How we designed winning algorithms for abstract argumentation and which insight we attained

Federico Cerutti[a], Massimiliano Giacomin[b], Mauro Vallati[c]

[a]*School of Computer Science & Informatics, Queen's Buildings, Cardiff University, CF24 3AA, Cardiff, United Kingdom*
[b]*Department of Information Engineering, University of Brescia, via Branze, 38, 25123, Brescia, Italy*
[c]*School of Computing and Engineering, University of Huddersfield, Huddersfield, HD1 3DH, United Kingdom*

Abstract

In this paper we illustrate the design choices that led to the development of ArgSemSAT, the winner of the preferred semantics track at the 2017 International Competition on Computational Models of Arguments (ICCMA 2017), a biennial contest on problems associated to the Dung's model of abstract argumentation frameworks, widely recognised as a fundamental reference in computational argumentation. The algorithms of ArgSemSAT are based on multiple calls to a SAT solver to compute complete labellings, and on encoding constraints to drive the search towards the solution of decision and enumeration problems. In this paper we focus on preferred semantics (and incidentally stable as well), one of the most popular and complex semantics for identifying acceptable arguments. We discuss our design methodology that includes a systematic exploration and empirical evaluation of labelling encodings, algorithmic variations and SAT solver choices. In designing the successful ArgSemSAT, we discover that: (1) there is a labelling encoding that appears to be universally better than other, logically equivalent ones; (2) composition of different techniques such as AllSAT and enumerating stable extensions when searching for preferred semantics brings advantages; (3) injecting domain specific knowledge in the algorithm design can lead to significant improvements.

Keywords: Abstract Argumentation, SAT-based Algorithms, Stable and Preferred Semantics

1. Introduction

Computational Argumentation is emerging in Artificial Intelligence as a reasoning paradigm able to handle incomplete and inconsistent information in a

Email addresses: `CeruttiF@cardiff.ac.uk` (Federico Cerutti), `massimiliano.giacomin@unibs.it` (Massimiliano Giacomin), `m.vallati@hud.ac.uk` (Mauro Vallati)

way that fosters the development of robust intelligent systems in e.g. the legal, medical, e-government and debate systems domains [1]. In particular, argument-based debate systems aim not only at supporting human-human argumentation, but also at developing machine-generated arguments.

In this setting, Dung's model of abstract argumentation frameworks [2] is widely recognised as a fundamental reference in virtue of its simplicity and ability to capture a variety of more specific approaches as special cases, in the areas of non-monotonic reasoning, logic programming and structured argumentation [2, 3, 4, 5, 6]. More specifically, Dung's model plays a role in the *assessment layer* of argumentation, i.e. to establish the justification of arguments and their conclusions. In argument-based debate technology applications, it supports the assessment of debatewide features such as which arguments are winning [1].

An abstract argumentation framework (AF) consists of a set of arguments and an *attack* relation between them. The concept of *extension* plays a key role in this simple setting, where an extension is intuitively a set of arguments which can "survive the conflict together." Different notions of extension correspond to alternative *argumentation semantics*, whose definitions and properties are actively investigated since more than two decades (see [7, 8] for an introduction).

The three *traditional* argumentation semantics introduced in Dung's paper [2], namely *grounded*, *stable*, and *preferred* semantics, currently represent the most widely adopted approaches to determine the justification status of arguments in abstract argumentation. They are all based on *complete* semantics, also introduced in [2] as a unifying framework for different semantics, since extensions are selected among the complete ones. While grounded semantics always identifies a single extension which can be computed in polynomial time, stable and preferred semantics allow multiple extensions, yielding in some cases the capability of assigning a committed status to arguments that are left undecided by grounded semantics [9]. Differently from stable semantics, preferred semantics also guarantees the existence of extensions for any argumentation framework, thus providing a justification status of arguments in cases where applying stable semantics collapses [2]. In this respect, the name of preferred semantics reflects a sort of preference w.r.t. the other traditional argumentation semantics.

The main computational problems in abstract argumentation are naturally related to extensions and include decision problems, e.g. determining whether an argument belongs to all extensions, and construction problems, such as enumerating all of the extensions prescribed by a given semantics for a given AF. Unfortunately, the complexity of extension-related decision problems turns out to be intractable for most of the semantics [10, 11] and this clearly has a substantial detrimental effect whenever dealing with real-world problems, e.g. [12]. As to stable semantics, credulous and skeptical reasoning are in the first level of the polynomial hierarchy. As to preferred semantics, the advantages of preferred semantics mentioned above come at a price in terms of computational complexity, in particular skeptical reasoning is located at the second level. These complexity issues motivate a recent interest for the investigation on efficient algorithms for abstract argumentation and their empirical assessment. In order to foster the development of such algorithms as well as determining the state-of-the-art of current implementations, the *International Competition on Computational Models of Argumentation* (ICCMA)[1] has been established on a biennial basis starting in 2015.

While different approaches turn out to be complementary and can be successfully combined in portfolios [13], reduction-based approaches, most of them based on SAT, had the best performance during ICCMA 2017,[2] the last competition. It is worth noticing that this is not a universal truth as we already discussed at length in [13].

In this paper we illustrate the design choices that led to the development of SAT-based algorithms for preferred semantics, the most challenging semantics—from the complexity perspective—among the traditional ones [2]. These design choices are at the basis of the solver ArgSemSAT, the winner of the track at ICCMA 2017 devoted to preferred semantics. We also consider stable semantics since it can play a significant role also in the context of preferred semantics.

As it will be clear in the following, high performance of ArgSemSAT has been achieved by applying SAT-reductions in a relatively simple way. In particular, no decomposition techniques on the argumentation framework have been applied, since previous experiences on the application of the SCC-recursive schema [14, 15] seem to indicate that such techniques are efficient only when the number of strongly connected components is large. Moreover, we have exploited classical SAT solvers with hard clauses, leaving apart the use of soft clauses which may play a role in the identification of preferred extensions [16]. It should be pointed out that we are not claiming that more advanced SAT-based techniques have a detrimental effect on performance, rather that one of the main reasons why ArgSemSAT achieves efficiency is a careful identification of the features of the reduction to SAT.

In particular, it turns out that modelling labellings (i.e. a counterpart of extensions based on labels) with different boolean formulas, as well as expressing the constraints involving labellings with different logically equivalent encodings, has a dramatic impact on performance. In addition, there might be different options in the exploration of the set of labellings; for instance in the context of preferred semantics one may either begin from scratch or start from stable labellings by exploiting the relevant algorithms. Finally, algorithms can be implemented using different SAT solvers.

Since all of these aspects are interrelated, a systematic analysis is needed to explore all different options, followed by an empirical evaluation to identify the most efficient combinations. This is the reason why in this paper we do not compare ArgSemSAT against the other solvers of the competition, rather we focus on the above features that—as experiments show—have a dramatic impact on performance. This way, we describe a methodology that can be applied also to different SAT-based approaches for abstract argumentation, possibly exploiting advanced features based e.g. on graph decomposition and soft clauses.

[1]`http://argumentationcompetition.org` (on 15 February 2019).
[2]`http://argumentationcompetition.org/2017/results.html` (on 15 February 2019).

Our methodology comprises two steps: first correct algorithms and their variations need to be identified. In this paper we introduce at an abstract level the algorithms of ArgSemSAT in Section 3. We then explore possible encodings of the labellings and of the relevant constraints in Section 4, and detail algorithm implementations in Section 5.

Then, the various algorithms and variants need to be empirically evaluated, and Section 6 is devoted to the experimental analysis of ArgSemSAT configurations and to assess a number of hypotheses concerning the interplay of the different features mentioned above.

One of the main take-home messages of our work is that it is of pivotal importance to promote the flexibility and configurability of solvers. The well-known *no free-lunch theorem* [17] indicates that any two solvers will deliver indistinguishable performance when compared on a sufficiently large and diverse set of benchmarks. However, since usually solvers are exploited on a much smaller set of instances, the ability to specialise a solver for a class of instances of interest can lead to paramount performance improvement. What our work fosters, and our experimental analysis suggests, is that any general (i.e., domain independent) solver dealing with complex problems should incorporate different methods and techniques, and then allow the selection, configuration, and use of the most suitable ones given the instances at hand. This is well-aligned with the research done in the area of algorithm configuration [18, 19]. Furthermore, it is worth emphasising that the specialisation of solvers according to some testing benchmarks can also lead to a better understanding of solvers behaviours, in terms of weaknesses and strengths (see, for instance [20]). Analysing and comparing different configurations can shed some light on the elements that are affecting performance, potentially leading to further improvements.

Such a data-driven methodology can then be applied to analyse advantages and disadvantages of approaches based on advanced features. For instance, in [15, 21] we discussed how SCC-decomposition might play a role when dealing with certain classes of AFs. Besides the presence at the local level of similar variants as in a global algorithm, such an advanced feature could potentially be exploited by an algorithm, hence the need to analyse and compare different configurations against a large set of benchmarks.

The validity of such a methodology is ensured by the fact that ArgSemSAT is the winner of the preferred semantics track at ICCMA 2017: further analyses with currently prototypical approaches to exploit advanced features are left as future work.

Comparison with relevant work is provided in Section 7, while in Section 8 we have some conclusive remarks. Unless otherwise stated, proofs are listed in Appendix A. Appendix B summarises the parameters used in the algorithms we introduce in Section 5. Finally, Appendix C illustrates a pilot study investigating the difference in performance when using one propositional variable instead of three in the search for stable extensions using a SAT solver.

2. Background

2.1. Argumentation Frameworks and Semantics

An argumentation framework [2] consists of a set of arguments[3] and a binary attack relation between them.

Definition 1. An *argumentation framework* (*AF*) is a pair $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a set of arguments and $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$. We say that $\mathbf{b}$ *attacks* $\mathbf{a}$ iff $(\mathbf{b}, \mathbf{a}) \in \mathcal{R}$, also denoted as $\mathbf{b} \to \mathbf{a}$. The set of attackers of an argument $\mathbf{a}$ is denoted as $\mathbf{a}^- \triangleq \{\mathbf{b} : \mathbf{b} \to \mathbf{a}\}$, the set of arguments attacked by $\mathbf{a}$ is denoted as $\mathbf{a}^+ \triangleq \{\mathbf{b} : \mathbf{a} \to \mathbf{b}\}$. An argument $\mathbf{a}$ without attackers, i.e. such that $\mathbf{a}^- = \varnothing$, is said *initial.* We also extend attack notations to sets of arguments, i.e. given $E, S \subseteq \mathcal{A}$, $E \to \mathbf{a}$ iff $\exists \mathbf{b} \in E$ s.t. $\mathbf{b} \to \mathbf{a}$; $\mathbf{a} \to E$ iff $\exists \mathbf{b} \in E$ s.t. $\mathbf{a} \to \mathbf{b}$; $E \to S$ iff $\exists \mathbf{b} \in E, \mathbf{a} \in S$ s.t. $\mathbf{b} \to \mathbf{a}$; $E^- \triangleq \{\mathbf{b} \mid \exists \mathbf{a} \in E, \mathbf{b} \to \mathbf{a}\}$ and $E^+ \triangleq \{\mathbf{b} \mid \exists \mathbf{a} \in E, \mathbf{a} \to \mathbf{b}\}$.

Each argumentation framework has an associated directed graph where the vertices are the arguments, and the edges are the attacks.

The basic properties of conflict–freeness, acceptability, and admissibility of a set of arguments are fundamental for the definition of argumentation semantics.

Definition 2. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$:

- a set $S \subseteq \mathcal{A}$ is a *conflict–free* set of Γ if $\nexists\, \mathbf{a}, \mathbf{b} \in S$ s.t. $\mathbf{a} \to \mathbf{b}$;
- an argument $\mathbf{a} \in \mathcal{A}$ is *acceptable* in Γ with respect to a set $S \subseteq \mathcal{A}$ if $\forall \mathbf{b} \in \mathcal{A}$ s.t. $\mathbf{b} \to \mathbf{a}$, $\exists\ \mathbf{c} \in S$ s.t. $\mathbf{c} \to \mathbf{b}$;
- the function $\mathcal{F}_\Gamma : 2^{\mathcal{A}} \to 2^{\mathcal{A}}$ such that $\mathcal{F}_\Gamma(S) = \{\mathbf{a} \mid \mathbf{a} \text{ is acceptable in } \Gamma \text{ w.r.t. } S\}$ is called the *characteristic function* of Γ;
- a set $S \subseteq \mathcal{A}$ is an *admissible* set of Γ if S is a conflict–free set of Γ and every element of S is acceptable in Γ with respect to S, i.e. $S \subseteq \mathcal{F}_\Gamma(S)$.

An argumentation semantics σ prescribes for any *AF* Γ a set of *extensions*, denoted as $\mathcal{E}_\sigma(\Gamma)$, namely a set of sets of arguments satisfying the conditions dictated by σ. The paper is focused on stable (denoted as ST), and preferred (denoted as PR) semantics, introduced as follows.

Definition 3. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$:

- a set $S \subseteq \mathcal{A}$ is a *stable extension* of Γ, i.e. $S \in \mathcal{E}_{\mathsf{ST}}(\Gamma)$, iff S is a conflict-free set of Γ and $S \cup S^+ = \mathcal{A}$;
- a set $S \subseteq \mathcal{A}$ is a *preferred extension* of Γ, i.e. $S \in \mathcal{E}_{\mathsf{PR}}(\Gamma)$, iff S is a maximal (w.r.t. set inclusion) admissible set of Γ.

[3]In this paper we consider only *finite* sets of arguments: see [22] for a discussion on infinite sets of arguments.

The notion of complete extension introduced in [2] represents a common ground for both semantics, i.e. both preferred and stable extensions can be defined as complete extensions satisfying specific maximality properties. This is recalled in Proposition 1.

Definition 4. Given an AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, a set $S \subseteq \mathcal{A}$ is a *complete extension* of Γ iff S is a conflict-free set of Γ and $S = \mathcal{F}_\Gamma(S)$. The set of complete extensions of Γ is denoted as $\mathcal{E}_{\mathsf{CO}}(\Gamma)$.

Proposition 1 (Originally introduced in [2])**.** Given an AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, a set $S \subseteq \mathcal{A}$ is a preferred extension of Γ iff S is a maximal (w.r.t. set inclusion) complete extension, it is a stable extension of Γ iff it is a complete extension such that for all $\mathbf{a} \in \mathcal{A}$, $\mathbf{a} \in (S \cup S^+)$.

From the above proposition it is easy to see that $\mathcal{E}_{\mathsf{ST}}(\Gamma) \subseteq \mathcal{E}_{\mathsf{PR}}(\Gamma) \subseteq \mathcal{E}_{\mathsf{CO}}(\Gamma)$.

Taking into account that the extensions prescribed by any semantics are conflict-free, one can note that each extension S implicitly defines a three-valued *labelling* of arguments: an argument $\mathbf{a}$ is labelled `in` iff $\mathbf{a} \in S$; it is labelled `out` iff $S \to \mathbf{a}$; it is labelled `undec` if neither of the above conditions holds. In the light of this correspondence, argumentation semantics can be equivalently defined in terms of labellings rather than of extensions [23, 24, 25, 8].

Definition 5. Given a set of arguments S, a *labelling* of S is a total function $\mathcal{L}ab : S \mapsto \{\mathtt{in}, \mathtt{out}, \mathtt{undec}\}$. The set of all *labellings* of S is denoted as $\mathfrak{L}_S$. Given an AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, a *labelling* of Γ is a labelling of $\mathcal{A}$. The set of all *labellings* of Γ is denoted as $\mathfrak{L}(\Gamma)$.

Given a labelling $\mathcal{L}ab$, we write $\mathtt{in}(\mathcal{L}ab)$ for $\{A | \mathcal{L}ab(A) = \mathtt{in}\}$, $\mathtt{out}(\mathcal{L}ab)$ for $\{A | \mathcal{L}ab(A) = \mathtt{out}\}$ and $\mathtt{undec}(\mathcal{L}ab)$ for $\{A | \mathcal{L}ab(A) = \mathtt{undec}\}$.

Labellings can be related by the commitment relation introduced in [26]:

Definition 6. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab_1, \mathcal{L}ab_2 \in \mathfrak{L}(\Gamma)$. $\mathcal{L}ab_2$ is more or equally committed than $\mathcal{L}ab_1 (\mathcal{L}ab_1 \sqsubseteq \mathcal{L}ab_2)$ iff $\mathtt{in}(\mathcal{L}ab_1) \subseteq \mathtt{in}(\mathcal{L}ab_2)$ and $\mathtt{out}(\mathcal{L}ab_1) \subseteq \mathtt{out}(\mathcal{L}ab_2)$. Furthermore, $\mathcal{L}ab_1 \sqsubsetneq \mathcal{L}ab_2$ iff $\mathcal{L}ab_1 \sqsubseteq \mathcal{L}ab_2$ and $\mathcal{L}ab_2 \not\sqsubseteq \mathcal{L}ab_1$.

As shown in [26], $\sqsubseteq$ is a partial order between labellings:

Proposition 2 (Originally introduced in [26])**.** The relation $\sqsubseteq$ between labellings is reflexive, transitive and anti-symmetric.

Complete labellings can be defined[4] as follows.

Definition 7. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. A labelling $\mathcal{L}ab \in \mathfrak{L}(\Gamma)$ is a *complete labelling* of Γ iff it satisfies the following conditions for any $\mathbf{a} \in \mathcal{A}$:

[4]The reader may notice that the definition is redundant, e.g. the third condition may be dropped since it is entailed by the first two ones. The choice of this definition is for the sake of exposition.

- $\mathcal{L}ab(\mathbf{a}) = \texttt{in} \Leftrightarrow \forall \mathbf{b} \in \mathbf{a}^{-} \mathcal{L}ab(\mathbf{b}) = \texttt{out}$;
- $\mathcal{L}ab(\mathbf{a}) = \texttt{out} \Leftrightarrow \exists \mathbf{b} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{b}) = \texttt{in}$;
- $\mathcal{L}ab(\mathbf{a}) = \texttt{undec} \Leftrightarrow \forall \mathbf{b} \in \mathbf{a}^{-} \mathcal{L}ab(\mathbf{b}) \neq \texttt{in} \wedge \exists \mathbf{c} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{c}) = \texttt{undec}$.

The stable and preferred labellings can then be defined on the basis of complete labellings.

Definition 8. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. A labelling $\mathcal{L}ab \in \mathfrak{L}(\Gamma)$ is

- a *stable labelling* of Γ iff it is a complete labelling of Γ and there is no argument labelled $\texttt{undec}$;
- a *preferred labelling* of Γ iff it is a complete labelling of Γ maximising the set of arguments labelled $\texttt{in}$.

The sets including the complete, stable and preferred labellings of an argumentation framework Γ are denoted as $\mathfrak{L}_{\mathsf{CO}}(\Gamma)$, $\mathfrak{L}_{\mathsf{ST}}(\Gamma)$ and $\mathfrak{L}_{\mathsf{PR}}(\Gamma)$, respectively. From the above definition, it is easy to see that $\mathfrak{L}_{\mathsf{ST}}(\Gamma) \subseteq \mathfrak{L}_{\mathsf{PR}}(\Gamma) \subseteq \mathfrak{L}_{\mathsf{CO}}(\Gamma)$.

On the basis of the results in [2] and [26], the following relations between complete and preferred labellings can be derived:

Proposition 3 (Originally introduced in [26])**.** Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ and a labelling $\mathcal{L}ab$, $\mathcal{L}ab$ is a preferred labelling of Γ if and only if $\mathcal{L}ab$ is a maximal complete labelling (w.r.t. $\sqsubseteq$) of Γ. Moreover, given a complete labelling $\mathcal{L}ab$, there exists a preferred labelling $\mathcal{L}ab'$ such that $\mathcal{L}ab \sqsubseteq \mathcal{L}ab'$.

In order to show the connection between extensions and labellings, let us recall from [8] the definition of the function `Ext2Lab`, returning the labelling corresponding to a conflict–free set of arguments S.

Definition 9. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ and a conflict–free set $S \subseteq \mathcal{A}$, the corresponding labelling $\texttt{Ext2Lab}(S)$ is the labelling $\mathcal{L}ab$ such that

- $\mathcal{L}ab(\mathbf{a}) = \texttt{in} \Leftrightarrow \mathbf{a} \in S$
- $\mathcal{L}ab(\mathbf{a}) = \texttt{out} \Leftrightarrow \exists\ \mathbf{b} \in S$ s.t. $\mathbf{b} \to \mathbf{a}$
- $\mathcal{L}ab(\mathbf{a}) = \texttt{undec} \Leftrightarrow \mathbf{a} \notin S \wedge \nexists\ \mathbf{b} \in S$ s.t. $\mathbf{b} \to \mathbf{a}$

As shown in [25], there is a bijective correspondence between the complete, stable, preferred extensions and the complete, stable, preferred labellings, respectively.

Proposition 4 (Originally introduced in [25])**.** Given an an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $\mathcal{L}ab$ is a complete (stable, preferred) labelling of Γ if and only if there is a complete (stable, preferred) extension S of Γ such that $\mathcal{L}ab = \texttt{Ext2Lab}(S)$.

	Semantics σ	
	ST	PR
DC-σ	NP-compl.	NP-compl.
DS-σ	coNP-compl.	Π_2^p-compl.

Table 1: Computational complexity of credulous and skeptical acceptance in finite AFs: DC denotes credulous acceptance, while DS skeptical acceptance.

2.2. Computational Problems in Abstract Argumentation

The main computational problems in abstract argumentation are related to the different semantics and can be partitioned into two classes: *decision* problems and *construction* problems. Decision problems pose yes/no questions like "Does this argument belong to one (all) extensions?" or "Is this set an extension?", while construction problems require to explicitly produce some of the extensions prescribed by a semantics. The complexity of extension-related decision problems has been deeply investigated and, for most of the semantics proposed in the literature they have been proven to be intractable. Intractability extends directly to construction/enumeration problems, given that their solutions provide direct answers to decision problems.

In this paper we focus on *extension enumeration* (a construction problem), *credulous acceptance* and *skeptical acceptance* of an argument (both of them decision problems).

Extension enumeration is the problem of constructing all of the extensions prescribed by a given semantics for a given AF. Following [27], we denote such a problem as EE-σ.

Credulous and skeptical acceptance refer to a specific argument, where the natural questions arising are related to extension inclusion. In particular, an argument $\mathbf{a}$ is *credulously* accepted with regard to a given semantics σ and a given AF Γ iff $\mathbf{a}$ belongs to at least one extension of Γ under σ, i.e. $\exists E \in \mathcal{E}_\sigma(\Gamma)$ s.t. $\mathbf{a} \in E$. We denote the problem of determining whether an argument of AF is credulously accepted with regard to a semantics σ as DC-σ. An argument $\mathbf{a}$ is *skeptically* accepted with regard to a given semantics σ and a given AF Γ iff $\mathbf{a}$ belongs to each extension of Γ under σ, i.e. $\forall E \in \mathcal{E}_\sigma(\Gamma)$ $\mathbf{a} \in E$. We denote the corresponding decision problem as DS-σ.[5]

The complexity of credulous and skeptical acceptance problems for stable and preferred semantics are shown in Table 1 [10]. We refer the reader to [11] for a recent investigation on the complexity of the extension enumeration problem under different semantics.

Given the bijective correspondence between extensions and labellings proved by Proposition 4, the above problems can be directly formulated as the problem

[5]Please note that if $\mathcal{E}_\sigma(\Gamma) = \emptyset$, it is trivially true that $\forall \mathbf{a} \in \mathcal{A}$ DS-$\sigma(\mathbf{a}) = \top$ (vacuous truth).

of computing all labellings prescribed by a given semantics (corresponding to extensions enumeration), determining whether there is a labelling where an argument is labelled `in` (credulous acceptance) and whether an argument is labelled `in` by all labellings (skeptical acceptance).

2.3. Propositional satisfiability and SAT solvers

A propositional formula over a set of boolean variables is satisfiable if and only if there exists a truth assignment of the variables such that the formula evaluates to True. Checking whether such an assignment exists is the satisfiability (SAT) problem, and an algorithm able to solve this problem is called SAT solver.

Most SAT solvers require as input a propositional formula in conjunctive normal form (CNF), i.e. expressed as a conjunction of clauses, where each clause is a disjunction of literals (in the context of propositional logic, a literal is either a propositional variable or its negation). Moreover, if the formula is satisfiable SAT solvers are generally able to return as output a satisfying assignment.

