{\mathcal{P}'}(\alpha) =$ UNDECIDED if $S_{\mathcal{P}'}(\sim\alpha) =$ UNDECIDED.

If $\sim\alpha \notin$ *Core*$(u,\mathcal{P})$ and $\sim\alpha$ is not preserved, then $\sim\alpha \in$ *Infer*$(u,\mathcal{P})$, and thus arguments cannot be built either for α or for $\sim\alpha$. Therefore, $S_{\mathcal{P}'}(\alpha) =$ UNDECIDED if $S_{\mathcal{P}'}(\sim\alpha) =$ UNDECIDED. □

Example 7.35. Continuing from Examples 7.31 and 7.33, Theorem 7.34 tells us that $S_{\mathcal{P}'_1}(\sim h) =$ OUT since $S_{\mathcal{P}'_1}(h) =$ IN and $h \in$ *Core*$(u,\mathcal{P}_1,S_{\mathcal{P}_1})$. Also, we have that $S_{\mathcal{P}'_1}(d) = S_{\mathcal{P}'_1}(\sim e) = S_{\mathcal{P}'_1}(\sim f) =$ UNDECIDED. □

Figure 7.2 and Table 7.1 report the relationship between the different sets of literals considered in this section. Note that, for the rule addition case, *Core*$(u,\mathcal{P},S_{\mathcal{P}})$ coincides with the influenced set since *Infer*$(u,\mathcal{P})\cap N' = \emptyset$, while *Core*$(u,\mathcal{P},\mathcal{S}_{\mathcal{P}}) \subseteq \mathcal{I}(u,\mathcal{P},S_{\mathcal{P}})$ when u is a rule removal since $Lit_{\mathcal{P}'} \setminus N$ and $Lit_{\mathcal{P}'} \setminus N'$ (the inferable set) may both be non-empty, and the latter intersects the influenced set.

7.4.3 Incremental Algorithm

Algorithm 6 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 at Line 1. Then it checks if the update belongs to the cases identified in Propositions 7.22–7.25 (and it is thus irrelevant) and, if so, immediately returns the updated status that coincides with the initial one (Line 3). The next check is for literals not occurring in the input program, for which we compute the (updated) status w.r.t. $\mathcal{P}'$ by calling the DeLP-Solver[2]

[2] See `http://lidia.cs.uns.edu.ar/delp_client/index.php`.

Sets	$Lit_{\mathcal{P}}$		$Lit_{\mathcal{P}'}$		N		N'		$\mathcal{R}$		PR		$\mathcal{I}$		*Infer*		*Core*	
	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−
$Lit_{\mathcal{P}}$	=	=																
$Lit_{\mathcal{P}'}$	$\supseteq$	$\subseteq$	=	=														
N	$\subseteq$	$\subseteq$	$\subseteq$		=	=												
N'		$\subseteq$	$\subseteq$	$\subseteq$	$\supseteq$	$\subseteq$	=	=										
$\mathcal{R}$		$\subseteq$	$\subseteq$			$\subseteq$	$\subseteq$		=	=								
PR	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$		$\subseteq$	$\subseteq$	$\subseteq$			=	=						
$\mathcal{I}$		$\subseteq$	$\subseteq$			$\subseteq$	$\subseteq$		$\subseteq$	$\subseteq$	$\not\subseteq$	$\not\subseteq$	=	=				
Infer		$\subseteq$	$\subseteq$	$\subseteq$	$\not\subseteq$		$\not\subseteq$	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$	=	=		
$Core$		$\subseteq$	$\subseteq$	$\subseteq$		$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\subseteq$	$\not\subseteq$	$\not\subseteq$	=	$\subseteq$	$\not\subseteq$	$\not\subseteq$	=	=

Table 7.1: Relationship between sets of literals $Lit_{\mathcal{P}}$, $Lit'_{\mathcal{P}}$, N, N' $\mathcal{R}$, *PR*, *Infer*, $\mathcal{I}$ and *Core* for addition (+) and removal (–) case. Symbol $\not\subseteq$ denotes disjoint sets, while blank means that the relationship between the sets depends on the input program.

Algorithm 6 Dynamic DeLP-Solver

Input: DeLP-program $\mathcal{P}$, Initial status $S_{\mathcal{P}}$, Update $u = \pm r$.
Output: Status $S_{\mathcal{P}'}$ w.r.t. the updated program $\mathcal{P}' = u(\mathcal{P})$.

1: $S_{\mathcal{P}'}(\alpha) = S_{\mathcal{P}}(\alpha)\ \ \forall \alpha \in Lit_{\mathcal{P}}$;
2: **if** one of Propositions 7.22–7.25 holds (the update is irrelevant) **then**
3: **return** $S_{\mathcal{P}'}$;
4: **if** $\{head(r), \sim head(r)\} \cap Lit_{\mathcal{P}} = \emptyset$ **then**
5: $S_{\mathcal{P}'}(head(r)) \leftarrow$ DELP-SOLVER$(\mathcal{P}', head(r))$
6: **return** $S_{\mathcal{P}'}$
7: **for** $\alpha \in Core(u, \mathcal{P})$ **do**
8: $S_{\mathcal{P}'}(\alpha) \leftarrow$ DELP-SOLVER$(\mathcal{P}', \alpha)$;
9: **for** $\alpha \in Infer(u, \mathcal{P})$ **do**
10: **if** $S_{\mathcal{P}'}(\sim\alpha) =$ IN
11: **then** $S_{\mathcal{P}'}(\alpha) \leftarrow$ OUT
12: **else** $S_{\mathcal{P}'}(\alpha) \leftarrow$ UNDECIDED;
13: **for** $\alpha \in Lit \setminus Lit_{\mathcal{P}'}$ **do**
14: $S_{\mathcal{P}'}(\alpha) =$ UNDECIDED
15: **return** $S_{\mathcal{P}'}$.

with input $\mathcal{P}'$ and the new literal $head(r)$, and return the initial status updated by changing only that of $head(r)$ (Line 6).