A variation of the SAT problem is the so-called AllSAT problem, namely given a CNF formula as input, determining the set of all its satisfying assignments. A typical AllSAT solver (ALLSatS) iteratively computes satisfying assignments using a traditional SAT solver and adds to the input formula some *blocking clauses* which are the complement of total or partial assignments to exclude previously identified solutions. More specifically, a practical ALLSatS receives a CNF formula as input and solves the AllSAT problem by determining an equivalent formula in disjunctive normal form (DNF), i.e. as a disjunction of clauses where each clause is a conjunction of literals. From [28]—that builds on top of [29] and [30]—the generic structure of an AllSAT solver is presented in Algorithm 1. Line 3 uses a SAT solver (SatS) to compute a satisfying assignment for the formula F . Line 4 calls the unspecified function compute to obtain the partial assignment (cube) q_i from m_i (total assignment). A blocking clause c_i is found and added to F_i preventing these assignments from being enumerated any further. This loop ends when F_i becomes unsatisfiable.

Algorithm 1 Algorithm template for AllSAT solver, [28, Algorithm 1]

```
    Input: $F$ (a CNF Formula)
    Output: $Q$ (a DNF Formula) such that $F \iff Q$
1:  $i := 1,\ F_1 := F,\ Q := \bot$
2:  while not unsat($F_i$) do
3:      $m_i :=$ SatS($F_i$)                      ▷ get a satisfying assignment of $F_i$
4:      $(q_i, c_i) :=$ compute($m_i$)             ▷ get cube and blocking clause
5:      $Q := Q \vee q_i, F_{i+1} := F_i \wedge c_i$       ▷ update $Q$ and $F_i$
6:      $i := i + 1$
7:  end while
8:  return $Q$
```

As noticed in [28], a very simple implementation of compute is to set $q_i := m_i$, and $c_i := \neg q_i$. In [28] the authors discuss at length more efficient ways to implement such a compute function, and achieve speed improvements up to three orders of magnitude.

There are also extensions to the SAT and AllSAT problems where clauses are partitioned into hard and soft clauses, and the solutions are determined among all assignments satisfying all hard clauses and maximising the sum of numerical weights associated to soft clauses. However, as specified in Section 1, in this paper we focus on *classical* SAT and AllSAT solvers: an application of weighted MaxSAT to compute the preferred extensions is described in [16].

3. Abstract Algorithms for Stable and Preferred Semantics

In this section we provide in abstract form the algorithms we consider for the solution to the computational problems under stable and preferred semantics, respectively. These algorithms perform a search in the space of (a subset of) the complete labellings, where each step of the search process requires the invocation of a non-deterministic function which returns a complete labelling satisfying a number of specific constraints. Acceptable solutions of the non-deterministic functions are encoded as a propositional formula, in such a way that each satisfying assignment corresponds to a desired complete labelling. This way, the function can be implemented by means of a SAT solver. Different ways of identifying the corresponding formulas will be introduced in Section 4.

3.1. Stable Semantics

The first problem we consider is the enumeration of stable labellings. Algorithm 2 starts from an empty set of labellings $\mathcal{L}_s$ (initalised at line 1) and simply consists of a loop (lines 2–4) which at each iteration identifies a novel stable labelling (line 2) and records it in $\mathcal{L}_s$ (line 3). In particular, $\underline{\mathsf{FindST}}^{\neq}$ is a non-deterministic function that, given an argumentation framework Γ and a set of stable labellings $\mathcal{L}_s \subseteq \mathfrak{L}_{\mathsf{ST}}(\Gamma)$, returns a stable labelling $\mathcal{L}ab \in \mathfrak{L}_{\mathsf{ST}}(\Gamma)$ such that $\mathcal{L}ab \notin \mathcal{L}_s$ if such a labelling exists, and returns $\perp$ in the other case. As shown in Section 4, this function (as well as $\underline{\mathsf{FindST}_{\mathbf{a}}}$ in Algorithms 3 and 4) can identify a stable labelling as a satisfying assignment of a propositional formula which encodes the constraints of complete labellings enriched with the condition of having no undec arguments. The issue of encodings complete labellings is dealt with in Section 4.1.

It can be seen that the loop of Algorithm 2 enumerates all the satisfying assignments of a propositional formula, corresponding to the stable labellings of Γ. As a consequence, the loop can be replaced by a single call to an AllSAT solver: this will be available as an option of the implemented algorithm (see Section 5).

Credulous acceptance under stable semantics can simply be checked by searching for a stable labelling which assigns the in-label to the input argument. Accordingly, Algorithm 3 consists of a single line exploiting the non-deterministic function $\underline{\mathsf{FindST}_{\mathbf{a}}}$ which receives as input an argumentation framework $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, an argument $\mathbf{a} \in \mathcal{A}$ and a label $s \in \{\mathtt{in}, \mathtt{out}\}$, and returns as output a

Algorithm 2 Enumeration of Stable Labellings

```
    SAT-EL-ST
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$
    Output: $\mathcal{L}_s = \mathfrak{L}_{\mathsf{ST}}(\Gamma)$
1: $\mathcal{L}_s := \varnothing$
2: while $\mathcal{L}ab := \underline{\mathsf{FindST}}^{\neq}(\Gamma, \mathcal{L}_s)$ do
3:    $\mathcal{L}_s := \mathcal{L}_s \cup \{\mathcal{L}ab\}$
4: end while
5: return $\mathcal{L}_s$
```

stable labelling $\mathcal{L}ab \in \mathfrak{L}_{\mathsf{ST}}(\Gamma)$ such that $\mathcal{L}ab(\mathbf{a}) = s$ if such a labelling exists, $\bot$ otherwise.

Algorithm 3 Credulous acceptance under Stable Semantics

```
    SAT-DC-ST
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $\mathbf{a} \in \mathcal{A}$
    Output: $\top$ if $\mathbf{a}$ is credulously accepted under $\mathsf{ST}$, $\bot$ otherwise
1: return $\underline{\mathsf{FindST}}_{\mathbf{a}}(\Gamma, \mathtt{in}) \neq \bot$
```

As to skeptical acceptance, Algorithm 4 checks whether there is a stable labelling such that the argument is `out` (line 1), returning $\bot$ if this is the case (line 2). If such a labelling does not exist, then either the argumentation framework has no stable labellings or every stable labelling labels the argument `in`: in both cases the argument is skeptically accepted, and the algorithm returns $\top$ accordingly (line 4).

Algorithm 4 Skeptical acceptance under Stable Semantics

```
    SAT-DS-ST
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $\mathbf{a} \in \mathcal{A}$
    Output: $\top$ if $\mathbf{a}$ is skeptically accepted under $\mathsf{ST}$, $\bot$ otherwise
1: if $\underline{\mathsf{FindST}}_{\mathbf{a}}(\Gamma, \mathtt{out}) \neq \bot$ then
2:    return $\bot$
3: end if
4: return $\top$
```

Correctness of Algorithms 2, 3 and 4 is pointed out in the following theorem, whose proof is obvious and is thus omitted.

Theorem 1. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ and $\mathbf{a} \in \mathcal{A}$:

- $\{\mathtt{in}(\mathcal{L}ab) \mid \mathcal{L}ab \in \textbf{SAT-EL-ST}(\Gamma))\} = \mathcal{E}_{\mathsf{ST}}(\Gamma)$
- $\textbf{SAT-DC-ST}(\Gamma, \mathbf{a}) \equiv (\exists S \in \mathcal{E}_{\mathsf{ST}}(\Gamma), \mathbf{a} \in S)$
- $\textbf{SAT-DS-ST}(\Gamma, \mathbf{a}) \equiv (\forall S \in \mathcal{E}_{\mathsf{ST}}(\Gamma), \mathbf{a} \in S)$

3.2. Preferred Semantics

Due to the complexity gap between SAT and the preferred labelling enumeration problem, it is not possible to identify in polynomial time a propositional formula (of polynomial size) whose models correspond to preferred labellings. In particular, differently to the case of stable semantics, in order to identify preferred labellings it is necessary to maximise `in`-arguments among complete labellings, thus the constraints corresponding to preferred labellings cannot be obtained by adding a simple condition to the constraints of complete labellings.

Algorithm 5 enumerates the preferred labellings of a framework Γ by means of two nested loops. The outer loop (lines 2-7) starts from an empty set of labellings $\mathcal{L}_p$ (line 1) and identifies at each iteration a novel preferred labelling $\mathcal{L}ab$, which is added to $\mathcal{L}_p$ (line 6).

In particular, $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}$ is a non-deterministic function that, given an argumentation framework Γ and a set of preferred labellings $\mathcal{L}_p \subseteq \mathfrak{L}_{\mathsf{PR}}(\Gamma)$, picks out a labelling $\mathcal{L}ab$ from a set of complete labellings $\mathcal{L}$ subject to the following constraints:

- $\mathcal{L} \subseteq \mathfrak{L}_{\mathsf{CO}}(\Gamma)$, i.e. $\mathcal{L}$ only includes complete labellings of Γ
- $\forall \mathcal{L}ab' \in \mathcal{L}, \forall \mathcal{L}ab'' \in \mathcal{L}_p, \mathcal{L}ab' \not\sqsubseteq \mathcal{L}ab''$, i.e. $\mathcal{L}$ does not include those labellings that are less or equally committed w.r.t. a labelling of $\mathcal{L}_p$
- $(\mathfrak{L}_{\mathsf{PR}}(\Gamma) \setminus \mathcal{L}_p) \subseteq \mathcal{L}$, i.e. $\mathcal{L}$ includes the preferred labellings that are not already included in $\mathcal{L}_p$

If there is no labelling in $\mathcal{L}$, the function returns $\bot$.

Note that the above constraints on $\mathcal{L}$ are compatible, in particular the preferred labellings that are not included in $\mathcal{L}_p$ cannot be less committed than any labelling of $\mathcal{L}_p$, due to Proposition 3.

Ideally, the set $\mathcal{L}$ should only include the preferred labellings of Γ that have not yet been found, i.e. $\mathcal{L} = \mathfrak{L}_{\mathsf{PR}}(\Gamma) \setminus \mathcal{L}_p$, entailing that a preferred labelling is directly identified at line 2. Since this is not possible in general, $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_p)$, line 2, identifies a complete labelling which is not necessary preferred, but due to the second constraint on $\mathcal{L}$ there is a preferred labelling which is more or equally committed w.r.t. it. This preferred labelling is identified by the inner loop (lines 3-5) which starts from the complete labelling $\mathcal{L}ab$ and produces a sequence of more committed complete labellings strictly ordered w.r.t. $\sqsubsetneq$. In particular, $\underline{\mathsf{FindCL}}^{\sqsupsetneq}$ is a non-deterministic function that, given an argumentation framework Γ and a complete labelling $\mathcal{L}ab \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$, returns a complete labelling $\mathcal{L}ab'$ such that $\mathcal{L}ab \sqsubsetneq \mathcal{L}ab'$ if such a labelling exists, $\bot$ otherwise. When the sequence can no more be extended, $\underline{\mathsf{FindCL}}^{\sqsupsetneq}(\Gamma, \mathcal{L}ab)$ returns $\bot$ at line 3, and its last element $\mathcal{L}ab$ correspond to a maximal complete labelling w.r.t. $\sqsubseteq$, namely to a preferred labelling.

Since stable labellings are also preferred, instead of starting from an empty set of preferred labellings at line 1 it is possible to first compute the stable labellings by Algorithm 2, i.e. line 1 is replaced with $\mathcal{L}_p :=$ **SAT-EL-ST**(Γ). This straightforward variation is reported in Algorithm 6.

Algorithm 5 Enumeration of Preferred Labellings

```
    EL-PR
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$
    Output: $\mathcal{L}_p = \mathfrak{L}_{\mathsf{PR}}(\Gamma)$
1:  $\mathcal{L}_p := \varnothing$
2:  while $\mathcal{L}ab := \underline{\mathsf{FindCL}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_p)$ do
3:      while $\mathcal{L}ab' := \underline{\mathsf{FindCL}}^{\sqsupsetneq}(\Gamma, \mathcal{L}ab)$ do
4:          $\mathcal{L}ab := \mathcal{L}ab'$
5:      end while
6:      $\mathcal{L}_p := \mathcal{L}_p \cup \{\mathcal{L}ab\}$
7:  end while
8:  return $\mathcal{L}_p$
```

Algorithm 6 Enumeration of Preferred Labellings: variation using the enumeration of Stable Labellings

```
    EL-PR-withST
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$
    Output: $\mathcal{L}_p = \mathfrak{L}_{\mathsf{PR}}(\Gamma)$
1:  $\mathcal{L}_p :=$ SAT-EL-ST$(\Gamma)$
2:  while $\mathcal{L}ab := \underline{\mathsf{FindCL}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_p)$ do
3:      while $\mathcal{L}ab' := \underline{\mathsf{FindCL}}^{\sqsupsetneq}(\Gamma, \mathcal{L}ab)$ do
4:          $\mathcal{L}ab := \mathcal{L}ab'$
5:      end while
6:      $\mathcal{L}_p := \mathcal{L}_p \cup \{\mathcal{L}ab\}$
7:  end while
8:  return $\mathcal{L}_p$
```

Since preferred labellings do not directly correspond to the satisfying assignments of a propositional formula, differently to the case of stable semantics AllSAT solvers are not directly applicable here. An option would be to determine all complete labellings by solving an AllSAT problem and then to identify the maximal ones w.r.t. $\sqsubseteq$, however this would be inefficient compared to Algorithms 5 and 6, which are able to cut the search space of complete labellings.

Turning to credulous acceptance, similarly to the case of stable semantics it suffices to determine whether a complete labelling exists which assigns the `in` label to the input argument (thus ensuring that there is a preferred labelling, possibly coinciding with the identified complete one, satisfying this condition). This corresponds to Algorithm 7, where $\underline{\mathsf{FindCL}_{\mathbf{a}}}$ is a function that, given an argumentation framework Γ, an argument $\mathbf{a} \in \mathcal{A}$ and a label $s \in \{\mathtt{in}, \mathtt{out}\}$, returns a complete labelling $\mathcal{L}ab \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$ such that $\mathcal{L}ab(\mathbf{a}) = s$ if such a labelling exists, $\bot$ otherwise.

Algorithm 7 Credulous acceptance under Preferred Semantics

```
    DC-PR
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $\mathbf{a} \in \mathcal{A}$
    Output: $\top$ if $\mathbf{a}$ is credulously accepted under PR, $\bot$ otherwise
1:  return $\underline{\mathsf{FindCL}_{\mathbf{a}}}(\Gamma, \mathtt{in}) \neq \bot$
```

Skeptical acceptance can be determined by enumerating the preferred labellings of the input argumentation framework, and checking whether all of them assign the label `in` to the input argument. This corresponds to Algorithm 8, a straightforward adaptation of Algorithm 5 which iterates over the preferred labellings of Γ and returns $\bot$ whenever it identifies a preferred labelling where the input argument $\mathbf{a}$ is not labelled `in` (lines 6 and 7). If for all preferred labellings this is not the case, then $\mathbf{a}$ is skeptically accepted and the algorithm returns $\top$ at line 11.

Similarly to the case of Algorithm 5, it is also possible to start from the stable labellings to check whether one of them does not assign the label `in` to $\mathbf{a}$. However Algorithm 9 shows a rather different alternative to Algorithm 8. Since stable extensions by definition do not assign the label `undec`, it is more efficient to check with a single call to the function $\underline{\mathsf{FindCL}_{\mathbf{a}}}$ whether there is a complete labelling which assigns the label `out` to $\mathbf{a}$, entailing the existence of a preferred labelling satisfying this condition and thus disproving skeptical acceptance of $\mathbf{a}$. Moreover, it can also be checked whether there is a complete labelling assigning the label `in` to $\mathbf{a}$, and if this is not the case then $\mathbf{a}$ is not skeptically accepted either (line 1–3).

If the algorithm enters line 4 then there are no preferred labellings where $\mathbf{a}$ is `out`, and the algorithm looks for a preferred labelling assigning the label `undec` to $\mathbf{a}$ in order to prove that $\mathbf{a}$ is not skeptically accepted. To this purpose, a loop (lines 5–13) starts from an empty set of complete labellings $\mathcal{L}_c$, and considers at each iteration a maximal complete labelling such that $\mathbf{a}$ is labelled `undec`. For ease of explanation, let us define $\mathcal{L}_{mu}$ as the set including the

Algorithm 8 Skeptical acceptance under Preferred Semantics 1

```
    DS-PR-straightforward
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $\mathbf{a} \in \mathcal{A}$
    Output: $\top$ if $\mathbf{a}$ is skeptically accepted under $\mathsf{PR}$, $\bot$ otherwise
 1: $\mathcal{L}_p := \varnothing$
 2: while $\mathcal{L}ab := \underline{\mathsf{FindCL}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_p)$ do
 3:   while $\mathcal{L}ab' := \underline{\mathsf{FindCL}}^{\sqsupsetneq}(\Gamma, \mathcal{L}ab)$ do
 4:     $\mathcal{L}ab := \mathcal{L}ab'$
 5:   end while
 6:   if $\mathcal{L}ab(\mathbf{a}) \neq \mathtt{in}$ then
 7:     return $\bot$
 8:   end if
 9:   $\mathcal{L}_p := \mathcal{L}_p \cup \{\mathcal{L}ab\}$
10: end while
11: return $\top$
```

labellings of this kind, i.e. $\mathcal{L}_{mu} \triangleq \{\mathcal{L}ab \in \mathfrak{L}_{\mathsf{CO}}(\Gamma) \mid \mathcal{L}ab(\mathbf{a}) = \mathtt{undec} \wedge \nexists \mathcal{L}ab' \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$ s.t. $(\mathcal{L}ab'(\mathbf{a}) = \mathtt{undec} \wedge \mathcal{L}ab \sqsubsetneq \mathcal{L}ab')\}$. $\underline{\mathsf{FindCLU}}_{\mathbf{a}}^{\not\sqsubseteq}$ is a non-deterministic function that, given an argument $\mathbf{a} \in \mathcal{A}$, an argumentation framework Γ and a set of complete labellings $\mathcal{L}_c \subseteq \mathcal{L}_{mu}$, picks out a complete labelling $\mathcal{L}ab$ (not necessarily maximal) from a set of complete labellings $\mathcal{L}$ subject to the following constraints:

- $\mathcal{L} \subseteq \{\mathcal{L}ab' \in \mathfrak{L}_{\mathsf{CO}}(\Gamma) \mid \mathcal{L}ab'(\mathbf{a}) = \mathtt{undec}\}$, i.e. $\mathcal{L}$ only includes complete labellings of Γ which assign the label $\mathtt{undec}$ to $\mathbf{a}$;
- $\forall \mathcal{L}ab' \in \mathcal{L}, \forall \mathcal{L}ab'' \in \mathcal{L}_c, \mathcal{L}ab' \not\sqsubseteq \mathcal{L}ab''$, i.e. $\mathcal{L}$ does not include those labellings that are less or equally committed w.r.t. a labelling of $\mathcal{L}_c$;
- $(\{\mathcal{L}ab' \in \mathfrak{L}_{\mathsf{PR}}(\Gamma) \mid \mathcal{L}ab'(\mathbf{a}) = \mathtt{undec}\} \backslash \mathcal{L}_c) \subseteq \mathcal{L}$, i.e. $\mathcal{L}$ includes the preferred labellings where $\mathbf{a}$ is $\mathtt{undec}$ that are not already included in $\mathcal{L}_c$.

If there is no such labelling in $\mathcal{L}$, the function returns $\bot$.

Note that the definition of $\underline{\mathsf{FindCLU}}_{\mathbf{a}}^{\not\sqsubseteq}$ is well founded, i.e. the above constraints are compatible due to the maximality w.r.t. $\sqsubseteq$ of preferred labellings (see Proposition 3).

At each iteration of the loop (lines 5–13), a complete labelling $\mathcal{L}ab$ such that $\mathcal{L}ab(\mathbf{a}) = \mathtt{undec}$ is selected (line 5). Then the inner loop (lines 6-8) starts from the labelling $\mathcal{L}ab$ and replaces it with a (not necessarily strictly) more committed labelling in $\mathcal{L}_{mu}$. In particular, similarly to $\underline{\mathsf{FindCL}}^{\sqsupsetneq}$, $\underline{\mathsf{FindCL}}_{\mathbf{a}}^{\sqsupsetneq}$ is a non-deterministic function that, given an argument $\mathbf{a}$, an argumentation framework Γ, a complete labelling $\mathcal{L}ab \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$ such that $\mathcal{L}ab(\mathbf{a}) = \mathtt{undec}$ and a label $s \in \{\mathtt{in}, \mathtt{undec}\}$, returns a complete labelling $\mathcal{L}ab' \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$ such that $\mathcal{L}ab'(\mathbf{a}) = s$ and $\mathcal{L}ab \sqsubsetneq \mathcal{L}ab'$ if such a labelling exists, $\bot$ otherwise. At each iteration of the inner loop, $\underline{\mathsf{FindCL}}_{\mathbf{a}}^{\sqsupsetneq}(\Gamma, \mathcal{L}ab, \mathtt{undec})$ produces a strictly more committed labelling such that $\mathbf{a}$ is $\mathtt{undec}$, until the resulting sequence can not more be extended and the loop is exited. This way, at line 9 the variable $\mathcal{L}ab$ includes a labelling of $\mathcal{L}_{mu}$.

It should be noted that such a labelling $\mathcal{L}ab$ obtained is not necessarily a preferred labelling, since there might exist a strictly more committed labelling such that $\mathbf{a}$ is `in` (recall that from line 4 we are guaranteed that there are no preferred labellings where $\mathbf{a}$ is labelled `out`). The existence of such a labelling is checked at line 9. If $\underline{\mathsf{FindCL}}^{\sqsupset}_{\mathbf{a}}(\Gamma, \mathcal{L}ab, \mathtt{in}) = \bot$, then there is no strictly more committed labelling $\mathcal{L}ab''$ such that $\mathcal{L}ab'' = \mathtt{in}$, i.e. $\mathcal{L}ab$ is a preferred labelling such that $\mathcal{L}ab(\mathbf{a}) = \mathtt{undec}$, thus $\mathbf{a}$ is not skeptically accepted and the algorithm returns $\bot$. Otherwise, $\mathcal{L}ab$ is a labelling of $\mathcal{L}_{mu}$ which is not preferred (there is a strictly more committed complete labelling) and such that all more committed preferred labellings assign to $\mathbf{a}$ the label `in` (by the maximality of $\mathcal{L}ab$ there is no more committed preferred labelling assigning to $\mathbf{a}$ the label `undec`, and as mentioned above there is no preferred labelling where $\mathbf{a}$ is `out`). As a consequence, $\mathcal{L}ab$ is recorded by adding it to $\mathcal{L}_c$ at line 12, so that at the next iteration of the loop all complete labellings less committed w.r.t. it (including $\mathcal{L}ab$ itself) are excluded from the search, since according to the considerations above this does not exclude the possible preferred labellings where $\mathbf{a}$ is `undec`.

If at line 5 the function $\underline{\mathsf{FindCLU}}^{\not\sqsubseteq}_{\mathbf{a}}(\Gamma, \mathcal{L}_c)$ does not return any labelling, then there are no preferred labellings where $\mathbf{a}$ is `undec`, and the algorithm returns $\top$ (line 14).

Algorithm 9 Skeptical acceptance under Preferred Semantics 2

```
    DS-PR
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $\mathbf{a} \in \mathcal{A}$
    Output: $\top$ if $\mathbf{a}$ is skeptically accepted under PR, $\bot$ otherwise
 1: if $\underline{\mathsf{FindCL}}_{\mathbf{a}}(\Gamma, \mathtt{in}) = \bot \lor \underline{\mathsf{FindCL}}_{\mathbf{a}}(\Gamma, \mathtt{out}) \neq \bot$ then
 2:   return $\bot$
 3: end if
 4: $\mathcal{L}_c := \varnothing$
 5: while $\mathcal{L}ab := \underline{\mathsf{FindCLU}}^{\not\sqsubseteq}_{\mathbf{a}}(\Gamma, \mathcal{L}_c)$ do
 6:   while $\mathcal{L}ab' := \underline{\mathsf{FindCL}}^{\sqsupset}_{\mathbf{a}}(\Gamma, \mathcal{L}ab, \mathtt{undec})$ do
 7:     $\mathcal{L}ab := \mathcal{L}ab'$
 8:   end while
 9:   if $\underline{\mathsf{FindCL}}^{\sqsupset}_{\mathbf{a}}(\Gamma, \mathcal{L}ab, \mathtt{in}) = \bot$ then
10:     return $\bot$
11:   end if
12:   $\mathcal{L}_c := \mathcal{L}_c \cup \{\mathcal{L}ab\}$
13: end while
14: return $\top$
```

The correctness of Algorithms 5-9 is proved by the following theorem.

Theorem 2. Given an AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ and $\mathbf{a} \in \mathcal{A}$:

- $\{\mathtt{in}(\mathcal{L}ab) \mid \mathcal{L}ab \in \mathbf{EL\text{-}PR}(\Gamma))\} = \mathcal{E}_{\mathsf{PR}}(\Gamma)$ (Algorithm 5);

- $\{\mathtt{in}(\mathcal{L}ab) \mid \mathcal{L}ab \in \textbf{EL-PR-withST}(\Gamma))\} = \mathcal{E}_{\mathsf{PR}}(\Gamma)$ (Algorithm 6);
- $\textbf{DC-PR}(\Gamma, \mathbf{a}) \equiv (\exists S \in \mathcal{E}_{\mathsf{PR}}(\Gamma), \mathbf{a} \in S)$ (Algorithm 7);
- $\textbf{DS-PR-straightforward}(\Gamma, \mathbf{a}) \equiv (\forall S \in \mathcal{E}_{\mathsf{PR}}(\Gamma), \mathbf{a} \in S)$ (Algorithm 8);
- $\textbf{DS-PR}(\Gamma, \mathbf{a}) \equiv (\forall S \in \mathcal{E}_{\mathsf{PR}}(\Gamma), \mathbf{a} \in S)$ (Algorithm 9).

4. Encoding Complete Labellings and their Constraints

Each of the non-deterministic functions exploited by the algorithms described in Section 3 returns a single complete labelling subject to specific constraints (e.g. being related by $\subsetneq$ w.r.t. a given complete labelling). Taking into account that a stable labelling is a complete labelling, this also holds for the functions returning stable labellings that appear in the algorithms of Section 3.1.

Since a complete labelling can be expressed by means of local conditions relating the label of each node with those of its attackers (see Definition 7), using a SAT solver is a natural way to implement these functions. According to these considerations, two problems have to be faced in this respect:

- How to encode the constraints corresponding to complete labellings as a propositional formula?
- How to encode as propositional clauses, to be conjoined with the formula above, the specific constraints on the complete labelling returned by each function?

While these might seem clear-cut tasks, several syntactically different encodings can be devised which, although logically equivalent, can significantly affect the performance of the overall process of searching a satisfying assignment. For instance, adding some "redundant" clauses to a formula may speed up the search process, thanks to the additional information provided by the additional constraints. On the other hand, increasing syntactic complexity might lead to worse performances, thus a careful selection of the encoding is needed. These aspects will be the main focus of Section 6.

The aim of the following two subsections instead is to explore possible encodings for complete labellings and the relevant constraints.

4.1. Logical Encodings of Complete Labellings

Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ we are interested in identifying a boolean formula such that its satisfying assignments have a one-to-one correspondence with the complete labellings of Γ.

As mentioned above, several ways to encode complete labellings can be envisaged. In order to explore alternative encodings, let us consider again the requirements of Definition 7. They can be expressed as a conjunction of 6 terms, i.e. $C^{\rightarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}} \wedge C^{\leftarrow}_{\mathtt{undec}}$, where

- $C^{\rightarrow}_{\mathtt{in}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{in} \Rightarrow \forall \mathbf{b} \in \mathbf{a}^{-} \mathcal{L}ab(\mathbf{b}) = \mathtt{out})$;

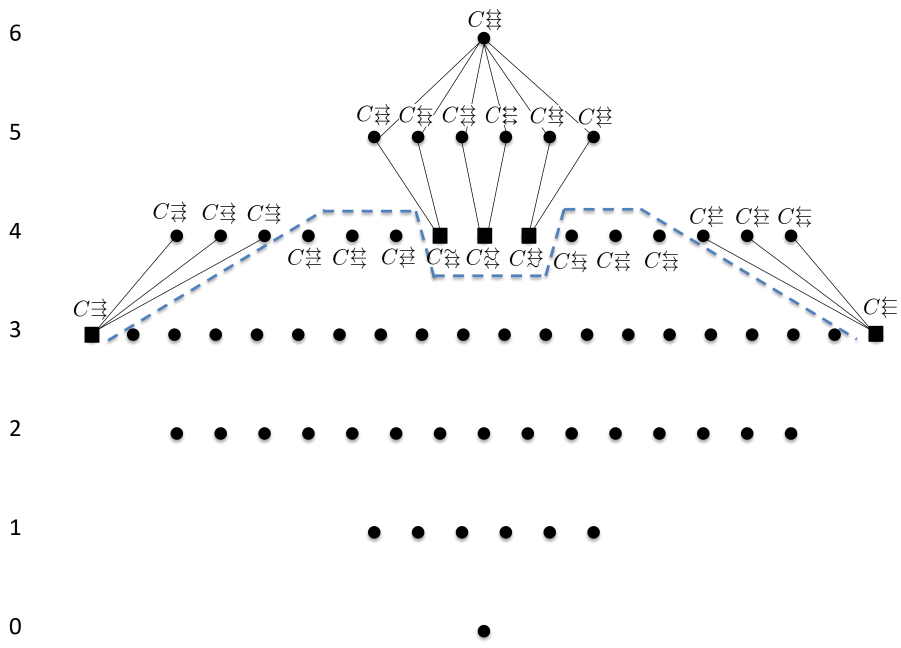

Figure 1: Partial Hasse Diagram of 64 possible encodings of complete labellings.

- $C^{\leftarrow}_{\mathtt{in}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{in} \Leftarrow \forall \mathbf{b} \in \mathbf{a}^{-} \mathcal{L}ab(\mathbf{b}) = \mathtt{out})$;
- $C^{\rightarrow}_{\mathtt{out}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{out} \Rightarrow \exists \mathbf{b} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{b}) = \mathtt{in})$;
- $C^{\leftarrow}_{\mathtt{out}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{out} \Leftarrow \exists \mathbf{b} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{b}) = \mathtt{in})$;
- $C^{\rightarrow}_{\mathtt{undec}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{undec} \Rightarrow \forall \mathbf{b} \in \mathbf{a}^{-} \mathcal{L}ab(\mathbf{b}) \neq \mathtt{in} \wedge \exists \mathbf{c} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{c}) = \mathtt{undec})$;
- $C^{\leftarrow}_{\mathtt{undec}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{undec} \Leftarrow \forall \mathbf{b} \in \mathbf{a}^{-} \mathcal{L}ab(\mathbf{b}) \neq \mathtt{in} \wedge \exists \mathbf{c} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{c}) = \mathtt{undec})$.

Let us also define $C^{\leftrightarrow}_{\mathtt{in}} \equiv C^{\rightarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{in}}$, $C^{\leftrightarrow}_{\mathtt{out}} \equiv C^{\rightarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{out}}$, $C^{\leftrightarrow}_{\mathtt{undec}} \equiv C^{\rightarrow}_{\mathtt{undec}} \wedge C^{\leftarrow}_{\mathtt{undec}}$.

We aim at exploring all the encodings corresponding to the 64 possible subsets of the 6 terms above, characterised by a cardinality (i.e. the number of terms) between 0 and 6 and partially ordered according to the $\subseteq$-relation. We will denote these encodings as $C^{i}_{\overset{o}{u}}$, with $i, o, u \in \{\rightarrow, \leftarrow, \leftrightarrow, \sim\}$, i.e. $C^{i}_{\overset{o}{u}} \equiv C^{i}_{\mathtt{in}} \wedge C^{o}_{\mathtt{out}} \wedge C^{u}_{\mathtt{undec}}$, where $C^{\sim}_{\mathtt{in}} = C^{\sim}_{\mathtt{out}} = C^{\sim}_{\mathtt{undec}} = \top$ (in other words, the symbol

$\sim$ denotes that the corresponding term is not present in the encoding). For instance, $C^{\substack{\rightarrow\\ \leftrightarrow\\ \sim}} \equiv C_{\texttt{in}}^{\rightarrow} \wedge C_{\texttt{out}}^{\leftrightarrow}$.

Using basic combinatorial calculus it can be seen that the 64 subsets can be partitioned as follows: one encoding with cardinality 0 (i.e. the empty encoding $C^{\substack{\sim\\ \sim\\ \sim}}$), 6 encodings with cardinality 1, 15 encodings with cardinality 2, 20 encodings with cardinality 3, 15 encodings with cardinality 4, 6 encodings with cardinality 5 and one encoding with cardinality 6 (i.e. $C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \leftrightarrow}}$ corresponding to Definition 7). Figure 1 depicts these encodings by arranging them according to their cardinality (indicated at the left), and partially shows the inclusion relationships (focusing on those that will be relevant in the following).

The encodings can be partitioned into three classes:

1. *weak* encodings, i.e. such that there is an argumentation framework and a labelling satisfying all their terms which however is not a complete labelling;

2. *correct and non-redundant* encodings, i.e. able to correctly identify complete labellings and such that any strict subset of their terms is weak;

3. *correct and redundant* encodings, i.e. able to correctly identify complete labellings and such that there is a strict subset of their terms which is correct.

The corresponding characterisation of the 64 encodings is illustrated in Figure 1: the dotted line separates the weak encodings (under the line) from the correct ones (over the line), and among the correct ones those that are non-redundant are depicted as squares, redundant encodings appear as circles[6]. This characterisation is shown to be correct by Theorem 3, which requires two preliminary propositions.

Proposition 5. Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. A total function $\mathcal{L}ab : \mathcal{A} \mapsto \{\texttt{in}, \texttt{out}, \texttt{undec}\}$ is a complete labelling iff it satisfies any of the following conjunctive constraints for any $\mathbf{a} \in \mathcal{A}$:

$$C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \sim}} = C_{\texttt{in}}^{\leftrightarrow} \wedge C_{\texttt{out}}^{\leftrightarrow}$$
$$C^{\substack{\leftrightarrow\\ \sim\\ \leftrightarrow}} = C_{\texttt{in}}^{\leftrightarrow} \wedge C_{\texttt{undec}}^{\leftrightarrow}$$
$$C^{\substack{\sim\\ \leftrightarrow\\ \leftrightarrow}} = C_{\texttt{out}}^{\leftrightarrow} \wedge C_{\texttt{undec}}^{\leftrightarrow}$$
$$C^{\substack{\rightarrow\\ \rightarrow\\ \rightarrow}} = C_{\texttt{in}}^{\rightarrow} \wedge C_{\texttt{out}}^{\rightarrow} \wedge C_{\texttt{undec}}^{\rightarrow}$$
$$C^{\substack{\leftarrow\\ \leftarrow\\ \leftarrow}} = C_{\texttt{in}}^{\leftarrow} \wedge C_{\texttt{out}}^{\leftarrow} \wedge C_{\texttt{undec}}^{\leftarrow}$$

[6] $C_{\texttt{in}}^{\leftrightarrow} \wedge C_{\texttt{out}}^{\leftrightarrow}$ and $C_{\texttt{in}}^{\rightarrow} \wedge C_{\texttt{out}}^{\rightarrow} \wedge C_{\texttt{undec}}^{\rightarrow}$ correspond to the alternative definitions of complete labellings in [31], where a proof of their equivalence is provided.

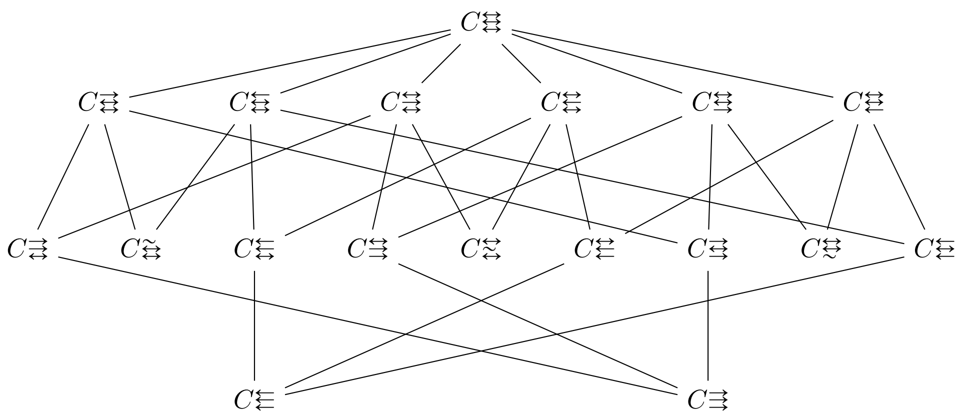

Figure 2: Hasse-like diagram for the 18 correct encodings of complete labellings.

Proposition 6. The following 6 encodings are weak:

$C^{\substack{\rightarrow\\\leftarrow\\\leftrightarrow}} = C^{\leftrightarrow}_{\texttt{undec}} \wedge C^{\rightarrow}_{\texttt{in}} \wedge C^{\leftarrow}_{\texttt{out}}$
$C^{\substack{\leftarrow\\\rightarrow\\\leftrightarrow}} = C^{\leftrightarrow}_{\texttt{undec}} \wedge C^{\leftarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}}$
$C^{\substack{\rightarrow\\\leftrightarrow\\\leftarrow}} = C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{in}} \wedge C^{\leftarrow}_{\texttt{undec}}$
$C^{\substack{\leftarrow\\\leftrightarrow\\\rightarrow}} = C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\leftarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{undec}}$
$C^{\substack{\leftrightarrow\\\rightarrow\\\leftarrow}} = C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\leftarrow}_{\texttt{undec}}$
$C^{\substack{\leftrightarrow\\\leftarrow\\\rightarrow}} = C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\leftarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$

In particular, the proof of Proposition 6 identifies for each encoding an argumentation framework admitting a unique complete labelling which drastically differs from the one satisfying the weak encoding, i.e. there are arguments labelled `in` that should be labelled `undec` or there are arguments labelled `out` or `undec` that should be labelled `in` (see the proof in Appendix A for details).

Theorem 3. All the encodings of cardinality 0, 1, and 2 are weak. Among the encodings of cardinality 3, $C^{\substack{\rightarrow\\\rightarrow\\\rightarrow}} = C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$ and $C^{\substack{\leftarrow\\\leftarrow\\\leftarrow}} = C^{\leftarrow}_{\texttt{in}} \wedge C^{\leftarrow}_{\texttt{out}} \wedge C^{\leftarrow}_{\texttt{undec}}$ are correct and non-redundant, the other 18 encodings are weak. Among the encodings of cardinality 4, $C^{\substack{\leftrightarrow\\\leftrightarrow\\\sim}} = C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\leftrightarrow}_{\texttt{out}}$, $C^{\substack{\leftrightarrow\\\sim\\\leftrightarrow}} = C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\leftrightarrow}_{\texttt{undec}}$ and $C^{\substack{\sim\\\leftrightarrow\\\leftrightarrow}} = C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\leftrightarrow}_{\texttt{undec}}$ are correct and non-redundant, the 6 encodings $C^{\substack{\leftrightarrow\\\rightarrow\\\rightarrow}} = C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$, $C^{\substack{\rightarrow\\\leftrightarrow\\\rightarrow}} = C^{\rightarrow}_{\texttt{in}} \wedge C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$, $C^{\substack{\rightarrow\\\rightarrow\\\leftrightarrow}} = C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\leftrightarrow}_{\texttt{undec}}$, $C^{\substack{\leftrightarrow\\\leftarrow\\\leftarrow}} = C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\leftarrow}_{\texttt{out}} \wedge C^{\leftarrow}_{\texttt{undec}}$, $C^{\substack{\leftarrow\\\leftrightarrow\\\leftarrow}} = C^{\leftarrow}_{\texttt{in}} \wedge C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\leftarrow}_{\texttt{undec}}$, $C^{\substack{\leftarrow\\\leftarrow\\\leftrightarrow}} = C^{\leftarrow}_{\texttt{in}} \wedge C^{\leftarrow}_{\texttt{out}} \wedge C^{\leftrightarrow}_{\texttt{undec}}$ are correct and redundant, the other 6 encodings are weak. All the encodings of cardinality 5 and 6 are correct and redundant.

Let $\mathfrak{C} = \{C^{\substack{\leftarrow\\\leftarrow\\\leftarrow}}, C^{\substack{\rightarrow\\\rightarrow\\\rightarrow}}, C^{\substack{\rightarrow\\\rightarrow\\\leftrightarrow}}, C^{\substack{\sim\\\leftrightarrow\\\leftrightarrow}}, C^{\substack{\leftarrow\\\leftarrow\\\leftrightarrow}}, C^{\substack{\leftrightarrow\\\rightarrow\\\rightarrow}}, C^{\substack{\leftrightarrow\\\sim\\\leftrightarrow}}, C^{\substack{\leftrightarrow\\\leftarrow\\\leftarrow}}, C^{\substack{\rightarrow\\\leftrightarrow\\\rightarrow}}, C^{\substack{\leftrightarrow\\\leftrightarrow\\\sim}}, C^{\substack{\leftarrow\\\leftrightarrow\\\leftarrow}}, C^{\substack{\rightarrow\\\leftrightarrow\\\leftrightarrow}}, C^{\substack{\leftarrow\\\leftrightarrow\\\leftrightarrow}}, C^{\substack{\leftrightarrow\\\rightarrow\\\leftrightarrow}}, C^{\substack{\leftrightarrow\\\leftarrow\\\leftrightarrow}}, C^{\substack{\leftrightarrow\\\leftrightarrow\\\rightarrow}}, C^{\substack{\leftrightarrow\\\leftrightarrow\\\leftarrow}}, C^{\substack{\leftrightarrow\\\leftrightarrow\\\leftrightarrow}}\}$ be the set of all the correct encodings of complete labellings as identified by Theorem 3. The Hasse diagram for these 18 correct encodings is reported in Figure 2. Correct and non-redundant encodings are the minimal ones under set inclusion and correspond to the five encodings identified in Proposition 5.

4.2. Logical Encodings of Complete Labelling Constraints

We then need to encode the specific constraints that characterise the complete labellings returned as output of the various functions considered in the algorithms presented in Section 3. In this subsection we identify different ways such constraints can be encoded, thus completing the analysis at the base of the implementation of the algorithms detailed in Section 5.

Let us first consider the algorithms described in Section 3.1 relevant to the stable semantics, which exploit the functions $\underline{\mathsf{FindST}}^{\neq}$ and $\underline{\mathsf{FindST}_{\mathbf{a}}}$ (where $\mathbf{a}$ is an argument). Both of them return a stable labelling $\mathcal{L}ab$, which is defined as a complete labelling without arguments labelled `undec` (see Section 2) and can thus be obtained by constraining the encoding of a complete labelling with the condition $C_{st} \equiv \forall \mathbf{b} \in \mathcal{A}, \mathcal{L}ab(\mathbf{b}) \neq \texttt{undec}$.

Focusing on the function $\underline{\mathsf{FindST}}^{\neq}$, used in Algorithm 2, $\underline{\mathsf{FindST}}^{\neq}(\Gamma, \mathcal{L}_s)$ returns a stable labelling $\mathcal{L}ab$ of Γ which is not included in $\mathcal{L}_s$, where $\mathcal{L}_s$ is a set of stable labellings. For each $\mathcal{L}ab' \in \mathcal{L}_s$, the condition $\mathcal{L}ab \neq \mathcal{L}ab'$ must thus be encoded. Since there are no arguments labelled `undec` by stable labellings, a natural way to do it amounts to requiring that an argument labelled `in` by $\mathcal{L}ab'$ is labelled `out` by $\mathcal{L}ab$, or vice versa. While this can be encoded as a disjunction over the whole set of arguments, Proposition 7, based on Lemma 1, shows that such disjunction can be restricted to the arguments labelled `in` by $\mathcal{L}ab'$ (or $\mathcal{L}ab$), or equivalently on arguments labelled `out` by $\mathcal{L}ab'$ (or $\mathcal{L}ab$). Lemma 1 shows that an argument $\mathbf{a}$ is labelled `out` by a complete labelling $\mathcal{L}ab_1$ and `in` or `undec` by another complete labelling $\mathcal{L}ab_2$ if and only if another argument $\mathbf{b}$ is labelled `in` by $\mathcal{L}ab_1$ and `out` or `undec` by $\mathcal{L}ab_2$.

Lemma 1. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab_1, \mathcal{L}ab_2 \in \mathfrak{L}(\Gamma)$ complete labellings. $\exists \mathbf{a} \in \mathcal{A}$ such that $\mathcal{L}ab_1(\mathbf{a}) = \texttt{out}$ and $\mathcal{L}ab_2(\mathbf{a}) \neq \texttt{out}$ iff $\exists \mathbf{b} \in \mathcal{A}$ such that $\mathcal{L}ab_1(\mathbf{b}) = \texttt{in}$ and $\mathcal{L}ab_2(\mathbf{b}) \neq \texttt{in}$.

Proposition 7. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab, \mathcal{L}ab' \in \mathfrak{L}(\Gamma)$ stable labellings. All of the following statements are equivalent to $\mathcal{L}ab \neq \mathcal{L}ab'$:

1. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \texttt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \texttt{in}$
2. $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \texttt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \texttt{out}$
3. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \texttt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \texttt{in}$ and $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \texttt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \texttt{out}$

Summing up, the encoding corresponding to $\mathcal{L}ab = \underline{\mathsf{FindST}}^{\neq}(\Gamma, \mathcal{L}_s)$ can be expressed as $C \wedge C_{st} \wedge C_{\neq}$, where C is any encoding belonging to $\mathfrak{C}$ and $C_{\neq}$ is the conjunction of any of the three conditions of Proposition 7 applied to all labellings of $\mathcal{L}_s$. It can be noted that the third condition of Proposition 7 corresponds to the conjunction of the first two conditions: As with redundant encodings for complete labellings, we also consider redundant constraints that may improve the efficiency of SAT-based algorithms, introduced in Section 3.

As to the function $\underline{\mathsf{FindST}_{\mathbf{a}}}$, exploited in Algorithms 3 and 4, it returns a stable labelling $\mathcal{L}ab$ of Γ such that $\mathcal{L}ab(\mathbf{a})$ is a specific label $s \in \{\mathtt{in}, \mathtt{out}\}$. Obviously, the corresponding encoding is $C \wedge C_{st} \wedge \mathcal{L}ab(\mathbf{a}) = s$.

Let us now turn to preferred semantics and first consider the function $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}$ used in Algorithms 5 and 6 for the enumeration of preferred labellings and in Algorithm 8 for skeptical acceptance under preferred semantics. In particular, given an argumentation framework Γ and a set of preferred labellings $\mathcal{L}_p \subseteq \mathfrak{L}_{\mathsf{PR}}(\Gamma)$, $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_p)$ returns a complete labelling $\mathcal{L}ab$ such that $\forall \mathcal{L}ab' \in \mathcal{L}_p, \mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$. Proposition 8, in the following, shows that this constraint is equivalent to require an argument labelled $\mathtt{out}$ by $\mathcal{L}ab'$ to be labelled $\mathtt{in}$ by $\mathcal{L}ab$, or vice versa.

Proposition 8. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab' \in \mathfrak{L}_{\mathsf{PR}}(\Gamma)$ a preferred labelling. Then, $\forall \mathcal{L}ab \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$, all of the following statements are equivalent to $\mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$:

1. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \mathtt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$
2. $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \mathtt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \mathtt{out}$
3. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \mathtt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$ and $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \mathtt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \mathtt{out}$

According to Proposition 8, the constraint $\forall \mathcal{L}ab' \in \mathcal{L}_p, \mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$ can be expressed in the same way as in the case of the function $\underline{\mathsf{FindST}}^{\neq}$, i.e. the encoding corresponding to $\mathcal{L}ab = \underline{\mathsf{FindCL}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_p)$ is $C \wedge C_{\neq}$, where $C \in \mathfrak{C}$ and $C_{\neq}$ is the conjunction of any of the three conditions of Proposition 8 applied to all labellings of $\mathcal{L}_p$. Note that $C \wedge C_{\neq}$ captures all preferred labellings that do not belong to $\mathcal{L}_p$, complying with the definition of $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}$ (see Section 3.2).