Otherwise, the status of the literals is updated by first calling the solver at Lines 7-8 for the core literals, and then using Theorem 7.34 to derive the status of inferable literals in Lines 9-12. 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 result states that this algorithm is sound and complete.

Theorem 7.36. *Let $\mathcal{P}$ be a DeLP-program, $u = \pm(r)$, and $\mathcal{P}' = u(\mathcal{P})$. For each $\alpha \in Lit$, Algorithm 6 returns $S_{\mathcal{P}'}(\alpha)$.*

Proof. Soundness follows from the fact that Algorithm 6 returns the right status $S_{\mathcal{P}'}(\alpha)$. Specifically, if the update is irrelevant then, using the results of Propositions 7.22–7.24, $S_{\mathcal{P}'}(\alpha) = S_{\mathcal{P}}(\alpha)$ for each $\alpha \in Lit$ is returned at Line 3.

Otherwise, the update is relevant, and two cases are considered. First, if the literal *head*(r) contained in the head of the rule r as well as its complementary literal $\sim$ *head*(r) do not appear in $Lit_{\mathcal{P}}$, 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 status of all literals of the initial DeLP program $\mathcal{P}$ carry over to $\mathcal{P}'$—the same marked dialectical trees can be used to obtain the status w.r.t. $\mathcal{P}$ and $\mathcal{P}'$ for all literals in $Lit_{\mathcal{P}}$. Therefore, the status of *head*(r) is computed in Line 5 and returned along with the previous status for the other literals in Lit in Line 6.

The second case we need to consider is when the update is relevant and the head of rule r or its complementary literal are in the language of the program $\mathcal{P}$. Using Theorem 7.29, the status of preserved literals PR does not change, and it is computed in Lines 1 by copying it from the initial status. Next, the status of core literals is computed from scratch in Lines 7-8, and the status of inferable literals is computed using the result of Theorem 7.34. Finally, in Lines 11-12 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 easily follows from the fact that $Lit_{\mathcal{P}'} = head(r) \cup PR \cup Core(u, \mathcal{P}) \cup$ *Infer*$(u, \mathcal{P})$, and the status of all these literals is computed by Algorithm 6. Moreover, the status of literals in $Lit \setminus Lit_{\mathcal{P}'}$ are assigned UNDECIDED. □

It is easy to see that the time complexity of Algorithm 6 is dominated by $O(|$*Core*$(u, \mathcal{P})| \times D(\mathcal{P}, \alpha))$, where $D(\mathcal{P}, \alpha)$ is the cost of a DeLP-solver call for computing the status of a literal α w.r.t. $\mathcal{P}$—Corollary 7.19 entails that solving this problem is NP-hard in general. Thus, theoretically, Algorithm 6 has the same worst-case complexity as total recomputation, as *Core*$(u, \mathcal{P})$ may consist of the whole set of literals whose status need to be recomputed (see Figure 7.2). However, as we experimentally show in the next section, in practice the incremental approach turns out to be much more efficient than the recomputing everything from scratch.

7.5 Implementation and Experiments

We report on a set of experiments designed to compare Algorithm 6 against full 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 Algorithm 6.

Dataset. Inspired by the structure of the DeLP-program in our running example, we generated a set of 40 DeLP-programs, each consisting of a number of literals in $[180, 220]$, of facts in $[10, 20]$, of strict rules in $[20, 30]$, and a number of defeasible

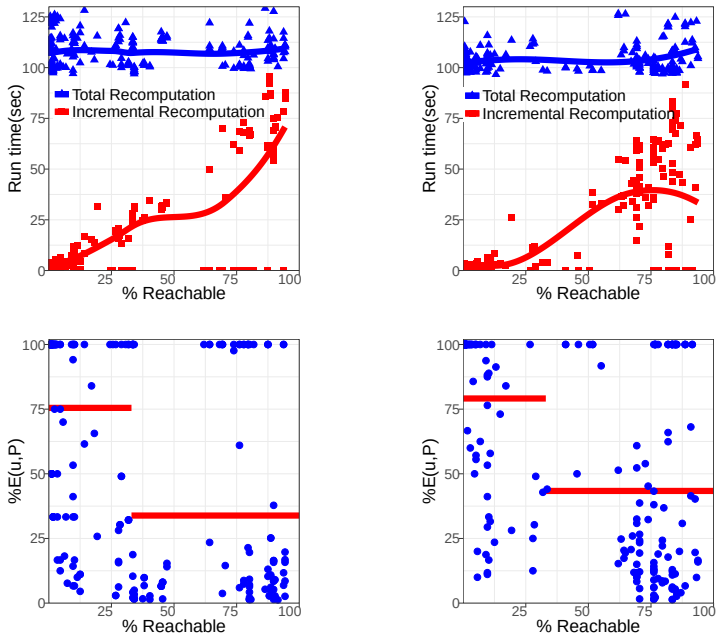


Fig. 7.3: Running time (top) and effectiveness (bottom) for rule addition (left-hand side) and deletion (right-hand side) cases.