As to the function $\underline{\mathsf{FindCL}}^{\sqsupsetneq}$, used again in Algorithms 5, 6 and 8, given an argumentation framework Γ and a complete labelling $\mathcal{L}ab'$, $\underline{\mathsf{FindCL}}^{\sqsupsetneq}(\Gamma, \mathcal{L}ab')$ returns a complete labelling $\mathcal{L}ab$ such that $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab$. This constraint amounts to require that $\mathtt{in}(\mathcal{L}ab') \subseteq \mathtt{in}(\mathcal{L}ab)$, $\mathtt{out}(\mathcal{L}ab') \subseteq \mathtt{out}(\mathcal{L}ab)$, and that there is an argument $\mathbf{a} \in \mathtt{undec}(\mathcal{L}ab')$ such that $\mathcal{L}ab(\mathbf{a}) \neq \mathtt{undec}$. Proposition 9, based on Lemma 1, shows that this constraint can be strengthened by replacing $\mathcal{L}ab(\mathbf{a}) \neq \mathtt{undec}$ with either $\mathcal{L}ab(\mathbf{a}) = \mathtt{in}$ or $\mathcal{L}ab(\mathbf{a}) = \mathtt{out}$.

Proposition 9. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab, \mathcal{L}ab' \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$. All of the following statements are equivalent to $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab$:

1. $\mathtt{in}(\mathcal{L}ab') \subseteq \mathtt{in}(\mathcal{L}ab) \wedge \mathtt{out}(\mathcal{L}ab') \subseteq \mathtt{out}(\mathcal{L}ab) \wedge \exists \mathbf{b} \in \mathtt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{b}) = \mathtt{out}$
2. $\mathtt{in}(\mathcal{L}ab') \subseteq \mathtt{in}(\mathcal{L}ab) \wedge \mathtt{out}(\mathcal{L}ab') \subseteq \mathtt{out}(\mathcal{L}ab) \wedge \exists \mathbf{c} \in \mathtt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{c}) = \mathtt{in}$

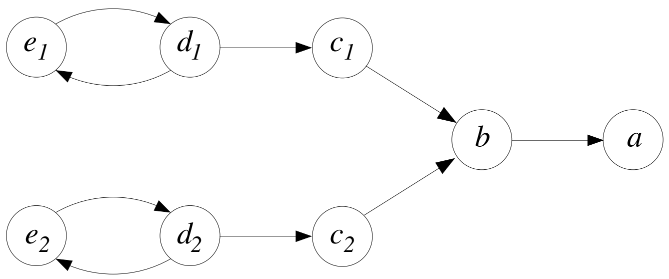

Figure 3: A counterexample to the direct use of Proposition 8.

3. $\texttt{in}(\mathcal{L}ab') \subseteq \texttt{in}(\mathcal{L}ab) \wedge \texttt{out}(\mathcal{L}ab') \subseteq \texttt{out}(\mathcal{L}ab) \wedge \exists \mathbf{b} \in \texttt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{b}) = \texttt{out} \wedge \exists \mathbf{c} \in \texttt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{c}) = \texttt{in}$

According to the proposition above, the encoding corresponding to $\mathcal{L}ab = \underline{\mathsf{FindCL}}^{\not\sqsupseteq}(\Gamma, \mathcal{L}ab')$ is $C \wedge C_{\not\sqsupseteq}$, where $C \in \mathfrak{C}$ and $C_{\not\sqsupseteq}$ is any of the three conditions of Proposition 9.

We now consider the function $\underline{\mathsf{FindCL}_{\mathbf{a}}}$, which is exploited by Algorithms 7 and 9 to identify a complete labelling where $\mathbf{a}$ is $\texttt{in}$ and by Algorithm 9 to identify a complete labelling where $\mathbf{a}$ is $\texttt{out}$. Accordingly, the encoding corresponding to the labelling $\mathcal{L}ab = \underline{\mathsf{FindCL}_{\mathbf{a}}}(\Gamma, s)$, where $s \in \{\texttt{in}, \texttt{out}\}$, is simply $C \wedge \mathcal{L}ab(\mathbf{a}) = s$, where $C \in \mathfrak{C}$.

Let us finally turn to the functions appearing in Algorithm 9, namely $\underline{\mathsf{FindCLU}_{\mathbf{a}}^{\not\sqsubseteq}}$ and $\underline{\mathsf{FindCL}_{\mathbf{a}}^{\not\sqsupseteq}}$, which roughly correspond to the functions $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}$ and $\underline{\mathsf{FindCL}}^{\not\sqsupseteq}$, respectively, but with an additional constraint on the label assigned to $\mathbf{a}$ by the output labelling $\mathcal{L}ab$.

In particular, given an argumentation framework Γ and a set of complete labellings $\mathcal{L}_c \subseteq \mathcal{L}_{mu}$, $\underline{\mathsf{FindCLU}_{\mathbf{a}}^{\not\sqsubseteq}}(\Gamma, \mathcal{L}_c)$ returns a complete labelling $\mathcal{L}ab$ such that $\mathcal{L}ab(\mathbf{a}) = \texttt{undec}$ and $\forall \mathcal{L}ab' \in \mathcal{L}_c, \mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$. It would be tempting to directly apply Proposition 8 to this case, however the example depicted in Figure 3 shows that the corresponding result does not hold in general under the hypothesis that $\mathcal{L}_c \subseteq \mathcal{L}_{mu}$. Indeed, there are only three complete labellings such that $\mathbf{a}$ is $\texttt{undec}$, namely the grounded labelling where all arguments are labelled $\texttt{undec}$, $\mathcal{L}ab = \{(\mathbf{e}_1, \texttt{undec}), (\mathbf{d}_1, \texttt{undec}), (\mathbf{c}_1, \texttt{undec})(\mathbf{e}_2, \texttt{out}), (\mathbf{d}_2, \texttt{in}), (\mathbf{c}_2, \texttt{out}), (\mathbf{b}, \texttt{undec}), (\mathbf{a}, \texttt{undec})\}$ and $\mathcal{L}ab' = \{(\mathbf{e}_1, \texttt{out}), (\mathbf{d}_1, \texttt{in}), (\mathbf{c}_1, \texttt{out}), (\mathbf{e}_2, \texttt{undec}), (\mathbf{d}_2, \texttt{undec}), (\mathbf{c}_2, \texttt{undec}), (\mathbf{b}, \texttt{undec}), (\mathbf{a}, \texttt{undec})\}$. The grounded labelling is not maximal w.r.t. $\sqsubseteq$, thus $\mathcal{L}_{mu} = \{\mathcal{L}ab, \mathcal{L}ab'\}$. It can be easily checked that, despite $\mathcal{L}ab' \in \mathcal{L}_{mu}$ and $\mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$, none of the conditions of Proposition 8 holds, i.e. there is no argument which is labelled $\texttt{out}$ by $\mathcal{L}ab'$ and $\texttt{in}$ by $\mathcal{L}ab$, or vice versa. On the other hand, the following proposition shows that these conditions are satisfied under the hypothesis that $\mathcal{L}ab$ is a preferred labelling.

Proposition 10. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework, $\mathbf{a}$ an argument in $\mathcal{A}$ and $\mathcal{L}ab' \in \mathfrak{L}(\Gamma)$ a maximal complete labelling such that $\mathcal{L}ab'(\mathbf{a}) = \texttt{undec}$, i.e. $\nexists \mathcal{L}ab^* \in \mathfrak{L}(\Gamma)$ such that $\mathcal{L}ab^*(\mathbf{a}) = \texttt{undec}$ and $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab^*$. Then, $\forall \mathcal{L}ab \in \mathfrak{L}_{\mathsf{PR}}(\Gamma)$ such that $\mathcal{L}ab(\mathbf{a}) = \texttt{undec}$, all of the following statements are equivalent to $\mathcal{L}ab \neq \mathcal{L}ab'$:

1. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \texttt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \texttt{in}$
2. $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \texttt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \texttt{out}$
3. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \texttt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \texttt{in}$ and $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \texttt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \texttt{out}$

According to the proposition above, the encoding corresponding to $\mathcal{L}ab = \underline{\mathsf{FindCLU}}^{\not\sqsubseteq}_{\mathbf{a}}(\Gamma, \mathcal{L}_c)$ is $C \wedge C_{\neq} \wedge \mathcal{L}ab(\mathbf{a}) = \texttt{undec}$, where $C \in \mathfrak{C}$ and $C_{\neq}$ is the conjunction of any of the three conditions of Proposition 10 applied to all labellings of $\mathcal{L}_c$. Indeed, the set of labellings satisfying these constraints complies with the set $\mathcal{L}$ in the definition of $\underline{\mathsf{FindCLU}}^{\not\sqsubseteq}_{\mathbf{a}}$ (see Section 3.2): it is a subset of the complete labellings (due to C) where $\mathbf{a}$ is labelled $\texttt{undec}$ (due to $\mathcal{L}ab(\mathbf{a}) = \texttt{undec}$) which does not include those labellings that are less or equally committed w.r.t. a labelling of $\mathcal{L}_c$ (due to the conditions of $C_{\neq}$), while including all preferred labellings where $\mathbf{a}$ is $\texttt{undec}$ that are outside $\mathcal{L}_c$ (this is guaranteed by Proposition 10).

We have finally to consider the function $\underline{\mathsf{FindCL}}^{\sqsupsetneq}_{\mathbf{a}}$, which given an argument $\mathbf{a}$, an argumentation framework Γ and a complete labelling $\mathcal{L}ab' \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$ returns a complete labelling $\mathcal{L}ab$ such that $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab$ and $\mathcal{L}ab(\mathbf{a}) = s$, with $s \in \{\texttt{in}, \texttt{undec}\}$. Taking into account Proposition 9, it is easy to see that the encoding corresponding to $\mathcal{L}ab$ is $C \wedge C_{\sqsupsetneq} \wedge \mathcal{L}ab(\mathbf{a}) = s$, where again $C \in \mathfrak{C}$ and $C_{\sqsupsetneq}$ is any of the three conditions of Proposition 9.

5. Algorithms Implementation

In this section we present the detailed pseudo-codes[7] implementing the abstract algorithms presented in Section 3.

In order to allow us to explore all the different encodings of complete labellings and the relevant constraints, we will exploit a number of parameters, denoted as π^*_*, and select algorithmic variants with conditional statements based on them: Table B.5 summarises them.

Before presenting the detailed algorithms, we provide some details on the encoding of complete labellings as required by existing SAT solvers.

5.1. SAT Encodings of Complete Labellings

To exploit existing SAT solvers, we need first to express the correct constraints of $\mathfrak{C}$ identified in Theorem 3 as a formula in conjunctive normal form. To this

[7]The actual C++ code is available at `https://sourceforge.net/projects/argsemsat/` (on 11 May 2018)

purpose, for each argument $\mathbf{a} \in \mathcal{A}$ we define three boolean variables,[8] $I_{\mathbf{a}}$, $O_{\mathbf{a}}$, and $U_{\mathbf{a}}$, with the intended meaning that $I_{\mathbf{a}}$ is true when argument $\mathbf{a}$ is labelled `in`, false otherwise, and analogously $O_{\mathbf{a}}$ and $U_{\mathbf{a}}$ correspond to labels `out` and `undec`. Formally, given $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ we define the corresponding set of variables as $\mathcal{V}(\Gamma) \triangleq \bigcup_{\mathbf{a} \in \mathcal{A}} \{I_{\mathbf{a}}, O_{\mathbf{a}}, U_{\mathbf{a}}\}$. The constraints of Definition 7 can then be expressed in terms of the variables $\mathcal{V}(\Gamma)$. The detail of the resulting CNF formula is given in Proposition 11: proof is omitted as it follows from straightforward manipulations.

Proposition 11. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$:

$$C_{\mathtt{in}}^{\leftarrow} \equiv \bigwedge_{\{\mathbf{a} \in \mathcal{A}\}} \left(I_{\mathbf{a}} \vee \left(\bigvee_{\{\mathbf{b} \ | \ \mathbf{b} \rightarrow \mathbf{a}\}} (\neg O_{\mathbf{b}}) \right) \right)$$

$$C_{\mathtt{in}}^{\rightarrow} \equiv \bigwedge_{\{\mathbf{a} \in \mathcal{A}\}} \left(\bigwedge_{\{\mathbf{b} \ | \ \mathbf{b} \rightarrow \mathbf{a}\}} \neg I_{\mathbf{a}} \vee O_{\mathbf{b}} \right)$$

$$C_{\mathtt{out}}^{\leftarrow} \equiv \bigwedge_{\{\mathbf{a} \in \mathcal{A}\}} \left(\bigwedge_{\{\mathbf{b} \ | \ \mathbf{b} \rightarrow \mathbf{a}\}} \neg I_{\mathbf{b}} \vee O_{\mathbf{a}} \right)$$

$$C_{\mathtt{out}}^{\rightarrow} \equiv \bigwedge_{\{\mathbf{a} \in \mathcal{A}\}} \left(\neg O_{\mathbf{a}} \vee \left(\bigvee_{\{\mathbf{b} \ | \ \mathbf{b} \rightarrow \mathbf{a}\}} I_{\mathbf{b}} \right) \right)$$

$$C_{\mathtt{undec}}^{\leftarrow} \equiv \bigwedge_{\{\mathbf{a} \in \mathcal{A}\}} \left(\bigwedge_{\{\mathbf{b} \ | \ \mathbf{b} \rightarrow \mathbf{a}\}} \left(U_{\mathbf{a}} \vee \neg U_{\mathbf{b}} \vee \left(\bigvee_{\{\mathbf{c} \ | \ \mathbf{c} \rightarrow \mathbf{a}\}} I_{\mathbf{c}} \right) \right) \right)$$

[8]It is worth noticing that variations of the proposed algorithm can be used with fewer variables. For instance, there are algorithms for preferred semantics using only one variable per argument, cf. [32, 33]. In ICCMA-15 an earlier version of ArgSemSAT still using three boolean variables per argument resulted faster than such algorithms when computing the skeptical acceptance of arguments, despite relying on an external SAT solver as a new process. For what concerns stable semantics, whose definition does not make use of the `undec` label, in Appendix C an interested reader can find the results of a pilot study aimed at quantifying an alleged improvement in using one variable instead of three when enumerating stable extensions. Our preliminary results show cases where there is not a statistical significant change; and when there is one, it might be that the version using three variables has a lower median runtime. It seems to us that there is evidence that performance does not necessary improve when considering fewer propositional variables when transforming the constraints imposed by semantics into CNFs.

$$C^{\rightarrow}_{\texttt{undec}} \equiv \bigwedge_{\{\mathbf{a}\in\mathcal{A}\}} \left(\left(\bigwedge_{\{\mathbf{b} \mid \mathbf{b}\rightarrow\mathbf{a}\}} (\neg U_{\mathbf{a}} \vee \neg I_{\mathbf{b}}) \right) \wedge \left(\neg U_{\mathbf{a}} \vee \left(\bigvee_{\{\mathbf{b} \mid \mathbf{b}\rightarrow\mathbf{a}\}} U_{\mathbf{b}} \right) \right) \right)$$

Furthermore, since for each argument **a** the variables $I_{\mathbf{a}}$, $O_{\mathbf{a}}$, and $U_{\mathbf{a}}$ are independent, an additional condition is needed to enforce an assignment corresponding to a total function, i.e. to guarantee that exactly one of the variables is assigned the value True. Moreover, as a small engineering improvement we explicitly constrain unattacked arguments to be labelled `in`, and for technical reasons we restrict to "non-empty" labellings (in the sense that at least one of the arguments is labelled `in`). In order to express the resulting CNF identifying non-empty complete labellings of a given argumentation framework, let us define a function `cnf` that receives as input an element of $\mathfrak{C}$ and an argumentation framework Γ, and returns the equivalent CNF representation as specified by Proposition 11, in conjunction with the following formulæ:

$$\bigwedge_{\mathbf{a}\in\mathcal{A}} \Big((I_{\mathbf{a}} \vee O_{\mathbf{a}} \vee U_{\mathbf{a}}) \wedge (\neg I_{\mathbf{a}} \vee \neg O_{\mathbf{a}}) \wedge (\neg I_{\mathbf{a}} \vee \neg U_{\mathbf{a}}) \wedge (\neg O_{\mathbf{a}} \vee \neg U_{\mathbf{a}}) \Big) \tag{1}$$

$$\bigwedge_{\{\mathbf{a} \mid \mathbf{a}^{-}=\emptyset\}} (I_{\mathbf{a}} \wedge \neg O_{\mathbf{a}} \wedge \neg U_{\mathbf{a}}) \tag{2}$$

$$\bigvee_{\mathbf{a}\in\mathcal{A}} I_{\mathbf{a}} \tag{3}$$

In particular, (1) enforces a total function, (2) constrains an unattacked argument to be labelled `in`, and (3) enforces that at least one argument is labelled `in`, thus forcing the search for a non-empty labelling.

5.2. Algorithms for Stable Semantics

In this section we detail the implementation of the algorithms presented in Section 3.1 on the basis of the encodings identified in Section 4.

To this purpose, we first need to introduce some auxiliary functions referring to a generic argumentation framework $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$.

The call to a SAT solver is expressed by means of the function SatS, which receives as input a CNF formula and returns a labelling over $\mathcal{A}$ satisfying the constraints expressed by the formula, or $\emptyset$ if the formula is not satisfiable.

Similarly, ALLSatS represents a call to an AllSAT solver. It receives as input a CNF formula as well as two parameters $\pi^{\mathsf{S}}_{\mathsf{All;O}}$ and $\pi^{\mathsf{S}}_{\mathsf{All;I}}$, and returns two elements, where the first is the (possibly empty) set of labellings over $\mathcal{A}$ satisfying the constraints expressed by the formula, and the second element is the set of blocking formulæ generated in the process. This second element is sometimes needed to exclude from the subsequent search the set of labellings returned by $\mathsf{ALLSatS}$. Specifically, the implementation exploits the AllSAT solver developed by [28] with the only modification to enable it to return the set of blocking clauses along with the set of the found models. The input parameters $\pi^{\mathsf{S}}_{\mathsf{All;O}}$ and $\pi^{\mathsf{S}}_{\mathsf{All;I}}$ allow one to choose the form of the blocking clauses according to Proposition 7. In particular, $\pi^{\mathsf{S}}_{\mathsf{All;O}}$ generates for each found labelling a blocking clause based on arguments labelled `in`, requiring one of them to be labelled `out`, while $\pi^{\mathsf{S}}_{\mathsf{All;I}}$ generates a blocking clause that considers only arguments labelled `out`. As shown below, they are in essence analogous to the parameters $\pi^{\mathsf{S}}_{\mathsf{O}}$ and $\pi^{\mathsf{S}}_{\mathsf{I}}$ used at lines 8–13 of Algorithm 10.

Finally, $\mathsf{I\text{-}ARGS}$ (resp. $\mathsf{O\text{-}ARGS}$, $\mathsf{U\text{-}ARGS}$) receives as input a labelling and returns the set of arguments labelled `in` (resp. `out`, `undec`).

Algorithm 10 details the implementation of Algorithm 2 for the enumeration of stable labellings. Besides $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, the algorithm receives as input the specific encoding $C* \in \mathfrak{C}$ of complete labellings to be exploited (see Section 4.1) as well as five parameters, i.e. $\pi^{\mathsf{S}}_{\mathsf{All}}$, $\pi^{\mathsf{S}}_{\mathsf{All;O}}$, $\pi^{\mathsf{S}}_{\mathsf{All;I}}$, $\pi^{\mathsf{S}}_{\mathsf{O}}$, and $\pi^{\mathsf{S}}_{\mathsf{I}}$. The first one enables the algorithm to directly exploit an AllSAT solver to identify all the stable labellings, using in this case as input parameters $\pi^{\mathsf{S}}_{\mathsf{All;O}}$ and $\pi^{\mathsf{S}}_{\mathsf{All;I}}$, while $\pi^{\mathsf{S}}_{\mathsf{O}}$ and $\pi^{\mathsf{S}}_{\mathsf{I}}$, in case the AllSAT solver is not exploited, allow one to select the way to encode the constraint relevant to the function $\underline{\mathsf{FindST}}^{\neq}$.

In particular, if $\pi^{\mathsf{S}}_{\mathsf{All}} = \top$ then the AllSAT solver is called at line 2 on a CNF which encodes the complete labellings without undecided arguments, i.e. satisfying the additional requirement $\bigwedge_{\mathbf{a} \in \mathcal{A}} \neg U_{\mathbf{a}}$. In the case $\pi^{\mathsf{S}}_{\mathsf{All}} = \bot$, the function $\underline{\mathsf{FindST}}^{\neq}(\cdot, \cdot)$, used at line 2 of Algorithm 2, is implemented within lines 6–15. Line 6 identifies a stable labelling (if this exists) that has not been identified in an earlier execution of the cycle. Proposition 7 shows that there are three possibilities to ensure that: an argument labelled `in` should then become labelled `out`, implemented at line 9 if $\pi^{\mathsf{S}}_{\mathsf{O}} = \top$; an argument labelled `out` should then become labelled `in`, implemented at line 12 if $\pi^{\mathsf{S}}_{\mathsf{I}} = \top$; or both of the previous ones, in case $\pi^{\mathsf{S}}_{\mathsf{O}} = \top \ \wedge \ \pi^{\mathsf{S}}_{\mathsf{I}} = \top$. As discussed in Section 4.2, we leave room for all the three possibilities, including the redundant constraint, since logically equivalent encodings may yield different behaviours from the efficiency point of view.

Note that the algorithm returns as output also the blocking formulæ used to exclude the generated solutions from the search. These are useful in case the algorithm is exploited as a first step of the enumeration of preferred labellings (cf. Algorithm 13).

Moving to credulous and skeptical acceptance, Algorithm 11 uses a call to a SAT solver (line 1) to implement in a rather straightforward fashion the function $\underline{\mathsf{FindST}}_{\mathbf{x}}(\cdot, \cdot)$ called at line 1 of Algorithm 3. Similarly, at line 1 of Algorithm 12 the function $\underline{\mathsf{FindST}}_{\mathbf{x}}(\cdot, \cdot)$ used at line 1 of Algorithm 4 is implemented with a simple call to a SAT Solver.

Algorithm 10 Enumeration of Stable Labellings using SAT solvers

```
    SAT-EL-ST
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $C* \in \mathfrak{C}$, $\langle \pi^{\mathsf{S}}_{\mathsf{All}}, \pi^{\mathsf{S}}_{\mathsf{All;O}}, \pi^{\mathsf{S}}_{\mathsf{All;I}}, \pi^{\mathsf{S}}_{\mathsf{O}}, \pi^{\mathsf{S}}_{\mathsf{I}} \rangle \in \{\top, \bot\}^5$
    Output: $\langle L_s \subseteq \mathfrak{L}_{\mathsf{ST}}(\Gamma),\ blocking \rangle$
 1: if $\pi^{\mathsf{S}}_{\mathsf{All}}$ then
 2:   return $\mathsf{ALLSatS}\left(\mathsf{cnf}(C*, \Gamma) \ \wedge \ \bigwedge_{\mathbf{a} \in \mathcal{A}} \neg U_{\mathbf{a}}, \pi^{\mathsf{S}}_{\mathsf{All;I}}, \pi^{\mathsf{S}}_{\mathsf{All;O}}\right)$
 3: end if
 4: $L_s \ := \ \varnothing, \ blocking \ := \ \top$
 5: do
 6:   $stb \ := \ \mathsf{SatS}\left(\mathsf{cnf}(C*, \Gamma) \ \wedge \ \bigwedge_{\mathbf{a} \in \mathcal{A}} \neg U_{\mathbf{a}} \ \wedge \ blocking\right)$
 7:   if $stb \ \neq \ \varnothing$ then
 8:     if $\pi^{\mathsf{S}}_{\mathsf{O}}$ then
 9:       $blocking \ := \ blocking \wedge \bigvee_{\mathbf{a} \in \mathsf{I\text{-}ARGS}(stb)} O_{\mathbf{a}}$
10:     end if
11:     if $\pi^{\mathsf{S}}_{\mathsf{I}}$ then
12:       $blocking \ := \ blocking \wedge \bigvee_{\mathbf{a} \in \mathsf{O\text{-}ARGS}(stb)} I_{\mathbf{a}}$
13:     end if
14:     $L_s \ := \ L_s \cup \{stb\}$
15:   end if
16: while $stb \ \neq \ \varnothing$
17: return $\langle L_s, blocking \rangle$
```

Algorithm 11 Credulous acceptance under Stable Semantics using SAT solvers

```
    SAT-DC-ST
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle, \mathbf{x} \in \mathcal{A}$, $C* \in \mathfrak{C}$
    Output: $\{\top, \bot\}$
 1: return $\left(\mathsf{SatS}\left(\mathsf{cnf}(C*, \Gamma) \ \wedge \ \bigwedge_{\mathbf{a} \in \mathcal{A}} \neg U_{\mathbf{a}} \ \wedge \ I_{\mathbf{x}}\right) \neq \varnothing\right)$
```

Algorithm 12 Skeptical acceptance under Stable Semantics using SAT solvers

```
    SAT-DS-ST
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle, \mathbf{x} \in \mathcal{A}$, $C* \in \mathfrak{C}$
    Output: $\{\top, \bot\}$
 1: if $\mathsf{SatS}\left(\mathsf{cnf}(C*, \Gamma) \ \wedge \ \bigwedge_{\mathbf{a} \in \mathcal{A}} \neg U_{\mathbf{a}} \ \wedge \ O_{\mathbf{x}}\right) \neq \varnothing$ then
 2:   return $\bot$
 3: end if
 4: return $\top$
```

Algorithm 13 Enumeration of Preferred Labellings using SAT solvers

```
    SAT-EL-PR
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $C* \in \mathfrak{C}$, $\langle \pi^{\mathsf{P}}_{\mathsf{S}}, \pi^{\mathsf{S}}_{\mathsf{All}}, \pi^{\mathsf{S}}_{\mathsf{All;O}}, \pi^{\mathsf{S}}_{\mathsf{All;I}}, \pi^{\mathsf{S}}_{\mathsf{O}}, \pi^{\mathsf{S}}_{\mathsf{I}}, \pi^{\mathsf{P}}_{\mathsf{iO}}, \pi^{\mathsf{P}}_{\mathsf{iI}}, \pi^{\mathsf{P}}_{\mathsf{eO}}, \pi^{\mathsf{P}}_{\mathsf{eI}} \rangle \in \{\top, \bot\}^{10}$
    Output: $L_s \subseteq \mathfrak{L}_{\mathsf{ST}}(\Gamma)$
1:  $L_p := \emptyset$, $blocking := \top$
2:  if $\pi^{\mathsf{P}}_{\mathsf{S}}$ then
3:    $\langle L_p, blocking \rangle :=$ SAT-EL-ST $\left(\Gamma,\ C*,\ \langle \pi^{\mathsf{S}}_{\mathsf{All}}, \pi^{\mathsf{S}}_{\mathsf{All;O}}, \pi^{\mathsf{S}}_{\mathsf{All;I}}, \pi^{\mathsf{S}}_{\mathsf{O}}, \pi^{\mathsf{S}}_{\mathsf{I}} \rangle\right)$
4:  end if
5:  do
6:    $iblock := \top$, $prf := \emptyset$
7:    do
8:      $cmp := \mathsf{SatS}\left(\mathsf{cnf}(C*, \Gamma) \wedge blocking \wedge iblock\right)$
9:      if $cmp\ ! = \emptyset$ then
10:       $prf := cmp$
11:       $iblock := \bigwedge_{\mathbf{a} \in \mathsf{I\text{-}ARGS}(cmp)} I_{\mathbf{a}} \wedge \bigwedge_{\mathbf{a} \in \mathsf{O\text{-}ARGS}(cmp)} O_{\mathbf{a}}$
12:       if $\pi^{\mathsf{P}}_{\mathsf{iO}}$ then
13:         $iblock := iblock \wedge \bigvee_{\mathbf{a} \in \mathsf{U\text{-}ARGS}(cmp)} O_{\mathbf{a}}$
14:       end if
15:       if $\pi^{\mathsf{P}}_{\mathsf{iI}}$ then
16:         $iblock := iblock \wedge \bigvee_{\mathbf{a} \in \mathsf{U\text{-}ARGS}(cmp)} I_{\mathbf{a}}$
17:       end if
18:     end if
19:   while $cmp \neq \emptyset \wedge \mathsf{U\text{-}ARGS}(cmp) \neq \emptyset$
20:   if $prf \neq \emptyset$ then
21:     $L_p := L_p \cup \{prf\}$
22:     if $\pi^{\mathsf{P}}_{\mathsf{eO}}$ then
23:       $blocking := blocking \wedge \bigvee_{\mathbf{a} \in \mathsf{I\text{-}ARGS}(prf)} O_{\mathbf{a}}$
24:     end if
25:     if $\pi^{\mathsf{P}}_{\mathsf{eI}}$ then
26:       $blocking := blocking \wedge \bigvee_{\mathbf{a} \in \mathsf{O\text{-}ARGS}(prf)} I_{\mathbf{a}}$
27:     end if
28:   end if
29: while $prf \neq \emptyset$
30: if $L_p = \emptyset$ then
31:   $L_p = \{\mathcal{L}\mathrm{U}\}$
32: end if
33: return $L_p$
```

5.3. Algorithms for Preferred Semantics

Algorithm 13 details the implementation of Algorithms 5 and 6 for the enumeration of preferred labellings. It receives in input an argumentation framework $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, the encoding $C* \in \mathfrak{C}$ (with the same meaning as in the case of the previous algorithms), and 10 configuration parameters. The first parameter $\pi^{\mathsf{P}}_{\mathsf{S}}$ is the selector between Algorithms 5 and 6, while the other 9 parameters specify how to encode the constraints relevant to the non-deterministic functions corresponding to the SAT solver invocations. In particular, $\langle \pi^{\mathsf{S}}_{\mathsf{All}}, \pi^{\mathsf{S}}_{\mathsf{All;O}}, \pi^{\mathsf{S}}_{\mathsf{All;I}}, \pi^{\mathsf{S}}_{\mathsf{O}}, \pi^{\mathsf{S}}_{\mathsf{I}} \rangle$ are related to the enumeration of stable labellings and have the same meaning as detailed in the previous section.

If $\pi^{\mathsf{P}}_{\mathsf{S}} = \top$ then the algorithm **SAT-EL-ST** is called at line 3, corresponding to line 1 of Algorithm 6, passing it the parameters $\langle \pi^{\mathsf{S}}_{\mathsf{All}}, \pi^{\mathsf{S}}_{\mathsf{All;O}}, \pi^{\mathsf{S}}_{\mathsf{All;I}}, \pi^{\mathsf{S}}_{\mathsf{O}}, \pi^{\mathsf{S}}_{\mathsf{I}} \rangle$ it expects. Then the outer loop of Algorithms 5 and 6 correspond to lines 5-29 of Algorithm 13, the inner loop to lines 7-19. The function $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}$, used at line 2 of Algorithms 5 and 6, is implemented within lines 8 and 22–27. Line 8 identifies a non-empty complete labelling that is not less committed w.r.t. any found preferred labelling. Proposition 8 shows that there are three ways to ensure that: an argument labelled `in` should then become labelled `out` (cf. line 23); an argument labelled `out` should then become labelled `in` (cf. line 26); or both of the previous ones. Those options are controlled by the parameters $\pi^{\mathsf{P}}_{\mathsf{eO}}$ and $\pi^{\mathsf{P}}_{\mathsf{eI}}$.

Moreover, the function $\underline{\mathsf{FindCL}}^{\sqsupsetneq}$, used at line 3 of Algorithm 6, is implemented within lines 8–18 of Algorithm 13. In particular, as from the second iteration of the inner loop, line 8 identifies a new complete labelling which is strictly more committed w.r.t. a previously found complete labelling. Proposition 9 shows that there are three ways to ensure that: besides maintaining the labels of arguments labelled `in` and `out` (line 11), an argument labelled `undec` should then become `out` (line 13); or an argument labelled `undec` should then become `in` (line 16); or both conditions can be enforced. Those options are controlled by the parameters $\pi^{\mathsf{P}}_{\mathsf{iO}}$ and $\pi^{\mathsf{P}}_{\mathsf{iI}}$.

It should be noted that if in the argumentation framework there is only a preferred extension which is empty, then due to the non-emptiness condition in the encodings of $\mathfrak{C}$ no labelling is identified in the main loop. In this case, the algorithm correctly returns a unique labelling, denoted as $\mathcal{L}$U, assigning the label `undec` to all arguements (line 31).

While the credulous acceptance w.r.t. preferred semantics is rather straightforward—see Algorithms 7 and 14—and we report it only for completeness, skeptical acceptance needs a little further explanation.

First of all, Algorithm 15 implements Algorithm 8, which as shown in Section 3.2 is a direct extension of the algorithm for enumerating preferred labellings to the case of checking skeptical acceptance. As such, the input parameters are exactly those of Algorithm 13.

Finally, Algorithm 16 implements Algorithm 9 which represents an improved algorithm for skeptical acceptance. As shown in Section 4.2, the constraints on the complete labelling $\mathcal{L}ab$ returned by the functions $\underline{\mathsf{FindCLU}}^{\not\sqsubseteq}_{\mathbf{a}}$ and $\underline{\mathsf{FindCL}}^{\sqsupsetneq}_{\mathbf{a}}$ are the same as those relevant to the functions $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}$ and $\underline{\mathsf{FindCL}}^{\sqsupsetneq}$, respectively, with the addition of the condition $\mathcal{L}ab(\mathbf{a}) = \texttt{undec}$ or $\mathcal{L}ab(\mathbf{a}) = \texttt{in}$. As a consequence, the input parameters $\langle \pi^{\mathsf{P}}_{\mathsf{iO}}, \pi^{\mathsf{P}}_{\mathsf{iI}}, \pi^{\mathsf{P}}_{\mathsf{eO}}, \pi^{\mathsf{P}}_{\mathsf{eI}} \rangle$ have the same meaning as in Algorithm 13. In particular, it is easy to see that $\underline{\mathsf{FindCLU}}^{\sqsubset\!\!\!\not\sqsupset}_{\mathbf{a}}(\Gamma, \mathcal{L}_c)$ is implemented within lines 8-28, with the relevant constraint enforced in lines 23-28, while $\underline{\mathsf{FindCL}}^{\overline{\neq}}_{\mathbf{a}}(\Gamma, \mathcal{L}ab, \texttt{undec})$ is implemented within lines 8-16, with the relevant constraint enforced in lines 10-16.

Algorithm 14 Credulous acceptance under Preferred Semantics using SAT solvers

```
    SAT-DC-PR
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $\mathbf{x} \in \mathcal{A}$, $C* \in \mathfrak{C}$
    Output: $\{\top, \bot\}$
 1: return $(\mathsf{SatS}\,(\mathsf{cnf}(C*, \Gamma) \ \wedge \ I_{\mathbf{x}}) \neq \emptyset)$
```

Algorithm 15 Skeptical acceptance under Preferred Semantics using SAT solvers 1

```
    SAT-DS-PR-straightforward
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle, \mathbf{x} \in \mathcal{A}, C* \in \mathfrak{C}, \langle \pi^{\mathsf{P}}_{\mathsf{S}}, \pi^{\mathsf{S}}_{\mathsf{AII}}, \pi^{\mathsf{S}}_{\mathsf{AII;O}}, \pi^{\mathsf{S}}_{\mathsf{AII;I}}, \pi^{\mathsf{S}}_{\mathsf{O}}, \pi^{\mathsf{S}}_{\mathsf{I}}, \pi^{\mathsf{P}}_{\mathsf{iO}}, \pi^{\mathsf{P}}_{\mathsf{iI}}, \pi^{\mathsf{P}}_{\mathsf{eO}}, \pi^{\mathsf{P}}_{\mathsf{eI}} \rangle \in \{\top, \bot\}^{10}$
    Output: $\{\top, \bot\}$
 1: $L_p := \varnothing,\ blocking := \top$
 2: if $\pi^{\mathsf{P}}_{\mathsf{S}}$ then
 3:   $\langle L_p, blocking \rangle := \textbf{SAT-EL-ST}\left(\Gamma,\ C*,\ \langle \pi^{\mathsf{S}}_{\mathsf{AII}}, \pi^{\mathsf{S}}_{\mathsf{AII;O}}, \pi^{\mathsf{S}}_{\mathsf{AII;I}}, \pi^{\mathsf{S}}_{\mathsf{O}}, \pi^{\mathsf{S}}_{\mathsf{I}} \rangle\right)$
 4:   for all $aL_p \in L_p$ do
 5:     if $\mathbf{x} \notin \mathsf{I\text{-}ARGS}(aL_p)$ then
 6:       return $\bot$
 7:     end if
 8:   end for
 9: end if
10: do
11:   $iblock := \top,\ prf := \varnothing$
12:   do
13:     $cmp := \mathsf{SatS}\,(\mathsf{cnf}(C*, \Gamma) \ \wedge blocking \ \wedge iblock)$
14:     if $cmp \neq \varnothing$ then
15:       $prf := cmp,\ iblock := \bigwedge_{\mathbf{a} \in \mathsf{I\text{-}ARGS}(cmp)} I_{\mathbf{a}} \wedge \bigwedge_{\mathbf{a} \in \mathsf{O\text{-}ARGS}(cmp)} O_{\mathbf{a}}$
16:       if $\pi^{\mathsf{P}}_{\mathsf{iO}}$ then
17:         $iblock := iblock \wedge \bigvee_{\mathbf{a} \in \mathsf{U\text{-}ARGS}(cmp)} O_{\mathbf{a}}$
18:       end if
19:       if $\pi^{\mathsf{P}}_{\mathsf{iI}}$ then
20:         $iblock := iblock \wedge \bigvee_{\mathbf{a} \in \mathsf{U\text{-}ARGS}(cmp)} I_{\mathbf{a}}$
21:       end if
22:     end if
23:   while $cmp \neq \varnothing \wedge \mathsf{U\text{-}ARGS}(cmp) \neq \varnothing$
24:   if $prf \neq \varnothing$ then
25:     if $\mathbf{x} \notin \mathsf{I\text{-}ARGS}(prf)$ then
26:       return $\bot$
27:     end if
28:     $L_p := L_p \cup \{prf\}$
29:     if $\pi^{\mathsf{P}}_{\mathsf{eO}}$ then
30:       $blocking := blocking \wedge \bigvee_{\mathbf{a} \in \mathsf{I\text{-}ARGS}(prf)} O_{\mathbf{a}}$
31:     end if
32:     if $\pi^{\mathsf{P}}_{\mathsf{eI}}$ then
33:       $blocking := blocking \wedge \bigvee_{\mathbf{a} \in \mathsf{O\text{-}ARGS}(prf)} I_{\mathbf{a}}$
34:     end if
35:   end if
36: while $prf \neq \varnothing$
37: if $L_p = \varnothing$ then
38:   return $\bot$
39: end if
40: return $\top$
```

Algorithm 16 Skeptical acceptance under Preferred Semantics using SAT solvers 2

```
    SAT-DS-PR
    Input: $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, $\mathbf{x} \in \mathcal{A}$, $C* \in \mathfrak{C}$, $\langle \pi^{\mathsf{P}}_{\mathsf{iO}}, \pi^{\mathsf{P}}_{\mathsf{iI}}, \pi^{\mathsf{P}}_{\mathsf{eO}}, \pi^{\mathsf{P}}_{\mathsf{eI}} \rangle \in \{\top, \bot\}^4$
    Output: $\{\top, \bot\}$
 1: if $\mathsf{SatS}(\mathsf{cnf}(C*, \Gamma) \wedge I_{\mathbf{x}}) = \varnothing \vee \mathsf{SatS}(\mathsf{cnf}(C*, \Gamma) \wedge O_{\mathbf{x}}) \neq \varnothing$ then
 2:   return $\bot$
 3: end if
 4: $blocking := \top$
 5: do
 6:   $iblock := \top$, $prf := \varnothing$
 7:   do
 8:     $cmp := \mathsf{SatS}(\mathsf{cnf}(C*, \Gamma) \wedge blocking \wedge iblock \wedge U_{\mathbf{x}})$
 9:     if $cmp \neq \varnothing$ then
10:       $prf := cmp$, $iblock := \bigwedge_{\mathbf{a} \in \mathsf{I\text{-}ARGS}(cmp)} I_{\mathbf{a}} \wedge \bigwedge_{\mathbf{a} \in \mathsf{O\text{-}ARGS}(cmp)} O_{\mathbf{a}}$
11:       if $\pi^{\mathsf{P}}_{\mathsf{iO}}$ then
12:         $iblock := iblock \wedge \bigvee_{\mathbf{a} \in \mathsf{U\text{-}ARGS}(cmp)} O_{\mathbf{a}}$
13:       end if
14:       if $\pi^{\mathsf{P}}_{\mathsf{iI}}$ then
15:         $iblock := iblock \wedge \bigvee_{\mathbf{a} \in \mathsf{U\text{-}ARGS}(cmp)} I_{\mathbf{a}}$
16:       end if
17:     end if
18:   while $cmp \neq \varnothing$
19:   if $prf \neq \varnothing$ then
20:     if $\mathsf{SatS}\left(\mathsf{cnf}(C*, \Gamma) \wedge blocking \wedge \bigwedge_{\mathbf{a} \in \mathsf{I\text{-}ARGS}(cmp)} I_{\mathbf{a}} \wedge \bigwedge_{\mathbf{a} \in \mathsf{O\text{-}ARGS}(cmp)} O_{\mathbf{a}} \wedge I_{\mathbf{x}}\right) = \varnothing$ then
21:       return $\bot$
22:     end if
23:     if $\pi^{\mathsf{P}}_{\mathsf{eO}}$ then
24:       $blocking := blocking \wedge \bigvee_{\mathbf{a} \in \mathsf{I\text{-}ARGS}(prf)} O_{\mathbf{a}}$
25:     end if
26:     if $\pi^{\mathsf{P}}_{\mathsf{eI}}$ then
27:       $blocking := blocking \wedge \bigvee_{\mathbf{a} \in \mathsf{O\text{-}ARGS}(prf)} I_{\mathbf{a}}$
28:     end if
29:   end if
30: while $prf \neq \varnothing$
31: return $\top$
```

6. Experimental Evaluation

6.1. Experimental Settings

We ground our experimental analysis with the ICCMA 2015 [9] [27] and ICCMA 2017 benchmarks.[10] Experiments have been run on a cluster with computing nodes equipped with 2.5 Ghz Intel Core 2 Quad Processors, 4 GB of RAM and Linux operating system. A cutoff of 900 seconds was imposed to compute the labellings—either preferred or stable—for each AF as well as for addressing each decision problem. Decision problems have been run three times on randomly selected arguments and results averaged, following the methodology designed for ICCMA 2015 [27]. For each ArgSemSAT-configuration we recorded the overall result: success (if it solved the considered problem), crashed, timed-out or ran out of memory.

To analyse the results we considered two metrics: IPC (the higher the better) and PAR10 (the lower the better). IPC is the International Planning Competition score, as defined for the Agile track of the 2014 edition of the competition [34, 35], and represents a relative performance measure among a set of compared solvers: for a solver $\mathcal{C}$ and a problem p, $Score(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$ otherwise (where $T_p(\mathcal{C})$ is the amount of time required by $\mathcal{C}$ to solve the problem p, and T_p^* is the minimum amount of time required by any compared system). The IPC score on a set of instances is given by the sum of the scores achieved on each considered instance. PAR10 is the Penalized Average Runtime with a penalty equal to ten times the cutoff time. It represents an absolute performance measure which trades off coverage—i.e. the percentage of the AFs that are correctly solved below the cutoff time—and runtime for successfully analysed AFs: runs that do not solve the given problem get ten times the cutoff time, other runs get the actual runtime. The PAR10 score of a solver on a set of AFs is the average of the associated runtimes.

It should be noted that IPC and PAR10 are independent measures, thus considering both of them yields a detailed picture of solver performance. For instance, a configuration of ArgSemSAT may have a good IPC but a worse PAR10, allowing one to infer that the coverage is low but the solver is fast in solving those instances it is able to solve (the opposite holds in the reverse case).

Algorithms have been implemented using two different SAT solvers, Minisat [36] as a library, and Glucose [37] invoked through PIPE communication.[11]

In the following we present the results of the experimental analysis, describing in particular four findings that have been achieved therefrom.

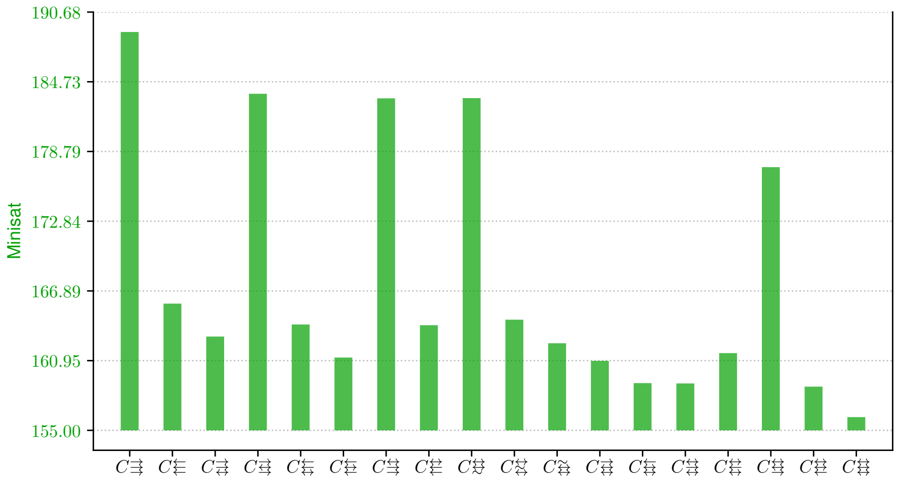

Figure 4: IPC scores of enumerating stable labellings using Minisat varying the chosen encoding on the ICCMA 2015 benchmarks.

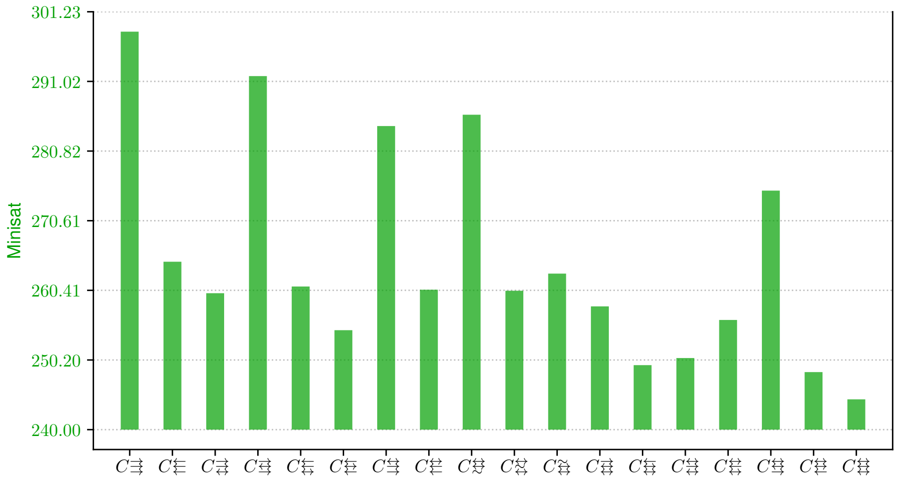

Figure 5: IPC scores of enumerating stable labellings using Minisat varying the chosen encoding on the ICCMA 2017 benchmarks.

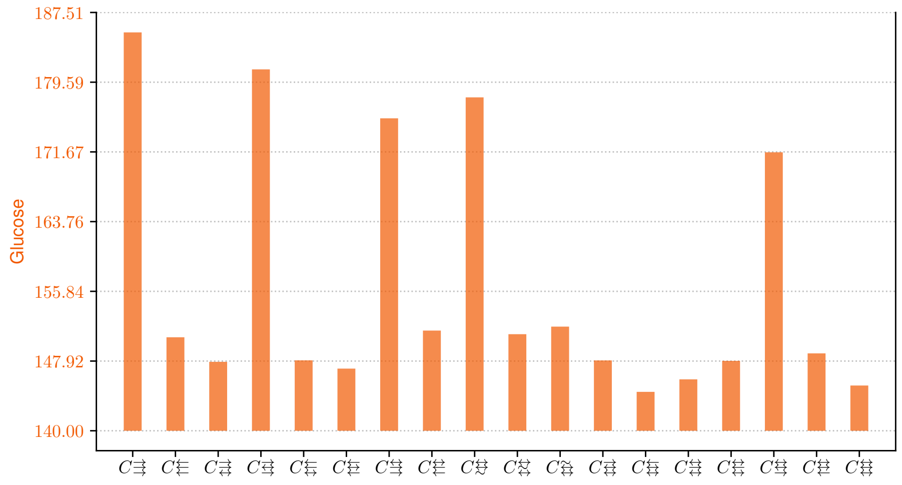

Figure 6: IPC scores of enumerating stable labellings using Glucose varying the chosen encoding on the ICCMA 2015 benchmarks.