rules in $[100, 150]$. More in detail, benchmark DeLP-programs were generated by a Python prototype as follows. First, numbers n, f, s, and d were randomly chosen in the intervals specified above for the number of literals, facts, strict and defeasible rules, respectively. Then, each DeLP-program was iteratively built by using every fact for defining the body of at least one (strict/defeasible) rule. Rule bodies were randomly generated consisting of up to 4 distinct and non-contradictory literals; rule heads were generated selecting literals not contradicting the body's literals, and also avoiding the generation of rules that would make the resulting program contradictory. Given a generated DeLP-program, rule removal updates were obtained by randomly selecting a rule in the program; rule addition updates were generated by using the same procedure from above for generating a rule during the construction of the benchmark program. For each program, we generated 5 different rule addition/deletion updates.

As far as we know, this is one of the first attempts to produce a benchmark for structured argumentation, along with [59] where benchmarks were generated for ABA [110].

Efficiency. Figure 7.3 (top) shows the running times for computing the status of all the literals in DeLP-programs after adding (left-hand side) or removing (right-hand side) a rule versus the percentage p of literals that are reachable from the head of an update—for the sake of readability, besides data points, figures also show lines

obtained by LOESS local regression. The incremental computation outperforms full recomputation from scratch: for percentages of reachability less than $p \leq 15\%$ the incremental approach takes only 2 seconds on average, while full recomputation takes almost 2 minutes (irrespectively of p). Algorithm 6 performs better in the rule removal case, compared with the rule addition case, as the status of a larger set of literals can be inferred from that of core literals for that scenario, saving part of the hard computation. Finally, the experiments also showed that, for both addition and removal updates, incremental computation remains on average quite faster than total recomputation, even when p is high. This is due to the fact that on average only 19% of all the literals turn out to be core literals, which are those whose status actually needs to be recomputed—this aspect is analyzed further in what follows.

Effectiveness. Given an update u and a program $\mathcal{P}$, effectiveness $E(u, \mathcal{P})$ is 100% if one of the conditions in Propositions 7.22–7.25 applies—that is, if we are able to recognize that u is irrelevant. Otherwise (either u is relevant or we are not able to recognize that it is irrelevant), let $M(u, \mathcal{P})$ be the set of literals whose status *needs* to be recomputed, that is, $M(u, \mathcal{P})$ is the set of literals α such that either (*i*) $S_{\mathcal{P}'}(\alpha) \neq S_{\mathcal{P}}(\alpha)$ or (*ii*) $S_{\mathcal{P}'}(\alpha) = S_{\mathcal{P}}(\alpha)$ and α is associated with different (marked) dialectical trees in $\mathcal{P}$ and $\mathcal{P}'$. Then, $E(u, \mathcal{P}) = \frac{|M(u,\mathcal{P})|}{|\mathit{Core}(u,\mathcal{P}) \cup \mathit{Infer}(u,\mathcal{P})|}$ is the percentage of literals whose status needs to be recomputed over the set of literals whose status is recomputed by Algorithm 6. Given that $M(u, \mathcal{P}) \subseteq (\mathit{Core}(u, \mathcal{P}) \cup \mathit{Infer}(u, \mathcal{P}))$, we have that as the difference between $\mathit{Core}(u, \mathcal{P}) \cup \mathit{Infer}(u, \mathcal{P})$ and $M(u, \mathcal{P})$ becomes smaller, $E(u, \mathcal{P})$ increases and the technique is more accurate.

Figure 7.3 (bottom) shows $E(u, \mathcal{P})$ versus p for rule addition and deletion updates. The average effectiveness is about 85% (resp., 86%) in the rule addition (resp., deletion) case for $p \leq 10\%$, meaning that only 15% (resp., 14%) of the recomputation was unnecessary. The average effectiveness remains quite high (more than 75%) for $p \leq 33\%$—see the red segment on the left-hand side of each figure reporting effectiveness. For $p > 33\%$, the average wasted effort was larger, though still resulting in lower running times.

Moreover, the experiments showed that, for almost half of the updates performed, the proposed technique computes only the status of literals whose status actually needs to be recomputed, thus fully avoiding wasted effort during the recomputation. In fact, $E(u, \mathcal{P}) = 100\%$ for 47% and 42% of the rule addition and deletion updates performed, respectively. More in detail, it turned out that 20% and 17% of the rule addition and deletion updates, respectively, were recognized as irrelevant—this means that Propositions 7.22–7.25 were applied in 43% and 40% of the updates for which $E(u, \mathcal{P}) = 100\%$.

Overall, the experiments showed 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 in any case faster than total recomputation—i.e., the overhead of applying the technique always pays off.

7.6 Summary

Overviews of key concepts in argumentation theory and formal models of argumentation can be found in [12, 25, 30, 102]. Further discussion regarding the uses of computational argumentation as an Agreement Technology can be found in Modgil et al. [97].

Several significant efforts dealing with dynamic aspects of abstract argumentation frameworks have been developed (e.g., [20, 49, 99]), where arguments are abstract entities with no internal structure [61]. The techniques proposed in [19, 89] and [2, 3, 4, 81, 82] are related to the proposed 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). On the other hand, here we apply the concept of reachability for (hyper-)graphs representing DeLP-programs from which structured arguments are derived. None of the above-mentioned works deal with structured arguments.

Although the technique we propose is related to incremental approaches for abstract argumentation discussed in previous chapters, 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 [42], the addition of an attack consists in modifying the body of a rule, whereas in the considered 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.

As in the abstract argumentation case, there have been some works following the belief revision approach. In [70], the issue of modifying strict rules to become defeasible was analyzed in the context of revisions effected over a knowledge base, while in [98] 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 [60], 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, [80] make use of heuristics designed to improve efficiency. None of these approaches have a deep connection with this work.

We have taken the first steps in tackling the problem of avoiding wasted effort when determining the warrant status of literals in a DeLP program after it is changed. We identified certain conditions under which 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 set of experiments—to yield significantly lower running times in practice.

The analysis in a concrete formalism like DeLP is less difficult conceptually because the internal structure of arguments is readily available, and thus one can leverage this instead of reasoning about the relationships between abstract arguments. However, we believe the basic ideas in the framework could carry over to other frameworks, v.g. ASPIC+, ABA. Although Algorithm 6 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, we are now currently working on extending the technique to deal with sets of updates to be performed simultaneously.

Conclusions and Future Work

This thesis mainly focused on advancing techniques for efficiently solving the problem of computing extensions of several kinds of argumentation frameworks. In the static setting, i.e. when the represented knowledge does not change over the time, an efficient algorithm is proposed to determine the set of both preferred and semi-stable extensions exploiting properties of the considered semantics. This layed the foundation to new algorithms for improving the efficiency of state-of-the-art solvers returning extensions under different semantics. Therefore, an interesting challenge is to find similar techniques for other multiple-status semantics (i.e. complete, stable, etc..).

Analogously, in the dynamic setting, an incremental algorithm taking as input an initial extension and a change (called update) of the argumentation graph, and returning an extension of the updated framework under the most popular argumentation semantics is presented. The experimental evaluation showed that, independently of the argumentation framework adopted, using such incremental algorithms yields lower running times for computing an extension. Although argumentation is an inherently dynamic process, few works were proposed in literature to deal with dynamic argumentation. This topic layed the groundwork to launch a new specific competition [33], where solvers are required to incrementally compute extensions after changes. It would be interesting to find out what happens if these solvers incorporate theoretical results presented in this thesis. Future work will be devoted to both (i) applying our technique to other argumentation semantics and (ii) extending it to cope with other computational problems, such as enumerating all the extensions and deciding credulous acceptance in dynamic argumentation frameworks.

By exploiting the incremental algorithm proposed in Chapter 4, we also introduced a technique for the incremental computation of extensions of dynamic EAFs, i.e., BAFs (possibly) incorporating second-order attacks. Additionally, there could be the possibility to discover similar techniques when a set of updates is performed over an EAF. For this purpose, the construction described in Section 2.4 for reducing the application of a set of updates to the application of a single attack update could be extended to deal with multiple updates for EAFs. In this perspective, another interesting direction for future work is to extend our technique to deal with other

interpretations of support, particularly the approach in [51, 52] where meta-AFs are also adopted to cope with bipolarity in argumentation. Also, tackling the problem of recomputing the skeptical acceptance in the context of BAFs would complete the picture. For this scope, it can be investigated an approach that combines the translation into a meta-AF with the incremental technique proposed in Chapter 6.

In this thesis, it is also addressed for the first time the problem of finding an efficient technique for the incremental computation of skeptical acceptance in dynamic AFs. Finally, it is also presented an algorithm able to efficiently solve 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. However, we believe the basic ideas in the framework could carry over to other frameworks, e.g. ASPIC+ or ABA. This is a direction to take in future work.

For all the addressed problems, experiments confirmed the efficiency of techniques, showing that them outperform state-of-the-art approaches/algorithms.

Acknowledgement

I would like to thank my supervisors Sergio Greco and Francesco Parisi. I believe that a key element in obtaining a PhD is to find the right supervisors, and I feel that I could not have been luckier in this regard. They are excellent supervisors without whose guidance this work would have been impossible. I furthermore thanks all thesis reviewers for precious comments/suggestions. I had the opportunity to discuss ideas with many colleagues, both inside and outside the University of Calabria. Some of these discussions lead to joint works, for which I want to thank Guillermo Ricardo Simari, Gerardo Ignacio Simari, Andrea Cohen and Sebastian Gottifredi. I want to thank all my colleagues for the great times, both during and after working hours. I would like to thank my family for their support, as well as my friends. Finally, I would like to thank Francesca, who has had to pay for this thesis with many lonely weekends, for her love, support and patience.

References

[1] http://argumentationcompetition.org/2017/results.html, 2017.