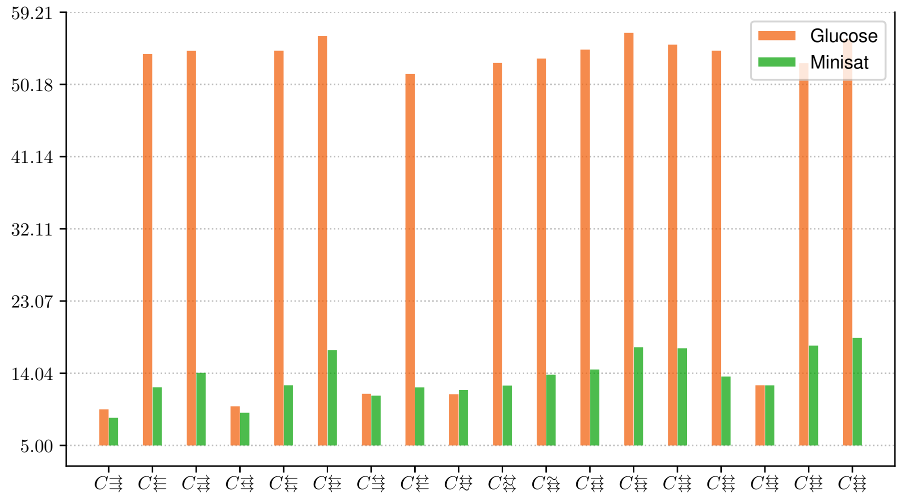

Figure 7: PAR10 scores of enumerating stable labellings using both Minisat and Glucose varying the chosen encoding on the ICCMA 2015 benchmarks.

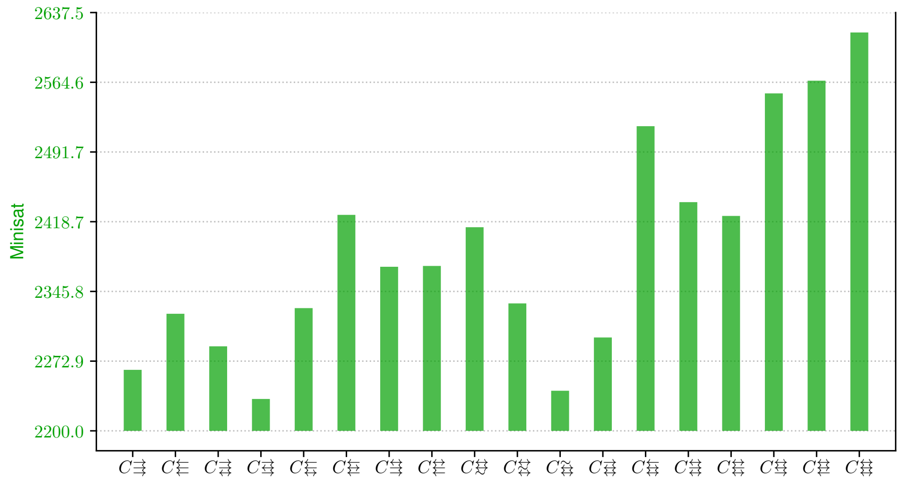

Figure 8: PAR10 scores of enumerating stable labellings using Minisat varying the chosen encoding on the ICCMA 2017 benchmarks.

6.2. *Experimental Results*

6.2.1. $C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$ *is the most efficient encoding of complete labellings*

In order to determine the most efficient encoding of complete labellings in the context of the problems tackled by ArgSemSAT, we computed the enumeration of stable and preferred labellings varying the chosen encoding among the 18 correct ones identified in Section 4.1 (see Figure 2). Moreover, we also considered credulous and skeptical acceptance w.r.t. preferred semantics, once again varying the chosen encoding. Results for credulous and skeptical acceptance w.r.t. stable semantics are omitted as they do not provide further insights.

Algorithms for stable and preferred semantics have been configured respectively with options: $\pi^{\mathsf{S}}_{\mathsf{I}} \wedge \pi^{\mathsf{S}}_{\mathsf{O}}$; and $\mathsf{P}^{\mathsf{IO}}_{\mathsf{IO}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$ (cf. Table B.5). We sampled 50 data points considering other possible configurations: the results we gathered do not provide evidence that different configurations could lead to rejecting our experimental hypothesis.

Figures 4 and 5 depict the IPC scores obtained by each encoding in enumerating stable labellings using Minisat. The same dynamics can be appreciated in Figure 6, where Glucose is used instead (on ICCMA 2015 benchmarks only). We chose not to depict the two graphs on the same figure because IPC is a relative measure—i.e. the value of configuration depends on the pool of configurations considered—therefore the resulting figure would have been misleading, as the

[9] `http://argumentationcompetition.org/2015/ICCMA2015_benchmarks.zip` (on 17th July 2017).

[10] `http://argumentationcompetition.org/2017/results.html` (on 6th February 2019).

[11] We refer any reader interested in knowing the effect of different SAT solvers configuration parameters when addressing argumentation problems to [13].

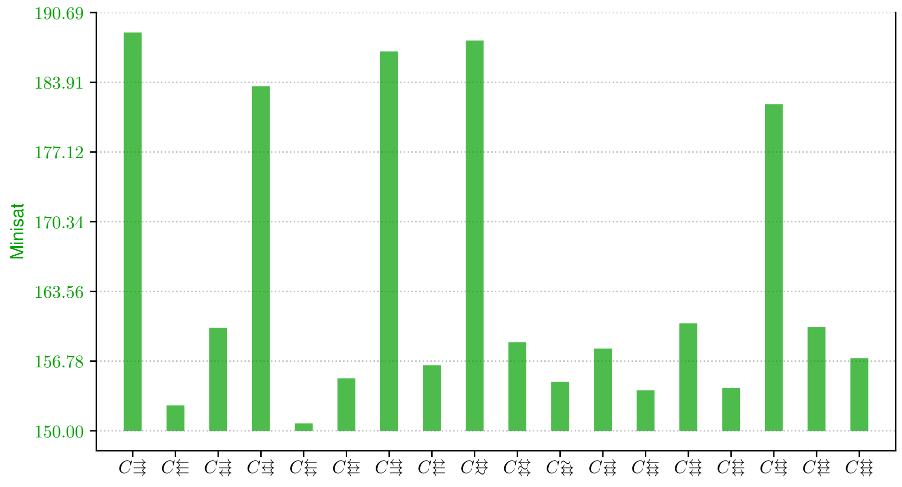

Figure 9: IPC scores of enumerating preferred labellings using Minisat varying the chosen encoding on the ICCMA 2015 benchmarks.

numeric values of the results gathered with a SAT solver are not directly comparable with the results gathered with the second SAT solver. Figures 7 and 8 depict the PAR10 scores obtained in the enumeration of stable labellings. As PAR10 is not a relative measure, i.e. the PAR10 score of a configuration does not depend on the pool of configurations considered, the results can be safely displayed on the same figure without risk of confusion. It is worth noticing that $C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$ has slightly higher PAR10 values than $C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\leftrightarrow}_{\texttt{undec}}$ and $C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\leftrightarrow}_{\texttt{undec}}$ when considering the ICCMA 2017 benchmarks (Figure 8), while still being the one with highest IPC score (Figure 5). This seems to suggest that for a few instances—the actual difference between the PAR10 values is very small—the algorithm running with $C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\leftrightarrow}_{\texttt{undec}}$ and $C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\leftrightarrow}_{\texttt{undec}}$ managed to find the solution in the assigned time, while the algorithm running with $C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$ failed to do so hence incurring in the penalty.

Results obtained by enumerating preferred labellings are similarly shown in Figures 9 and 10, which depict the IPC scores obtained by each encoding using Minisat, Figure 11, which evidences a substantially similar dynamics using Glucose (on ICCMA 2015 benchmarks only), and Figures 12 and 13, which directly compare w.r.t. PAR10 score the implementations.

Overall, five encodings emerge as *optimal* according to IPC score (Figures 4, 5, 6, 9, 10 and 11), namely $C^{\substack{\rightarrow\\\rightarrow\\\rightarrow}} \equiv C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$, $C^{\substack{\rightarrow\\\leftrightarrow\\\rightarrow}} \equiv C^{\rightarrow}_{\texttt{in}} \wedge C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$, $C^{\substack{\leftrightarrow\\\rightarrow\\\rightarrow}} \equiv C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$, $C^{\substack{\leftrightarrow\\\leftrightarrow}} \equiv C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\leftrightarrow}_{\texttt{out}}$ and $C^{\substack{\leftrightarrow\\\leftrightarrow\\\rightarrow}} \equiv C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\leftrightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$. The same observation substantially holds for PAR10 in the case of preferred labellings (Figures 12 and 13) and in the case of stable labellings using Glucose (Figure 7), while the resulting scores are more flat with Minisat, which is used as a library. It is worth noting that all these five configurations have a single commonality, i.e. they lack $C^{\leftarrow}_{\texttt{undec}}$, thus with a small abuse of notation we

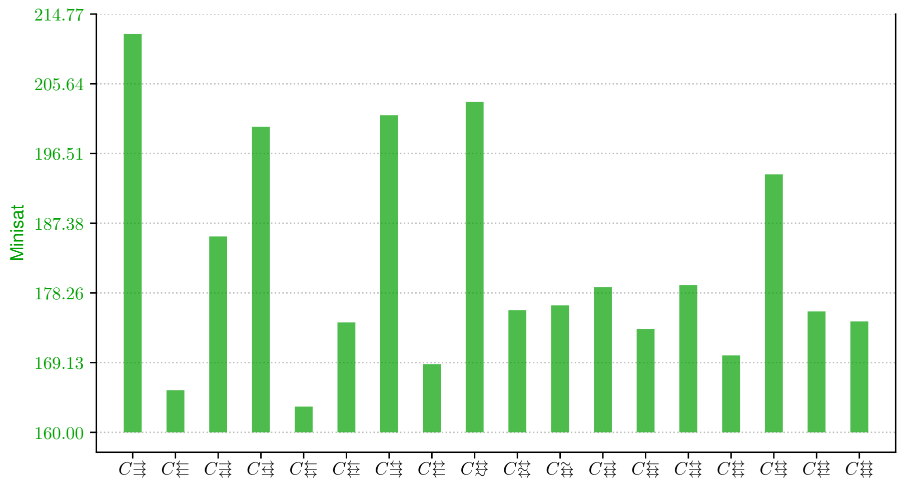

Figure 10: IPC scores of enumerating preferred labellings using Minisat varying the chosen encoding on the ICCMA 2017 benchmarks.

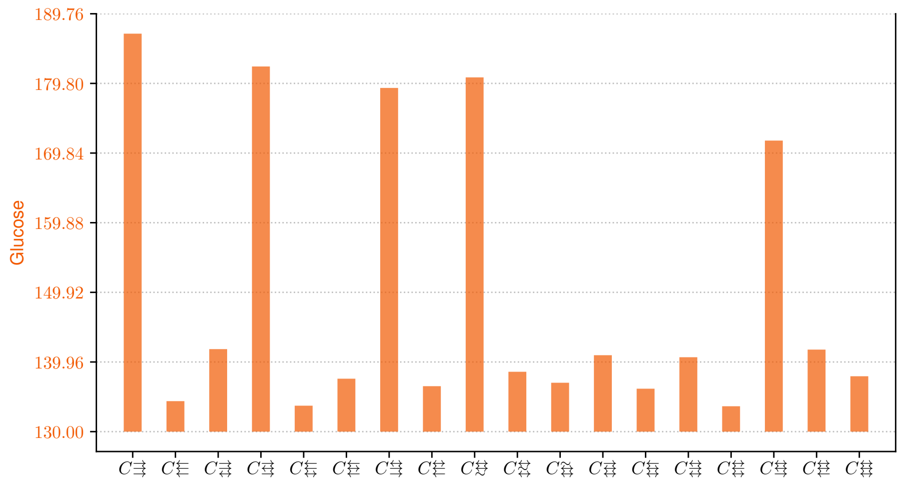

Figure 11: IPC scores of enumerating preferred labellings using Glucose varying the chosen encoding on the ICCMA 2015 benchmarks.

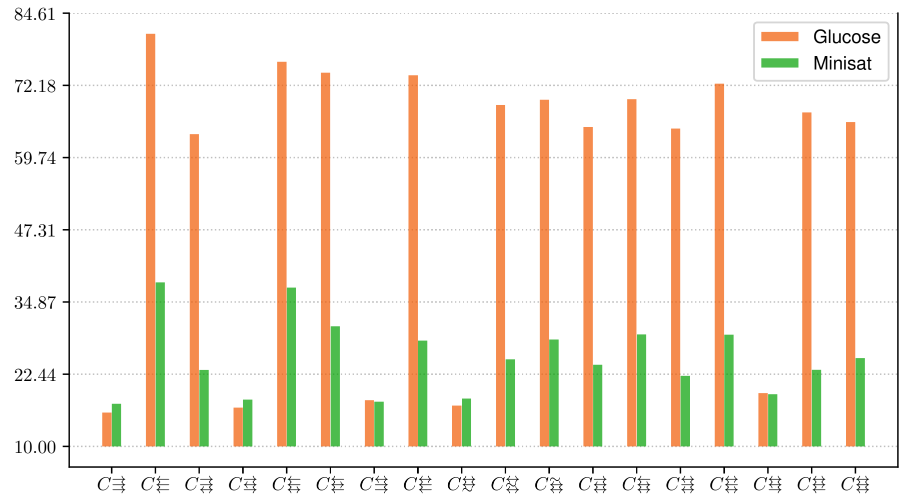

Figure 12: PAR10 scores of enumerating preferred labellings using both Minisat and Glucose varying the chosen encoding on the ICCMA 2015 benchmarks.

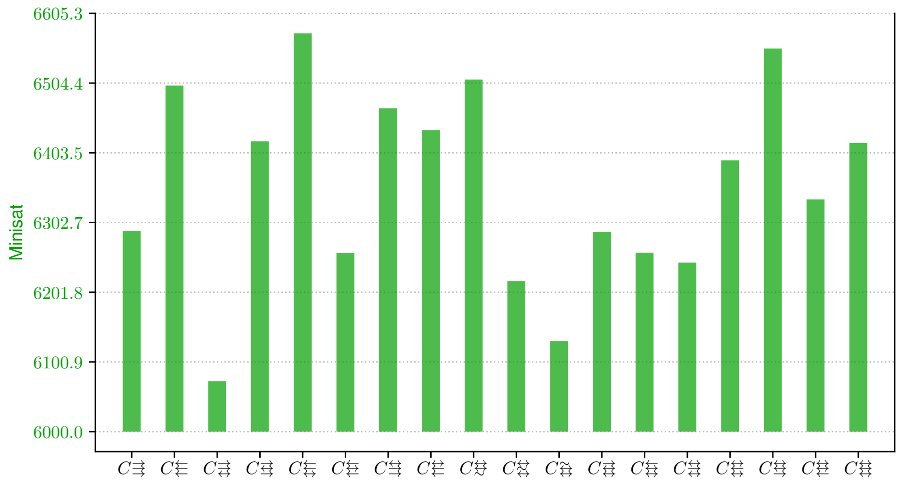

Figure 13: PAR10 scores of enumerating preferred labellings using Minisat varying the chosen encoding on the ICCMA 2017 benchmarks.

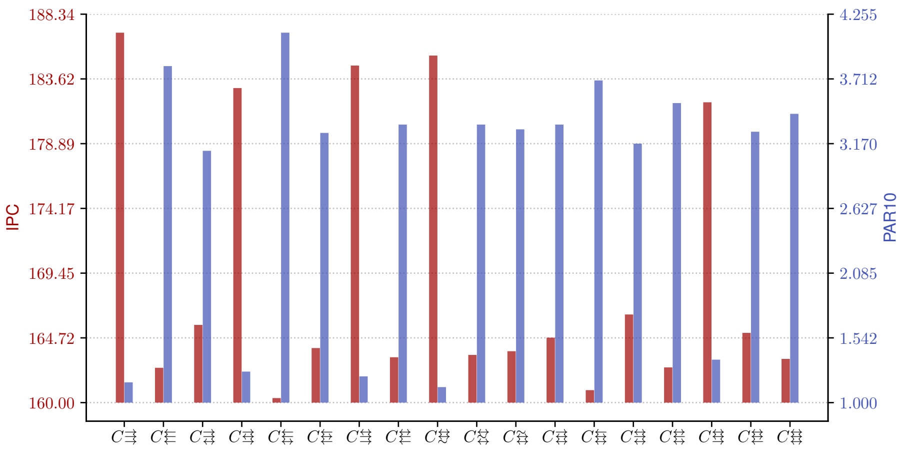

Figure 14: IPC (left axis, red) and PAR10 (right axis, blue) scores of solving credulous acceptance w.r.t. preferred semantics using Minisat only and varying the chosen encoding, cf. Figure 2, on the ICCMA 2015 benchmarks.

denote $\{C\substack{\rightarrow\\\rightarrow\\\rightarrow}, C\substack{\rightarrow\\\leftrightarrow\\\rightarrow}, C\substack{\leftrightarrow\\\rightarrow\\\rightarrow}, C\substack{\leftrightarrow\\\leftrightarrow\\\sim}, C\substack{\leftrightarrow\\\leftrightarrow\\\rightarrow}\}$ as $\neg C^{\leftarrow}_{\texttt{undec}}$. The only exception can be noticed in Figure 13, where it can be seen that $C\substack{\rightarrow\\\rightarrow\\\leftrightarrow}$ allows to solve more cases than $C\substack{\rightarrow\\\rightarrow\\\rightarrow}$ before the chosen cut-off time. From an investigation in the results, it appears that most of this difference depends on the *SemBuster* benchmark set,[12] where $C\substack{\rightarrow\\\rightarrow\\\rightarrow}$ fails to provide an answer in 4 cases, and in general is slower than $C\substack{\rightarrow\\\rightarrow\\\leftrightarrow}$. This is hardly a surprise since those benchmarks were specifically designed to have a large number of preferred extensions, but only one semi-stable extension.[13] Therefore for these benchmarks providing additional constraints on the undecided arguments proves to be substantially advantageous.

Among $\neg C^{\leftarrow}_{\texttt{undec}}$, the global optimum is always in $C\substack{\rightarrow\\\rightarrow\\\rightarrow} \equiv C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$, while the worst encoding is $C\substack{\leftrightarrow\\\leftrightarrow\\\rightarrow}$. As to the other three encondings, the order among them varies depending on the semantics, the SAT solver used and the performance metric (i.e. IPC or PAR10). On the other hand, with the exception of Figure 9, $C\substack{\rightarrow\\\leftrightarrow\\\rightarrow}$ is always the second best encoding in $\neg C^{\leftarrow}_{\texttt{undec}}$, and $C\substack{\leftrightarrow\\\leftrightarrow\\\sim}$ tends to outperform $C\substack{\leftrightarrow\\\rightarrow\\\rightarrow}$ (see in particular Figures 4, 6, 9 and 11), even if the differences between $C\substack{\leftrightarrow\\\rightarrow\\\rightarrow}$ and $C\substack{\leftrightarrow\\\leftrightarrow\\\sim}$ are often small, and, depending on the performance metric, Glucose and Minisat might lead to different ordering w.r.t. them.

Finally, it is worth mentioning the significant difference of performance between Minisat and Glucose in enumerating stable labellings in terms of PAR10 outside $\neg C^{\leftarrow}_{\texttt{undec}}$ (see Figure 7). The same great loss of performance is shown in Figure 12 in the case of preferred labellings.

Since we believe that the evidence gathered so far is strong enough to suggest that configurations in $\neg C^{\leftarrow}_{\texttt{undec}}$ will be optima irrespectively of whether

[12]`http://argumentationcompetition.org/2017/SemBuster.pdf` (on 08 February 2019)
[13]Semi-stable semantics attempts to minimise the number of undecided arguments.

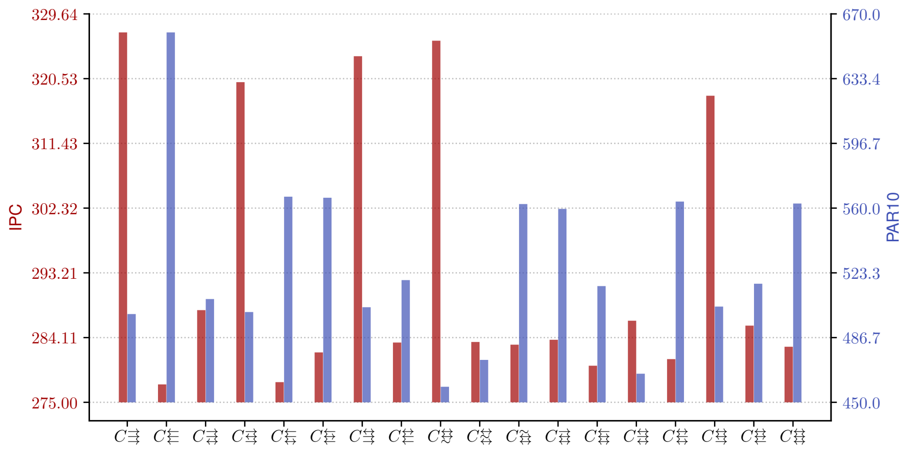

Figure 15: IPC (left axis, red) and PAR10 (right axis, blue) scores of solving credulous acceptance w.r.t. preferred semantics using Minisat only and varying the chosen encoding, cf. Figure 2 , on the ICCMA 2017 benchmarks

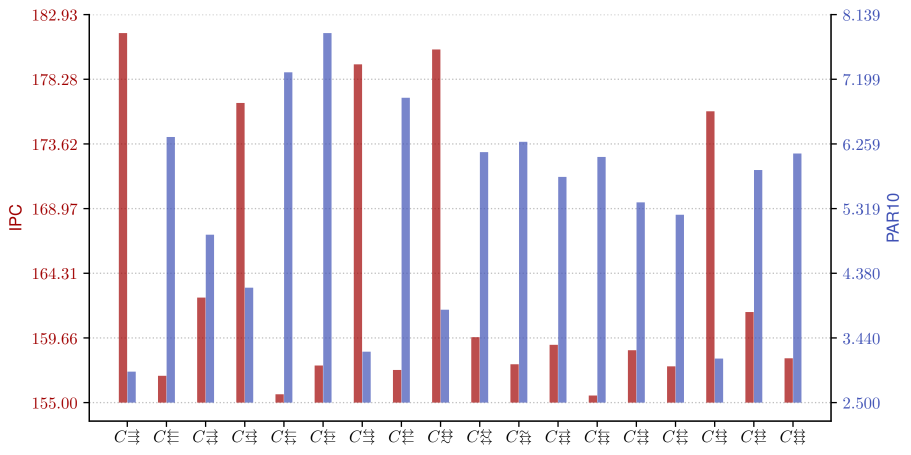

Figure 16: IPC (left axis, red) and PAR10 (right axis, blue) scores of solving skeptical acceptance w.r.t. preferred semantics using Minisat only and varying the chosen encoding, cf. Figure 2, on the ICCMA 2015 benchmarks.

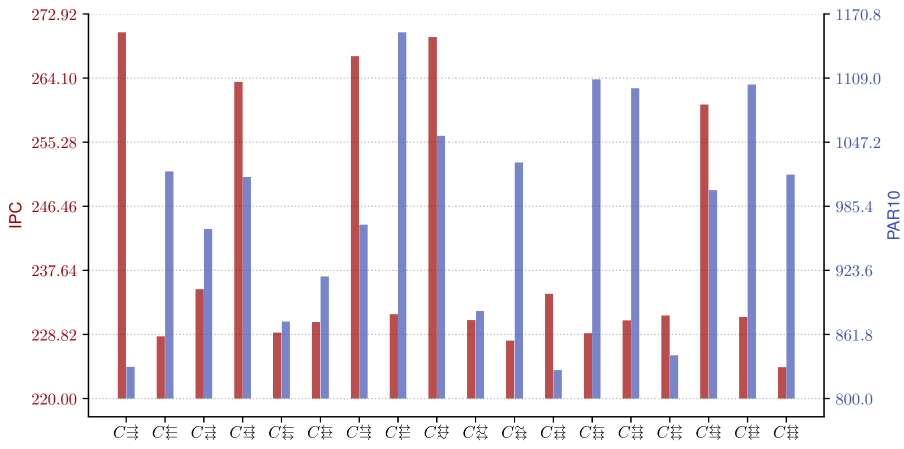

Figure 17: IPC (left axis, red) and PAR10 (right axis, blue) scores of solving skeptical acceptance w.r.t. preferred semantics using Minisat only and varying the chosen encoding, cf. Figure 2 , on the ICCMA 2017 benchmarks

we consider Minisat or Glucose, in the interest of time and space we limit our following results to Minisat only.

Figures 14, 15, 16, and 17 show the results of the experimental analysis when considering, respectively, credulous and skeptical acceptance w.r.t. preferred semantics using Minisat. Results are substantially in line with those highlighted for the enumeration of preferred labellings.

Summing up, this large experimentation counting 97,560 executions[14] of our algorithms shows that $C⇶ \equiv C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$ is consistently the encoding leading to the best performance.

We believe that the above results can be explained by looking at the CNF formulæ of the encodings of complete labellings presented in Proposition 11.

Indeed, it is known [38] that the 2-SAT problem—i.e. the SAT problem where all the formulæ are binary—is polynomial. Moreover in [39] the authors show how reducing the length of disjunctive formulæ leads to significant computational benefits. Modern SAT solvers rely on conflict-driven clause learning [40], which is an improvement built around the DPLL unit-propagation algorithm—a typical conflict-driven clause learning based SAT solver spends at least 80% of the time running the unit propagation procedure [41]—and thus it will also benefit from short—best with 2-SAT–formulæ.

According to Proposition 11, $C^{\leftarrow}_{\texttt{undec}}$ maximises the number of disjunctions per clause. This seems to explain why the best configurations belong to $\neg C^{\leftarrow}_{\texttt{undec}}$.

[14] 192 AFs in the ICCMA 2015 benchmark, 350 in the ICCMA 2017 benchmark, each of those analysed in 36 configurations both for enumeration of stable and of preferred labellings, on top of 3 runs for credulous and skeptical acceptance w.r.t. preferred semantics against 18 different configurations.

Moreover, there are only two non-redundant encodings in $\neg C^{\leftarrow}_{\texttt{undec}}$, namely $C\substack{\rightarrow\\\rightarrow\\\rightarrow}$ and $C\substack{\leftrightarrow\\\leftrightarrow\\\sim}$, and both of them minimise the number of clauses. The fact that $C\substack{\rightarrow\\\rightarrow\\\rightarrow}$ achieves a better performance can be explained in the case of the enumeration of stable labellings by taking into account that, for each argument $\mathbf{a}$, the condition $\neg U_{\mathbf{a}}$ is enforced, thus the constraint $C^{\rightarrow}_{\texttt{undec}}$ does not play any role. As a consequence, differently from $C\substack{\rightarrow\\\rightarrow\\\rightarrow}$, $C\substack{\leftrightarrow\\\leftrightarrow\\\sim}$ becomes redundant, and the presence of redundant clauses is known to have a negative impact on the performance of SAT solvers [42]. This may also explain the relatively good performance achieved by $C\substack{\rightarrow\\\leftrightarrow\\\rightarrow}$. We conjecture that similar reasons underlie the performance results in the case of preferred semantics, though the performance difference is less noticeable.

As to the comparison between Glucose and Minisat, the first seems to be more sensible to the presence of $C^{\leftarrow}_{\texttt{undec}}$, and its decrease of performance is possibly particularly accentuated by the fact that it has been not implemented as an internal library but called as an external process using PIPE communications. On the other hand, from Figure 12 Glucose emerges as the *winner* against Minisat according to PAR10 for the configurations in $\neg C^{\leftarrow}_{\texttt{undec}}$. It must also be acknowledged that the differences between the two are quite minimal so that they do not justify further investigations: Glucose is the result of almost 10 years of improvements from the SAT community, and enumerating preferred labellings is significantly more complex than enumerating stable labellings. It seems reasonable that a better solver would have the opportunity to demonstrate its value.

In this section we have experimentally shown that $C\substack{\rightarrow\\\rightarrow\\\rightarrow} \equiv C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$ overall is the encoding leading to the best performance for all the considered algorithms and different configurations of the algorithm parameters. In the subsequent subsections we discuss how such parameters exhibit positive interactions that can be exploited to gain a better performance.

6.2.2. Using an AllSAT solver leads to a significant improvement for enumerating stable labellings

We fixed the encoding to be the most efficient, i.e. $C\substack{\rightarrow\\\rightarrow\\\rightarrow} \equiv C^{\rightarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\rightarrow}_{\texttt{undec}}$, and we enumerated stable labellings on the benchmark AFs varying the configuration of the algorithm.

Tables 2 and 3 summarise the results in terms of IPC score, PAR10, and coverage. Only the parameters set to $\top$ are indicated, while false parameters are omitted for brevity. It is immediate to see that the use of AllSAT leads to a significant improvement for enumerating stable labellings. It is also worth noticing that varying the other parameters, i.e. $\pi^{\mathsf{S}}_{\mathsf{O}}$ and $\pi^{\mathsf{S}}_{\mathsf{I}}$, leads to statistically significant results with $p < 0.05$ but not with $p < 0.01$ (Friedman test, $\chi^2(2) = 6.776, p = 0.034$) on the ICCMA 2015 benchmarks. In the case of the ICCMA 2017, it is self-evident from the magnitude of values that the results are also stastistically significant. Varying parameters $\pi^{\mathsf{S}}_{\mathsf{All;I}}$ and $\pi^{\mathsf{S}}_{\mathsf{All;O}}$ do not lead to statistically significant variations on the ICCMA 2015 benchmarks, hence in Tables 2 and 3 we considered only the combination $\pi^{\mathsf{S}}_{\mathsf{All}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;I}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$.

Parameters	IPC Score	PAR10	Coverage
$\pi^{\mathsf{S}}_{\mathsf{O}}$	177.57	8.20	100.0
$\pi^{\mathsf{S}}_{\mathsf{I}}$	177.26	8.24	100.0
$\pi^{\mathsf{S}}_{\mathsf{O}} \wedge \pi^{\mathsf{S}}_{\mathsf{I}}$	177.15	8.49	100.0
$\pi^{\mathsf{S}}_{\mathsf{All}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;I}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$	**191.87**	**5.40**	100.0

Table 2: IPC score, PAR10 and coverage (percentage of AFs successfully analysed) when considering different algorithmic parameters for enumerating stable labellings on the ICCMA 2014 benchmarks. In bold the best results.

Parameters	IPC Score	PAR10	Coverage
$\pi^{\mathsf{S}}_{\mathsf{O}}$	214.45	4460.49	73.43
$\pi^{\mathsf{S}}_{\mathsf{I}}$	215.02	4503.22	73.14
$\pi^{\mathsf{S}}_{\mathsf{O}} \wedge \pi^{\mathsf{S}}_{\mathsf{I}}$	210.31	4834.84	71.14
$\pi^{\mathsf{S}}_{\mathsf{All}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;I}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$	**303.58**	**2263.87**	**86.86**

Table 3: IPC score, PAR10 and coverage (percentage of AFs successfully analysed) when considering different algorithmic parameters for enumerating stable labellings on the ICCMA 2017 benchmarks. In bold the best results.

The improvement yielded by the AllSAT solver is hardly a surprise, given that our algorithm for enumerating stable labellings is less engineered than the AllSAT algorithm that uses only Minisat data structures.

6.2.3. Enumerating stable labellings first can lead to a significant improvement for enumerating preferred labellings

Also in this case we fixed the encoding to be the most efficient one, i.e. $C\rightrightarrows \equiv C_{\texttt{in}}^{\rightarrow} \wedge C_{\texttt{out}}^{\rightarrow} \wedge C_{\texttt{undec}}^{\rightarrow}$, and we enumerated the preferred labellings on the benchmark AFs varying the configuration of the algorithm.

Figures 18, 19, 20, and 21 depict the box-and-whisker plot of three groups of results. As above, false parameters are omitted for brevity, unless necessary to avoid confusion.

Group (A) considers performance results collected when running the algorithm with the option $\pi^{\mathsf{P}}_{\mathsf{S}} = \bot$, i.e. without searching for stable labellings first, with combinations of options $\pi^{\mathsf{P}}_{\mathsf{iI}}$, $\pi^{\mathsf{P}}_{\mathsf{iO}}$, $\pi^{\mathsf{P}}_{\mathsf{eI}}$, and $\pi^{\mathsf{P}}_{\mathsf{eO}}$.

Group (B) considers the case where the algorithm is run with $\pi^{\mathsf{P}}_{\mathsf{S}} = \top$ and $\pi^{\mathsf{S}}_{\mathsf{All}} = \bot$, i.e. stable labellings are enumerated first, but without using an AllSAT solver.

Finally, group (C) considers the usual combinations of options $\pi^{\mathsf{P}}_{\mathsf{iI}}$, $\pi^{\mathsf{P}}_{\mathsf{iO}}$, $\pi^{\mathsf{P}}_{\mathsf{eI}}$, $\pi^{\mathsf{P}}_{\mathsf{eO}}$ with $\pi^{\mathsf{P}}_{\mathsf{S}} = \pi^{\mathsf{S}}_{\mathsf{All}} = \top$.

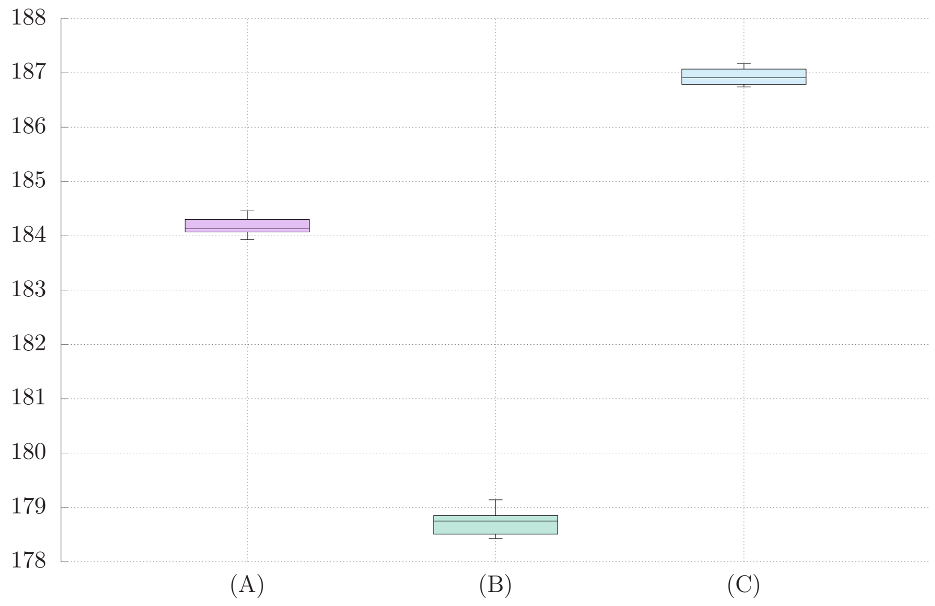

Figure 18: IPC scores (higher better) when enumerating preferred labellings on ICCMA 2015 benchmarks varying algorithm configurations: (A) computing without searching for stable labellings first; (B) searching for stable labellings first ($\pi^{\mathsf{P}}_{\mathsf{S}}$) but without using an AllSAT solver; (C) searching for stable labellings first ($\pi^{\mathsf{P}}_{\mathsf{S}}$) using an AllSAT solver ($\pi^{\mathsf{S}}_{\mathsf{All}}$, with $\pi^{\mathsf{S}}_{\mathsf{All;I}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$).

Overall, our empirical evaluation supports the claim that searching for stable labellings first leads to a significant improvement for enumerating preferred labellings, but only when an AllSAT solver is exploited. If this is not the case, Figures 18, 19, 20, and 21 suggest that starting from stable labellings improves the coverage of the algorithm, while slowing down the overall search. We believe this is due to the additional time required for proving that the search for stable labellings is completed, as well as to the cost of parameters passing, particularly in cases where the stable labellings are a few percentage of the preferred labellings. These disadvantages appear as completely compensated by the use of the engineered AllSAT algorithm.

Post-hoc analysis: on the positive interactions between options when enumerating stable labellings first. We also considered the interactions between choosing different options for enumerating stable labellings at the beginning using an AllSAT solver, together with the acceptable combination of parameters for computing blocking clauses when exploring the space of complete labellings, i.e. parameters $\pi^{\mathsf{P}}_{\mathsf{iI}}$, $\pi^{\mathsf{P}}_{\mathsf{iO}}$, $\pi^{\mathsf{P}}_{\mathsf{eI}}$, and $\pi^{\mathsf{P}}_{\mathsf{eO}}$. Figures 22 and 23 depict the results of this analysis. The best performance is obtained by combining $\pi^{\mathsf{P}}_{\mathsf{S}} \wedge \pi^{\mathsf{S}}_{\mathsf{All}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$ with $\mathsf{P}^{\mathsf{IO}}_{\mathsf{IO}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$ over the ICCMA 2015 benchmarks, and by combining $\pi^{\mathsf{P}}_{\mathsf{S}} \wedge \pi^{\mathsf{S}}_{\mathsf{All}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$ with $\mathsf{P}^{\mathsf{I}}_{\mathsf{I}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}}$ over the ICCMA 2017 benchmarks. As noticed earlier, the ICCMA 2017 benchmark set is substantially more complex

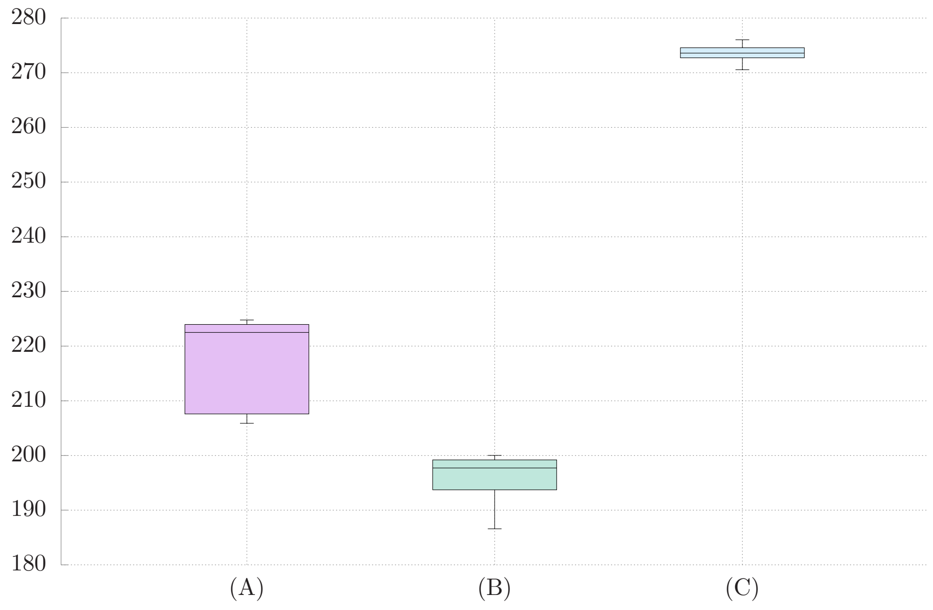

Figure 19: IPC scores (higher better) when enumerating preferred labellings on ICCMA 2017 benchmarks varying algorithm configurations: (A) computing without searching for stable labellings first; (B) searching for stable labellings first ($\pi^{\mathsf{P}}_{\mathsf{S}}$) but without using an AllSAT solver; (C) searching for stable labellings first ($\pi^{\mathsf{P}}_{\mathsf{S}}$) using an AllSAT solver ($\pi^{\mathsf{S}}_{\mathsf{All}}$, with $\pi^{\mathsf{S}}_{\mathsf{All;I}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$).

Version	**IPC Score**	**PAR10**	**Coverage**
Straightforward	186.49	2.22	100.0
Improved	**188.30**	**1.39**	100.0

Table 4: IPC score, PAR10 and coverage (percentage of AFs successfully analysed) when considering straightforward and improved algorithms for computing skeptical acceptability w.r.t. preferred semantics on ICCMA 2015 benchmarks. In bold the best results.

when dealing with preferred semantics, hence this difference suggests that such parameter optimisation is heavily benchmark dependent consistently with the *no free-lunch theorem* [17].

6.2.4. The improved algorithm for skeptical acceptance w.r.t. preferred semantics leads to significant improvements

We fixed the encoding to be the most efficient, i.e. $C_{\rightrightarrows} \equiv C_{\overrightarrow{\mathtt{in}}} \wedge C_{\overrightarrow{\mathtt{out}}} \wedge C_{\overrightarrow{\mathtt{undec}}}$, and we computed skeptical acceptance of arguments w.r.t. preferred semantics on the ICCMA 2015 benchmark AFs varying the configuration of the **straightforward** algorithm, i.e. Algorithm 15, and of the **improved** algorithm, i.e. Algorithm 16. In order to provide a fair comparison where neither algorithm is allowed to exploit an AllSAT solver, we did not consider the option of computing

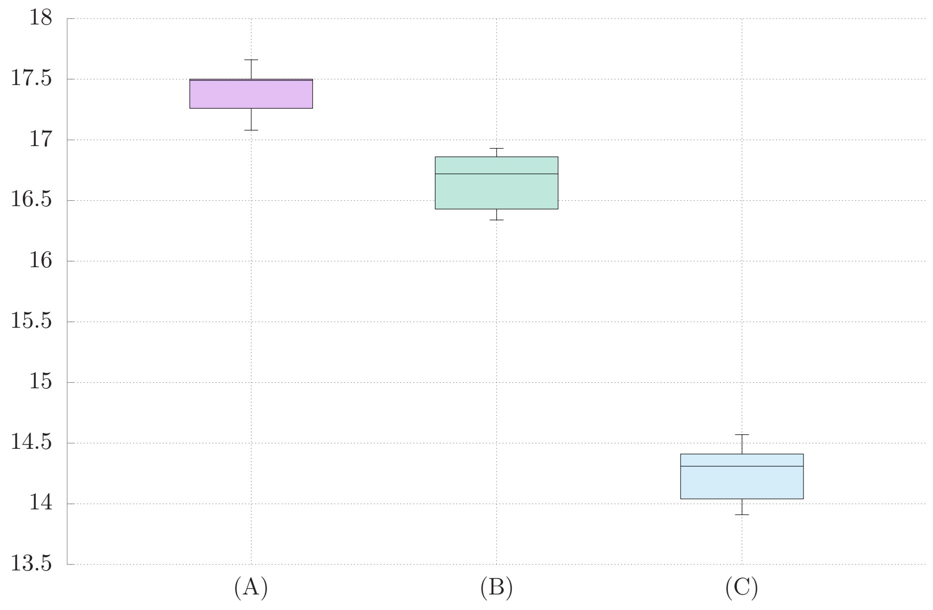

Figure 20: PAR10 scores (lower better) when enumerating preferred labellings on ICCMA 2015 benchmarks varying algorithm configurations: (A) computing without searching for stable labellings first; (B) searching for stable labellings first ($\pi^{\mathsf{P}}_{\mathsf{S}}$) but without using an AllSAT solver; (C) searching for stable labellings first ($\pi^{\mathsf{P}}_{\mathsf{S}}$) using an AllSAT solver ($\pi^{\mathsf{S}}_{\mathsf{All}}$, with $\pi^{\mathsf{S}}_{\mathsf{All;I}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$).

stable extensions first in the straightforward algorithm. As a matter of fact, considering this option would not lead to different conclusions, as discussed in the post-hoc analysis below.

Table 4 compares the best configuration for the straightforward algorithm against the best configuration for the improved algorithm. According to the results we achieved, the best configuration of the straightforward algorithm is with options $\pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}}$ while for the improved version is with $\pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$. The results of Table 4 are statistically significant (Wilcoxon, Z=-2.487, p = 0.013) and show that there is an improvement of performance using the improved algorithm.

This analysis, first performed before submitting ArgSemSAT to the ICCMA 2017, has been confirmed in the results of that competition. Indeed Algorithm 16 resulted to be the winner of the sub-track of ICCMA 2017 for skeptical acceptance of preferred semantics by a large margin.[15]

Post-hoc analysis: on the positive interactions between options when considering skeptical acceptance w.r.t. preferred semantics. We also analysed the interactions between choosing to use different options for considering (or not) stable labellings

[15]`http://argumentationcompetition.org/2017/ICCMA2017-slides-tafa.pdf` (on 6th February 2019), page 26.

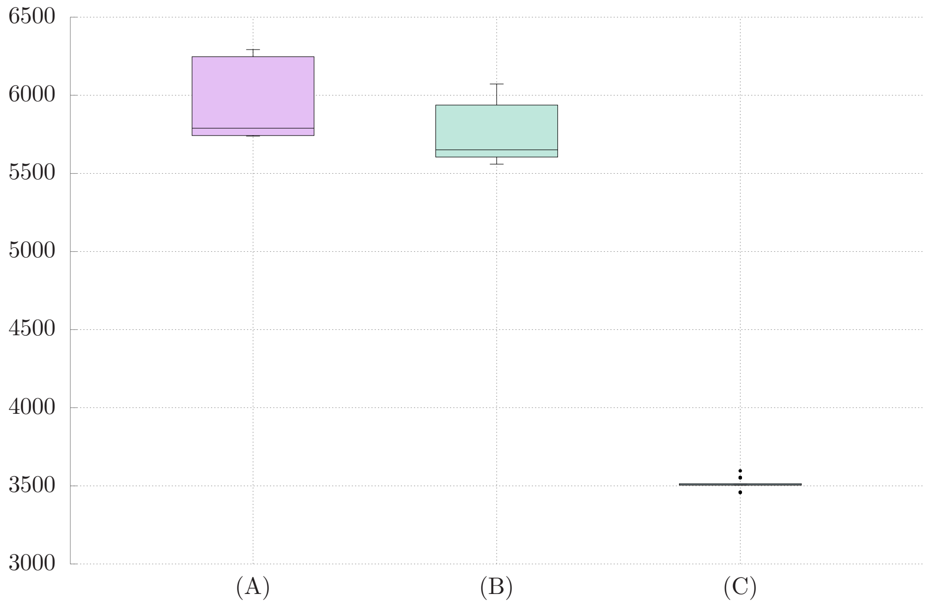

Figure 21: PAR10 scores (lower better) when enumerating preferred labellings on ICCMA 2017 benchmarks varying algorithm configurations: (A) computing without searching for stable labellings first; (B) searching for stable labellings first ($\pi^{\mathsf{P}}_{\mathsf{S}}$) but without using an AllSAT solver; (C) searching for stable labellings first ($\pi^{\mathsf{P}}_{\mathsf{S}}$) using an AllSAT solver ($\pi^{\mathsf{S}}_{\mathsf{All}}$, with $\pi^{\mathsf{S}}_{\mathsf{All;I}} \wedge \pi^{\mathsf{S}}_{\mathsf{All;O}}$).

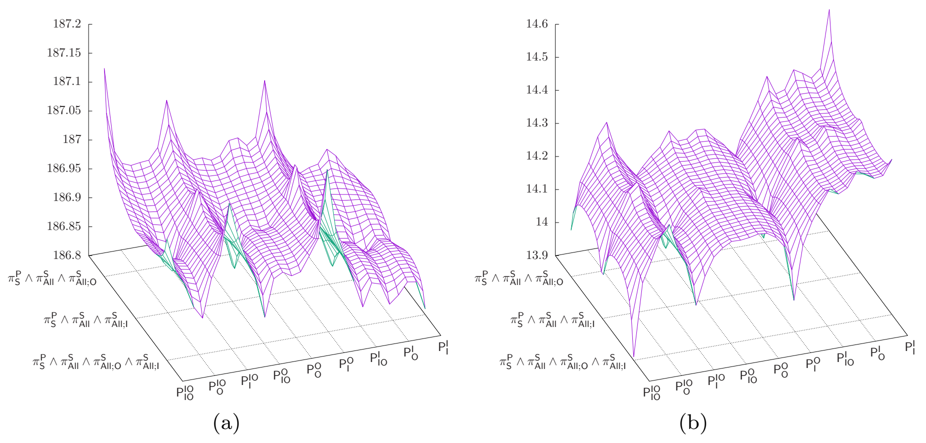

Figure 22: IPC scores (a) and PAR10 (b) when enumerating preferred labellings varying algorithm parameters on ICCMA 2015 benchmarks. Here, $\mathsf{P}^{\mathsf{I}}_{\mathsf{I}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}}$, $\mathsf{P}^{\mathsf{I}}_{\mathsf{O}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{I}}_{\mathsf{IO}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{O}}_{\mathsf{I}} \equiv \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}}$, $\mathsf{P}^{\mathsf{O}}_{\mathsf{O}} \equiv \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{O}}_{\mathsf{IO}} \equiv \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{IO}}_{\mathsf{I}} \equiv \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{IO}}_{\mathsf{O}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{IO}}_{\mathsf{IO}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$.