[2] Gianvincenzo Alfano, Sergio Greco, and Francesco Parisi. Computing stable and preferred extensions of dynamic bipolar argumentation frameworks. In *Proc. of Workshop on Advances In Argumentation In Artificial Intelligence co-located with XVI International Conference of the Italian Association for Artificial Intelligence (AI*IA)*, pages 28–42, 2017.

[3] Gianvincenzo Alfano, Sergio Greco, and Francesco Parisi. Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 49–55, 2017.

[4] Gianvincenzo Alfano, Sergio Greco, and Francesco Parisi. Computing extensions of dynamic abstract argumentation frameworks with second-order attacks. In *IDEAS*, pages 183–192, 2018.

[5] Gianvincenzo Alfano, Sergio Greco, and Francesco Parisi. A meta-argumentation approach for the efficient computation of stable and preferred extensions in dynamic bipolar argumentation frameworks. *Intelligenza Artificiale*, 12(2):193–211, 2018.

[6] Teresa Alsinet, Josep Argelich, Ramón Béjar, César Fernández, Carles Mateu, and Jordi Planes. An argumentative approach for discovering relevant opinions in twitter with probabilistic valued relationships. *Pattern Recognition Letters*, In press, 2017.

[7] Mario Alviano. The pyglaf argumentation reasoner. In *ICLP*, pages 2:1–2:3, 2017.

[8] Leila Amgoud, Jean-François Bonnefon, and Henri Prade. An argumentation-based approach to multiple criteria decision. In *Proc. of European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, pages 269–280, 2005.

[9] Leila Amgoud, Claudette Cayrol, and Marie-Christine Lagasquie-Schiex. On the bipolarity in argumentation frameworks. In *Proc. of International Workshop on Non-Monotonic Reasoning (NMR)*, pages 1–9, 2004.

[10] Leila Amgoud and Henri Prade. Using arguments for making and explaining decisions. *Artif. Intell.*, 173(3-4):413–436, 2009.

[11] Leila Amgoud and Srdjan Vesic. Revising option status in argument-based decision systems. *Journal of Logic and Computation*, 22(5):1019–1058, 2012.

[12] Katie Atkinson, Pietro Baroni, Massimiliano Giacomin, Anthony Hunter, Henry Prakken, Chris Reed, Guillermo R. Simari, Matthias Thimm, and Serena Villata. Towards artificial argumentation. *AI Magazine*, 38(3):25–36, 2017.

[13] Pietro Baroni, Guido Boella, Federico Cerutti, Massimiliano Giacomin, Leendert W. N. van der Torre, and Serena Villata. On the input/output behavior of argumentation frameworks. *Artificial Intelligence*, 217:144–197, 2014.

[14] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26(4):365–410, 2011.

[15] Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Giovanni Guida. Encompassing attacks to attacks in abstract argumentation frameworks. In *Proc. of European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 83–94, 2009.

[16] Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Giovanni Guida. AFRA: argumentation framework with recursive attacks. *Int. J. Approx. Reasoning*, 52(1):19–37, 2011.

[17] Pietro Baroni and Massimiliano Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artif. Intell.*, 171(10-15):675–700, 2007.

[18] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. Scc-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168(1-2):162–210, 2005.

[19] Pietro Baroni, Massimiliano Giacomin, and Beishui Liao. On topology-related properties of abstract argumentation semantics. A correction and extension to dynamics of argumentation systems: A division-based method. *Artificial Intelligence*, 212:104–115, 2014.

[20] Ringo Baumann. Splitting an argumentation framework. In *Proc. of International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR)*, pages 40–53, 2011.

[21] Ringo Baumann. Normal and strong expansion equivalence for argumentation frameworks. *Artificial Intelligence*, 193:18–44, 2012.

[22] Ringo Baumann. Context-free and context-sensitive kernels: Update and deletion equivalence in abstract argumentation. In *Proc. of European Conference on Artificial Intelligence (ECAI)*, pages 63–68, 2014.

[23] Ringo Baumann and Gerhard Brewka. Expanding argumentation frameworks: Enforcing and monotonicity results. In *Proc. of Third International Conference on Computational Models of Argument (COMMA)*, pages 75–86, 2010.

[24] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in AI and law: Editors' introduction. *Artif. Intell. Law*, 13(1):1–8, 2005.

[25] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10 - 15):619 – 641, 2007.

[26] Trevor J. M. Bench-Capon, Henry Prakken, and Giovanni Sartor. Argumentation in legal reasoning. In *Argumentation in Artificial Intelligence*, pages 363–382. 2009.

[27] P. Besnard, A. J. Garcia, A. Hunter, S. Modgil, H. Prakken, G. R. Simari, and F. Toni. Introduction to structured argumentation. *Argument & Computation – Special Issue: Tutorials on Structured Argumentation*, 5(1):1–4, 2014.

[28] P. Besnard and A. Hunter. Constructing argument graphs with deductive arguments: A tutorial. *Argument & Computation*, 5(1):5–30, 2014.

[29] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.

[30] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.

[31] Philippe Besnard and Anthony Hunter. Argumentation based on classical logic. In *Argumentation in Artificial Intelligence*, pages 133–152. 2009.

[32] Pierre Bisquert, Claudette Cayrol, Florence Dupin de Saint-Cyr, and Marie-Christine Lagasquie-Schiex. Characterizing change in abstract argumentation systems. In *Trends in Belief Revision and Argumentation Dynamics*, volume 48, pages 75–102. 2013.

[33] Stefano Bistarelli, Lars Kotthoff, Francesco Santini, and Carlo Taticchi. Containerisation and dynamic frameworks in iccma'19. In *Proceedings of the Second International Workshop on Systems and Algorithms for Formal Argumentation (SAFA 2018) co-located with the 7th International Conference on Computational Models of Argument (COMMA 2018), Warsaw, Poland, September 11, 2018.*, pages 4–9, 2018.

[34] J Anthony Blair and Ralph H Johnson. The current state of informal logic. *Informal Logic*, 9(2), 1987.

[35] Bernhard Bliem, Markus Hecher, and Stefan Woltran. On efficiently enumerating semi-stable extensions via dynamic programming on tree decompositions. In *Proc. of COMMA*, pages 107–118, 2016.

[36] Guido Boella, Dov M. Gabbay, Leendert W. N. van der Torre, and Serena Villata. Support in abstract argumentation. In *Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010.*, pages 111–122, 2010.

[37] Guido Boella, Souhila Kaci, and Leendert W. N. van der Torre. Dynamics in argumentation with single extensions: Abstraction principles and the grounded extension. In *Proc. of European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, pages 107–118, 2009.

[38] Guido Boella, Souhila Kaci, and Leendert W. N. van der Torre. Dynamics in argumentation with single extensions: Attack refinement and the grounded extension. In *Proc. of Sixth International Workshop on Argumentation in Multi-Agent Systems (ArgMAS)*, pages 150–159, 2009.

[39] Richard Booth, Martin Caminada, and Braden Marshall. DISCO: A web-based implementation of discussion games for grounded and preferred semantics. In *Proc. of COMMA*, pages 453–454, 2018.

[40] Gerhard Brewka, Sylwia Polberg, and Stefan Woltran. Generalizations of dung frameworks and their role in formal argumentation. *IEEE Intelligent Systems*, 29(1):30–38, 2014.

[41] Martin Caminada. Semi-stable semantics. In *Proc. of 1st International Conference on Computational Models of Argument (COMMA)*, pages 121–130, 2006.

[42] Martin Caminada, Samy Sá, João Alcântara, and Wolfgang Dvorák. On the equivalence between logic programming semantics and argumentation semantics. *Int. J. Approx. Reasoning*, 58:87–111, 2015.

[43] Martin W. A. Caminada, Wolfgang Dvorák, and Srdjan Vesic. Preferred semantics as socratic discussion. *Journal of Logic and Computation*, 26(4):1257–1292, 2016.

[44] M. Capobianco, C. I. Chesñevar, and G. R. Simari. Argumentation and the dynamics of warranted beliefs in changing environments. *Autonomous Agents and Multi-Agent Systems*, 11(2):127–151, 2005.

[45] M. Capobianco and G. R. Simari. A proposal for making argumentation computationally capable of handling large repositories of uncertain data. In *Proc. of SUM*, pages 95–110, 2009.

[46] Dan Cartwright and Katie Atkinson. Political engagement through tools for argumentation. In *Computational Models of Argument: Proceedings of COMMA 2008, Toulouse, France, May 28-30, 2008.*, pages 116–127, 2008.

[47] Dan Cartwright and Katie Atkinson. Using computational argumentation to support e-participation. *IEEE Intelligent Systems*, 24(5):42–52, 2009.

[48] Claudette Cayrol, Florence Dupin de Saint-Cyr, and Marie-Christine Lagasquie-Schiex. Revision of an argumentation system. In *Proc. of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 124–134, 2008.

[49] Claudette Cayrol, Florence Dupin de Saint-Cyr, and Marie-Christine Lagasquie-Schiex. Change in abstract argumentation frameworks: Adding an argument. *Journal of Artificial Intelligence Research*, 38:49–84, 2010.

[50] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. On the acceptability of arguments in bipolar argumentation frameworks. In *Proc. of European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, pages 378–389, 2005.

[51] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. Bipolar abstract argumentation systems. In *Argumentation in Artificial Intelligence*, pages 65–84. 2009.

[52] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. Coalitions of arguments: A tool for handling bipolar argumentation frameworks. *International Journal of Intelligent System*, 25(1):83–109, 2010.

[53] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. Change in abstract bipolar argumentation systems. In *Proc. of International Conference on Scalable Uncertainty Management (SUM)*, pages 314–329, 2015.

[54] L. A. Cecchi, P. R. Fillottrani, and G. R. Simari. On the complexity of DeLP through game semantics. In *Proc. of NMR*, pages 386–394, 2006.

[55] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. ArgSemSAT: Solving argumentation problems using SAT. In *COMMA*, pages 455–456, 2014.

[56] Federico Cerutti, Massimiliano Giacomin, Mauro Vallati, and Marina Zanella. An SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation. In *Proc. of KR*, 2014.

[57] Günther Charwat, Wolfgang Dvorák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artificial Intelligence*, 220:28–63, 2015.

[58] Andrea Cohen, Sebastian Gottifredi, Alejandro Javier Garca, and Guillermo Ricardo Simari. A survey of different approaches to support in argumentation systems. *The Knowledge Engineering Review*, 29(5):513–550, 2014.

[59] Robert Craven and Francesca Toni. Argument graphs and assumption-based argumentation. *Artif. Intell.*, 233:1–59, 2016.

[60] C. A. D. Deagustini, S. E. Fulladoza Dalibón, S. Gottifredi, M. A. Falappa, C. I. Chesñevar, and G. R. Simari. Relational databases as a massive information source for defeasible argumentation. *Knowledge-Based Systems*, 51:93–109, 2013.

[61] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.

[62] Phan Minh Dung, Paolo Mancarella, and Francesca Toni. Computing ideal sceptical argumentation. *Artificial Intelligence*, 171(10-15):642–674, 2007.

[63] Paul E. Dunne. The computational complexity of ideal semantics. *Artificial Intelligence*, 173(18):1559–1591, 2009.

[64] Paul E Dunne and Martin Caminada. Computational complexity of semi-stable semantics in abstract argumentation frameworks. In *European Workshop on Logics in Artificial Intelligence*, pages 153–165, 2008.

[65] Paul E. Dunne and Michael Wooldridge. Complexity of abstract argumentation. In *Argumentation in Artificial Intelligence*, pages 85–104. 2009.

[66] Wolfgang Dvorak, Matti Jarvisalo, Johannes Peter Wallner, and Stefan Woltran. Complexity-sensitive decision procedures for abstract argumentation. *Artificial Intelligence*, 206:53–78, 2014.

[67] Wolfgang Dvorak, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for argumentation. In *Proc. of European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)*, 2010.

[68] Wolfgang Dvorak and Stefan Woltran. Complexity of semi-stable and stage semantics in argumentation frameworks. *Information Processing Letters*, 110(11):425–430, 2010.

[69] Edith Elkind, Manuela Veloso, Noa Agmon, and Matthew E. Taylor, editors. *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[70] M. A. Falappa, G. Kern-Isberner, and G. R. Simari. Explanations, belief revision and defeasible reasoning. *Artif. Intell.*, 141(1/2):1–28, 2002.

[71] Marcelo A. Falappa, Alejandro Javier Garcia, Gabriele Kern-Isberner, and Guillermo Ricardo Simari. On the evolving relation between belief revision and argumentation. *The Knowledge Engineering Review*, 26(1):35–43, 2011.

[72] Bettina Fazzinga, Sergio Flesca, and Francesco Parisi. Efficiently estimating the probability of extensions in abstract argumentation. In *Proc. of International Conference on Scalable Uncertainty Management (SUM)*, pages 106–119, 2013.

[73] Bettina Fazzinga, Sergio Flesca, and Francesco Parisi. On the complexity of probabilistic abstract argumentation frameworks. *ACM Transactions on Computational Logic*, 16(3):22, 2015.

[74] Bettina Fazzinga, Sergio Flesca, and Francesco Parisi. On efficiently estimating the probability of extensions in abstract argumentation frameworks. *International Journal of Approximate Reasoning*, 69:106–132, 2016.

[75] Bettina Fazzinga, Sergio Flesca, Francesco Parisi, and Adriana Pietramala. PARTY: A mobile system for efficiently assessing the probability of extensions in a debate. In *Proc. of International Conference on Database and Expert Systems Applications (DEXA)*, pages 220–235, 2015.

[76] John Fox, David Glasspool, Vivek Patkar, Mark Austin, Liz Black, Matthew South, Dave Robertson, and Charles Vincent. Delivering clinical decision support services: there is nothing as practical as a good theory. *Journal of biomedical informatics*, 43(5):831–843, 2010.

[77] Sarah Alice Gaggl and Norbert Manthey. ASPARTIX-D ready for the competition, 2015.

[78] A. J. Garca and G. R. Simari. Defeasible logic programming: DeLP-servers, contextual queries, and explanations for answers. *Argument & Computation*, 5(1):63–88, 2014.

[79] Alejandro J. Garca and Guillermo R. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming (TPLP)*, 4(1-2):95–138, 2004.

[80] S. Gottifredi, N. D. Rotstein, A. J. Garca, and G. R. Simari. Using argument strength for building dialectical bonsai. *Annals of Mathematics and Artificial Intelligence*, 69(1):103–129, 2013.

[81] Sergio Greco and Francesco Parisi. Efficient computation of deterministic extensions for dynamic abstract argumentation frameworks. In *Proc. of European Conference on Artificial Intelligence (ECAI)*, pages 1668–1669, 2016.

[82] Sergio Greco and Francesco Parisi. Incremental computation of deterministic extensions for dynamic argumentation frameworks. In *Proc. of European Conference On Logics In Artificial Intelligence (JELIA)*, pages 288–304, 2016.

[83] Sergio Greco and Domenico Saccà. Complexity and expressive power of deterministic semantics for datalog¬. *Inf. Comput.*, 153(1):81–98, 1999.

[84] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[85] Antonis C. Kakas and Loizos Michael. Cognitive systems: Argument and cognition. *IEEE Intelligent Informatics Bulletin*, 17(1):14–20, 2016.

[86] Nadin Kökciyan, Nefise Yaglikci, and Pinar Yolum. Argumentation for resolving privacy disputes in online social networks: (extended abstract). In *Proc. of International Conference on Autonomous Agents and Multiagent Sytems (AAMAS)*, pages 1361–1362, 2016.

[87] Markus Kröll, Reinhard Pichler, and Stefan Woltran. On the complexity of enumerating the extensions of abstract argumentation frameworks. In *Proc. of IJCAI*, pages 1145–1152, 2017.

[88] Jean-Marie Lagniez, Emmanuel Lonca, and Jean-Guy Mailly. CoQuiAAS: A constraint-based quick abstract argumentation solver. In *ICTAI*, pages 928–935, 2015.

[89] Bei Shui Liao, Li Jin, and Robert C. Koons. Dynamics of argumentation systems: A division-based method. *Artificial Intelligence*, 175(11):1790–1814, 2011.

[90] Beishui Liao. Toward incremental computation of argumentation semantics: A decomposition-based approach. *Annals of Mathematics and Artificial Intelligence*, 67(3-4):319–358, 2013.

[91] Beishui Liao and Huaxin Huang. Partial semantics of argumentation: basic properties and empirical results. *Journal of Logic and Computation*, 23(3):541–562, 2013.

[92] Beishui Liao, Liyun Lei, and Jianhua Dai. Computing preferred labellings by exploiting sccs and most sceptically rejected arguments. In *TAFA Workshop*, pages 194–208, 2013.

[93] Hugo Mercier and Dan Sperber. Why do humans reason? arguments for an argumentative theory. *Behavioral and brain sciences*, 34(2):57–74, 2011.

[94] S. Modgil and H. Prakken. The *ASPIC*$^+$ framework for structured argumentation: A tutorial. *Argument & Computation*, 5(1):31–62, 2014.

[95] Sanjay Modgil. An abstract theory of argumentation that accommodates defeasible reasoning about preferences. In *Proc. of European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, pages 648–659, 2007.

[96] Sanjay Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173(9-10):901–934, 2009.

[97] Sanjay Modgil, Francesca Toni, Floris Bex, Ivan Bratko, Carlos I. Chesnevar, Wolfgang Dvorak, Marcelo A. Falappa, Xiuyi Fan, Sarah Alice Gaggl, Ale-

jandro J. Garca, Maria P. Gonzalez, Thomas F. Gordon, Joao Leite, Martin Mouzina, Chris Reed, Guillermo R. Simari, Stefan Szeider, Paolo Torroni, and Stefan Woltran. *Agreement Technologies*, volume 8 of *Law, Governance and Technology*, chapter 21: The Added Value of Argumentation: Examples and Challenges, pages 357–404. Springer, 2013.

[98] M. O. Moguillansky, N. D. Rotstein, M. A. Falappa, A. J. Garca, and G. R. Simari. Dynamics of knowledge in *DeLP* through argument theory change. *TPLP*, 13(6):893–957, 2013.

[99] Emilia Oikarinen and Stefan Woltran. Characterizing strong equivalence for argumentation frameworks. *Artificial Intelligence*, 175(14-15):1985–2009, 2011.

[100] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[101] Henry Prakken. An abstract framework for argumentation with structured arguments. *Argument & Computation*, 1(2):93–124, 2010.

[102] Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence*. Springer Publishing Company, Incorporated, New York, 1st edition, 2009.

[103] Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[104] Domenico Saccà and Carlo Zaniolo. Deterministic and non-deterministic stable models. *J. Log. Comput.*, 7(5):555–579, 1997.

[105] Raymond M Smullyan and Raymond Smullyan. *What is the name of this book?: the riddle of Dracula and other logical puzzles*. Prentice-Hall Englewood Cliffs, NJ, 1978.

[106] F. Stolzenburg, A. J. Garca, C. I. Chesnevar, and G. R. Simari. Computing generalized specificity. *Journal of Applied Non-Classical Logics*, 13(1):87–113, 2003.

[107] Hannes Strass and Stefan Ellmauthaler. godiamond-iccma 2017 system description.

[108] Matthias Thimm and Serena Villata. The first international competition on computational models of argumentation: Results and analysis. *Artificial Intelligence*, 252:267–294, 2017.

[109] Matthias Thimm, Serena Villata, Federico Cerutti, Nir Oren, Hannes Strass, and Mauro Vallati. Summary report of the first international competition on computational models of argumentation. *Artificial Intelligence Magazine*, 37(1):102, 2016.

[110] F. Toni. A tutorial on assumption-based argumentation. *Argument & Computation*, 5(1):89–117, 2014.

[111] Eugenio Di Tullio and Floriana Grasso. A model for a motivational system grounded on value based abstract argumentation frameworks. In *Electronic Healthcare - 4th International Conference, eHealth 2011, Málaga, Spain, November 21-23, 2011, Revised Selected Papers*, pages 43–50, 2011.

[112] I. D. Viglizzo, F. A. Tohmé, and G. R. Simari. *Ann. Math. Artif. Intell.*, 57(2):181–204, 2009.

[113] Serena Villata, Guido Boella, Dov M. Gabbay, and Leendert W. N. van der Torre. Modelling defeasible and prioritized support in bipolar argumentation. *Annals of Mathematics and Artificial Intelligence*, 66(1-4):163–197, 2012.

[114] Yuming Xu and Claudette Cayrol. The matrix approach for abstract argumentation frameworks. In *Proc. of International Workshop on Theory and Applications of Formal Argumentation (TAFA)*, pages 243–259, 2015.