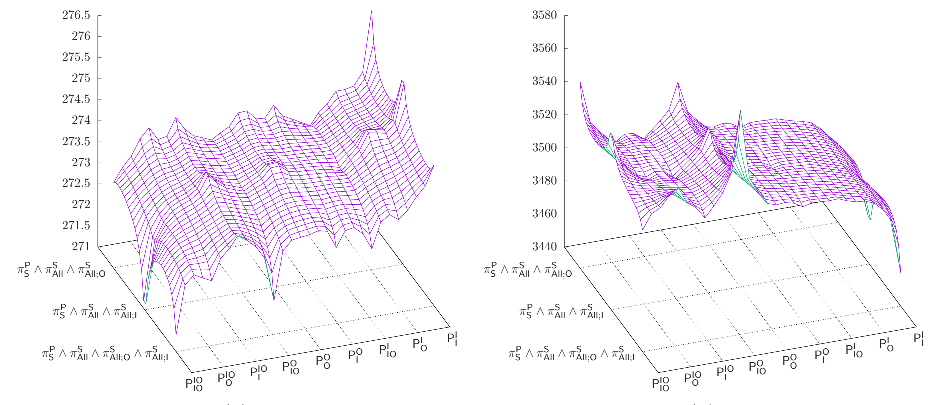

Figure 23: IPC scores (a) and PAR10 (b) when enumerating preferred labellings varying algorithm parameters on ICCMA 2017 benchmarks. Here, $P^I_I \equiv \pi^P_{iI} \wedge \pi^P_{eI}$, $P^I_O \equiv \pi^P_{iI} \wedge \pi^P_{eO}$, $P^I_{IO} \equiv \pi^P_{iI} \wedge \pi^P_{eI} \wedge \pi^P_{eO}$, $P^O_I \equiv \pi^P_{iO} \wedge \pi^P_{eI}$, $P^O_O \equiv \pi^P_{iO} \wedge \pi^P_{eO}$, $P^O_{IO} \equiv \pi^P_{iO} \wedge \pi^P_{eO}$, $P^{IO}_I \equiv \pi^P_{iO} \wedge \pi^P_{eO}$, $P^{IO}_O \equiv \pi^P_{iI} \wedge \pi^P_{iO} \wedge \pi^P_{eO}$, $P^{IO}_{IO} \equiv \pi^P_{iI} \wedge \pi^P_{iO} \wedge \pi^P_{eI} \wedge \pi^P_{eO}$.

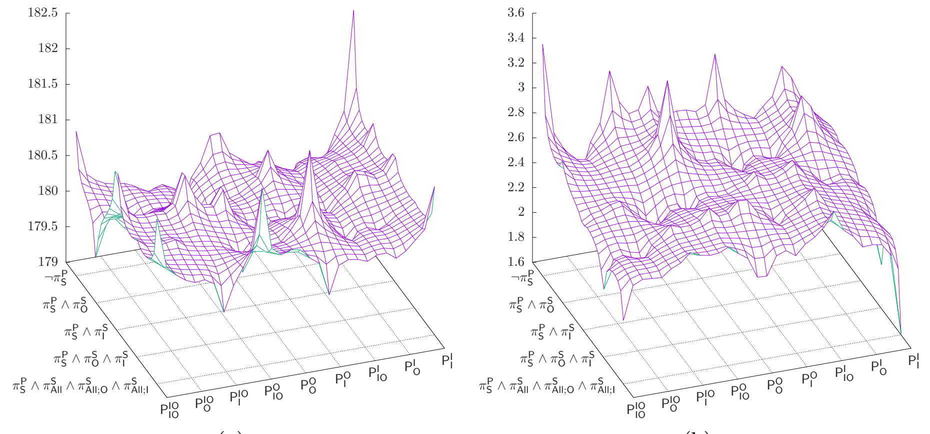

Figure 24: IPC scores (a) and PAR10 (b) when computing skeptical acceptance w.r.t. preferred semantics varying algorithm parameters on ICCMA 2015 benchmarks. Here, $P^I_I \equiv \pi^P_{iI} \wedge \pi^P_{eI}$, $P^I_O \equiv \pi^P_{iI} \wedge \pi^P_{eO}$, $P^I_{IO} \equiv \pi^P_{iI} \wedge \pi^P_{eI} \wedge \pi^P_{eO}$, $P^O_I \equiv \pi^P_{iO} \wedge \pi^P_{eI}$, $P^O_O \equiv \pi^P_{iO} \wedge \pi^P_{eO}$, $P^O_{IO} \equiv \pi^P_{iO} \wedge \pi^P_{eO}$, $P^{IO}_I \equiv \pi^P_{iO} \wedge \pi^P_{eO}$, $P^{IO}_O \equiv \pi^P_{iI} \wedge \pi^P_{iO} \wedge \pi^P_{eO}$, $P^{IO}_{IO} \equiv \pi^P_{iI} \wedge \pi^P_{iO} \wedge \pi^P_{eI} \wedge \pi^P_{eO}$.

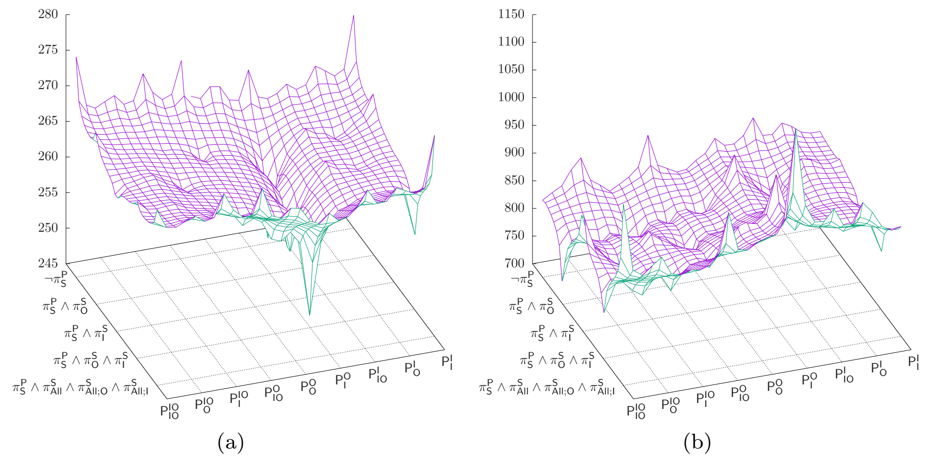

Figure 25: IPC scores (a) and PAR10 (b) when computing skeptical acceptance w.r.t. preferred semantics varying algorithm parameters on ICCMA 2017 benchmarks. Here, $\mathsf{P}^{\mathsf{I}}_{\mathsf{I}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}}$, $\mathsf{P}^{\mathsf{I}}_{\mathsf{O}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{I}}_{\mathsf{IO}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{O}}_{\mathsf{I}} \equiv \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}}$, $\mathsf{P}^{\mathsf{O}}_{\mathsf{O}} \equiv \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{O}}_{\mathsf{IO}} \equiv \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{IO}}_{\mathsf{I}} \equiv \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{IO}}_{\mathsf{O}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$, $\mathsf{P}^{\mathsf{IO}}_{\mathsf{IO}} \equiv \pi^{\mathsf{P}}_{\mathsf{iI}} \wedge \pi^{\mathsf{P}}_{\mathsf{iO}} \wedge \pi^{\mathsf{P}}_{\mathsf{eI}} \wedge \pi^{\mathsf{P}}_{\mathsf{eO}}$.

at the beginning of the skeptical acceptance w.r.t. preferred semantics in the **straightforward** algorithm, together with the acceptable combination of parameters for computing blocking clauses when exploring the space of complete extensions, i.e. parameters $\pi^{\mathsf{P}}_{\mathsf{iI}}$, $\pi^{\mathsf{P}}_{\mathsf{iO}}$, $\pi^{\mathsf{P}}_{\mathsf{eI}}$, and $\pi^{\mathsf{P}}_{\mathsf{eO}}$. Figures 24 and 25 depict the results of this analysis: although the magnitude of the effect in the performance of varying the parameters is minimal—sometimes even negligible, in particular according to the PAR10 measures, cf. Figure 24b—according to the IPC (Figure 24a and 25a) it appears that the best option is not to exploit the inclusion relation between the sets of stable and preferred labellings, as highlighted by the peak in Figure 24a and confirmed in Figure 25a.

7. Related Work

The idea of exploiting satisfiability techniques for solving abstract argumentation problems (even if no specific algorithm is devised) can be traced back to the work in [43], where encodings of stable and complete extensions are introduced in terms of variables associated to arguments indicating whether each argument belongs to the extension, rather than in terms of labels as in our approach. A similar approach has been used also more recently in a probabilistic setting [44], but with the aim of estimating the probability for a fixed set of arguments of being conflict-free or admissible, rather than to identify a formula whose models characterize the extensions under a given semantics.

In [32, 33] various NP fragments of abstract argumentation reasoning problems located at the second level of the polynomial hierarchy are identified. Driven by this analysis, complexity-sensitive algorithms for preferred semantics (as well as semi-stable and stage semantics) are proposed which require a polynomial number of calls to a SAT solver in the identified NP fragments. In particular, the proof of Theorem 3 in [33] contains a procedure for the enumeration of preferred extensions resembling Algorithm 5, and a procedure for skeptical acceptance under preferred semantics is reported which shares the basic schema underlying our Algorithms 8 and 9. There are however substantial differences w.r.t. [32, 33]. From a strictly technical point of view, we make use of labelling-based encodings, while in [33] similarly to [43] a single variable is associated to each argument. Moreover, our algorithms are somewhat generic w.r.t. the specific functions that identify the complete labellings at the basis of the search for preferred labellings (see in particular the definitions of $\underline{\mathsf{FindCL}}^{\ddagger}$ and $\underline{\mathsf{FindCLU}}^{\ddagger}_{\mathbf{a}}$ in Section 3.2). This feature as well as the use of labellings are at the basis of the novel strategy underlying Algorithm 9, which at each iteration looks for a maximal complete labelling where $\mathbf{a}$ is `undec`, and then checks with a single call to the SAT solver whether it can be extended to a preferred labelling where $\mathbf{a}$ is `in`, whereas the algorithm in [33] does not use this constraint in maximising complete extensions. From a more general point of view, our focus is different w.r.t. [32, 33], since we are concerned with a systematic exploration and evaluation of the possible encodings and algorithmic variants underlying the above mentioned methodology.

The complexity of the problem of enumerating extensions is investigated in [11] under different semantics and by considering classes of argumentation frameworks that lead to tractability. In the case of symmetric argumentation frameworks, preferred extensions coincide with maximal conflict-free sets, that can be computed with polynomial delay (between one solution and the subsequent one) by means of e.g. the algorithm proposed in [45] to enumerate maximal independent sets in undirected graphs. This also applies to stable semantics for symmetric argumentation frameworks without self-defeating arguments, since in this case stable semantics coincide with preferred semantics. Another tractable class considered in [45] includes argumentation frameworks without even cycles, and in particular acyclic graphs. In this case, there is a unique complete and preferred extension coinciding with the grounded extension, which can be computed in polynomial time. As to stable semantics, it can also be easily checked if the grounded extension is stable.

Since it is possible to determine in polynomial time whether a graph belongs to one of the above classes, it would be easy to use a dedicated polynomial algorithm to solve tractable input instances (in the above sense). Dedicated algorithms can then be used in a portfolio fashion: a pre-processing step can be put in place to identify whether a framework is a member of one of the above mentioned classes, and select the appropriate algorithm to use. There is an extensive literature on portfolio approaches, and techniques to select online promising algorithms. While the exploration of portfolio-based approaches for dealing with argumentation problems is outside the scope of this paper, we refer the interested reader to [46, 47] for an overview of the area, and to [48, 49, 50] for more general information about portfolio approaches in AI. On the other hand, it has to be remarked that the tractable classes mentioned above, while interesting from a computational complexity point of view, represent very specific instances from a knowledge representation perspective if one considers the structure of original arguments. For instance, symmetric argumentation frameworks require any attacker to be counterattacked, a condition which is violated whenever two rebutting arguments have different strength, or whenever an argument undercuts another argument by denying the application of a defeasible rule used in its construction. As to the class where even-length cycles are not present, this condition prevents e.g. two equally plausible arguments to attack each other, a situation which is very common in structured argumentation.

In [21] we discussed a parallel algorithm for computing preferred extensions based on the SCC-decomposability. Since ArgSemSAT exploits a SAT solver as a black box, any solver that supports the DIMACS format—the standard language used in SAT competitions—can be easily integrated and tested. Therefore, if ArgSemSAT is running on a multi-core machine, this black-box approach allows to take advantage of parallel SAT solvers as well. In particular, since the most effective approach to exploit parallelisation is to run a portfolio of solvers [51], this will result in having multiple SAT solvers invoked by ArgSemSAT, and maximising the use of available cores.

Finally, it has to be remarked that some of the results of this paper are based on a preliminary approach we presented in [52], where in particular the logically equivalent encodings of complete labellings exploited in this paper has been first identified. In [52] we only tackle the problem of enumerating preferred extensions, with a preliminary algorithm which is not parameterised w.r.t. all the different variants considered in this paper. Such variants include e.g. different ways to encode the constraints on complete labellings, different SAT solvers with a variety of configurations, and the exploitation of stable labellings. More generally, we believe that the main advancement of the present paper is the identification and application of the general methodology introduced in Section 1, which led to the winning solver of the preferred semantics track at ICCMA 2017.

We refrain from further comparison with state of the art approaches as this has been the subject of ICCMA 2015 [27, 53]; a re-run of the competition to further analyse the results [13]; and a survey of various approaches [54], which in turn has been superseded by a chapter in the Handbook of Formal Argumentation [55], also published as a journal paper in [56]. We refer an interested reader to those references for further analysis on other techniques for solving problems in abstract argumentation.

8. Conclusion

In this paper we illustrated the design choices that led to the development of ArgSemSAT, which managed the first place at the track devoted to preferred semantics of the most recent edition of the international competition on abstract argumentation (ICCMA 2017). The comprehensive experimental analysis we performed informed not only the development of ArgSemSAT but provided us with useful insight we desire to share with the community.

First of all, as discussed in Section 6.2.1, although there are 18 correct logically equivalent SAT encodings of complete labellings, one of them leads to the highest performance independently of the problem and the semantics considered. Moreover, as discussed in Sections 6.2.2 and 6.2.3, composing different techniques such as AllSAT and enumerating stable extensions when searching for preferred extensions, brings advantages. Finally, as we discussed in Section 6.2.4 for the case of skeptical acceptance of arguments w.r.t. preferred semantics, injecting domain specific knowledge in the algorithm design can lead to statistically significant improvements.

Such insight has been attained through the systematic analysis that we devised of different labelling-based encodings and of the ways the relevant constraints can be expressed, and the empirical evaluation of a number of algorithm variations as well as the use of different SAT solvers. While relatively simple algorithms, in particular without resorting to graph decomposition, seem to achieve the best performance at the current stage of the research, the experimental analysis shows that our methodology is a key factor in boosting solver coverage and speed. In fact, the choices concerning encodings, constraints and SAT solvers are interdependent, thus a systematic analysis and experimentation is needed to find a good combination.

As to future work, we believe our approach can be extended in several respects. First, there is a significant interest towards graph-decomposition techniques to lower the computational effort [57]. Experiments using the SCC-based schema [58] show that advantages can be achieved in the case the number of strongly connected components is high [15] or the density is low [14]. A related issue is the development of incremental algorithms in presence of addition/removal of arguments and attacks [59, 60, 61]. Another interesting extension is the use of stochastic local search in incomplete SAT-based algorithms, which has recently been proposed in [62] for enumerating preferred extensions. Moroever, SAT-based techniques can be exploited for several other semantics, such as semi-stable, ideal and eager semantics [63]. Again, we remark that the methodology described in this paper may play a key role in boosting performance also in these extended contexts.

Acknowledgements

The authors wish to thank Yinlei Yu, Pramod Subramanyan, Nestan Tsiskaridze, and Sharad Malik, for having shared the implementation of the AllSAT solver they developed for [28]; and Alessandro Cherubini and Tobia Zanetti for their contribution in coding ArgSemSAT.

References

[1] K. Atkinson, P. Baroni, M. Giacomin, A. Hunter, H. Prakken, C. Reed, G. Simari, M. Thimm, S. Villata, Toward artificial argumentation, AI Magazine 38 (3) (2017) 25–36.

[2] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games, Artificial Intelligence 77 (2) (1995) 321–357.

[3] C. Cayrol, On the relation between argumentation and non-monotonic coherence based entailment, in: Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95), Morgan Kaufmann Publishers Inc., 1995, pp. 1443–1448.

[4] G. Governatori, M. Maher, G. Antoniou, D. Billington, Argumentation semantics for defeasible logic, Journal of Logic and Computation 14 (5) (2004) 675–702.

[5] M. Caminada, S. Sá, J. Alcântara, W. Dvŏrák, On the equivalence between logic programming semantics and argumentation semantics, International Journal of Approximate Reasoning 58 (2015) 87–111.

[6] H. Prakken, An abstract framework for argumentation with structured arguments, Argument and Computation 1 (2) (2010) 93–124.

[7] P. Baroni, M. Giacomin, Semantics of abstract argumentation systems, in: Argumentation in Artificial Intelligence, Springer, 2009, pp. 25–44.

[8] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowledge Engineering Review 26 (4) (2011) 365–410.

[9] M. David, S. Karl, Floating conclusions and zombie paths: Two deep difficulties in the "directly skeptical" approach to defeasible inheritance nets, Artificial Intelligence 48 (2) (1991) 199–209.

[10] P. E. Dunne, M. Wooldridge, Complexity of abstract argumentation, in: Argumentation in Artificial Intelligence, Springer, 2009, pp. 85–104.

[11] M. Kröll, R. Pichler, S. Woltran, On the complexity of enumerating the extensions of abstract argumentation frameworks, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 1145–1152. doi:10.24963/ijcai.2017/159.

[12] A. Toniolo, T. Norman, A. Etuk, F. Cerutti, R. Ouyang, M. Srivastava, N. Oren, T. Dropps, J. Allen, P. Sullivan, Supporting reasoning with different types of evidence in intelligence analysis, in: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, Vol. 2, 2015, pp. 781–789.

[13] F. Cerutti, M. Vallati, M. Giacomin, On the impact of configuration on abstract argumentation automated reasoning, International Journal of Approximate Reasoning 92 (2018) 120 – 138. doi:https://doi.org/10.1016/j.ijar.2017.10.002.
URL `http://www.sciencedirect.com/science/article/pii/S0888613X16303085`

[14] B. Liao, L. Lei, J. Dai, Computing Preferred Labellings by Exploiting SCCs and Most Sceptically Rejected Arguments, in: Second International Workshop on Theory and Applications of Formal Argumentation (TAFA-13), 2013, pp. 194–208.

[15] F. Cerutti, M. Giacomin, M. Vallati, M. Zanella, An SCC recursive meta-algorithm for computing preferred labellings in abstract argumentation, in: Proc. of the 14th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2014), 2014, pp. 42–51.

[16] W. Faber, M. Vallati, F. Cerutti, M. Giacomin, Solving set optimization problems by cardinality optimization with an application to argumentation, in: ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), 2016, pp. 966–973.

[17] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE transactions on evolutionary computation 1 (1) (1997) 67–82.

[18] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. H. Hoos, K. Leyton-Brown, The configurable SAT solver challenge (CSSC), Artif. Intell. 243 (2017) 1–25.

[19] S. Kadioglu, Y. Malitsky, M. Sellmann, K. Tierney, Isac–instance-specific algorithm configuration, in: Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence, 2010, pp. 751–756.

[20] C. Fawcett, H. H. Hoos, Analysing differences between algorithm configurations through ablation, J. Heuristics 22 (4) (2016) 431–458.

[21] F. Cerutti, I. Tachmazidis, M. Vallati, S. Batsakis, M. Giacomin, G. Antoniou, Exploiting parallelism for hard problems in abstract argumentation, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA., 2015, pp. 1475–1481.

[22] P. Baroni, F. Cerutti, P. E. Dunne, M. Giacomin, Automata for infinite argumentation structures, Artificial Intelligence 203 (0) (2013) 104 – 150.

[23] B. Verheij, Two approaches to dialectical argumentation: admissible sets and argumentation stages, in: J.-J. C. Meyer, L. C. van der Gaag (Eds.), NAIC'96. Proceedings of the Eighth Dutch Conference on Artificial Intelligence, Vol. 96, 1996, pp. 357–368.

[24] B. Verheij, Artificial argument assistants for defeasible argumentation, Artificial Intelligence 150 (1) (2003) 291–324. doi:10.1016/S0004-3702(03)00107-3.

[25] M. Caminada, On the issue of reinstatement in argumentation, in: Proceedings of JELIA 2006, 2006, pp. 111–123.

[26] M. Caminada, G. Pigozzi, On judgment aggregation in abstract argumentation, Autonomous Agents and Multi-Agent Systems 22 (1) (2011) 64–102.

[27] M. Thimm, S. Villata, F. Cerutti, N. Oren, H. Strass, M. Vallati, Summary Report of The First International Competition on Computational Models of Argumentation, AI Magazine.
URL http://eprints.hud.ac.uk/26063/

[28] Y. Yu, P. Subramanyan, N. Tsiskaridze, S. Malik, All-SAT Using Minimal Blocking Clauses, in: 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, IEEE, 2014, pp. 86–91. doi:10.1109/VLSID.2014.22.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6733111

[29] K. L. McMillan, Applying SAT Methods in Unbounded Symbolic Model Checking, in: Proceedings of the 14th International Conference on Computer Aided Verification, Springer, Berlin, Heidelberg, 2002, pp. 250–264.
URL http://link.springer.com/10.1007/3-540-45657-0{_}19

[30] J. Brauer, A. King, J. Kriener, Existential Quantification As Incremental SAT, in: Proceedings of the 23rd International Conference on Computer Aided Verification, CAV'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 191–207.

[31] M. Caminada, D. M. Gabbay, A logical account of formal argumentation, Studia Logica (Special issue: new ideas in argumentation theory) 93 (2–3) (2009) 109–145.

[32] W. Dvŏrák, M. Järvisalo, J. P. Wallner, S. Woltran, Complexity-sensitive decision procedures for abstract argumentation, in: Proceedings of KR 2012, AAAI Press, 2012, pp. 54–64.

[33] W. Dvŏrák, M. Järvisalo, J. P. Wallner, S. Woltran, Complexity-sensitive decision procedures for abstract argumentation, Artificial Intelligence 206 (2014) 53–78.
URL http://www.sciencedirect.com/science/article/pii/S0004370213001069

[34] M. Vallati, L. Chrpa, M. Grzes, T. L. McCluskey, M. Roberts, S. Sanner, The 2014 international planning competition: Progress and trends, AI Magazine 36 (3) (2015) 90–98.
URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/2571

[35] M. Vallati, L. Chrpa, T. L. McCluskey, What you always wanted to know about the deterministic part of the international planning competition (IPC) 2014 (but were too afraid to ask), Knowledge Eng. Review 33 (2018) e3.

[36] N. Eén, N. Sörensson, An Extensible SAT-solver, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 502–518.
URL http://dx.doi.org/10.1007/978-3-540-24605-3{_}37

[37] G. Audemard, L. Simon, Lazy clause exchange policy for parallel sat solvers, in: Theory and Applications of Satisfiability Testing–SAT 2014, 2014, pp. 197–205.

[38] M. R. Krom, The decision problem for a class of first-order formulas in which all disjunctions are binary, Mathematical Logic Quarterly 13 (1-2) (1967) 15–20. doi:10.1002/malq.19670130104.
URL `http://dx.doi.org/10.1002/malq.19670130104`

[39] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings, 2005, pp. 61–75.

[40] A. Biere, M. Heule, H. van Maaren, T. Walsh, Conflict-driven clause learning sat solvers, Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications (2009) 131–153.

[41] S. Hölldobler, N. Manthey, A. Saptawijaya, Improving resource-unaware sat solvers, in: International Conference on Logic for Programming Artificial Intelligence and Reasoning, Springer, 2010, pp. 519–534.

[42] O. Fourdrinoy, É. Grégoire, B. Mazure, L. Saïs, Eliminating redundant clauses in sat instances, in: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 2007, pp. 71–83.

[43] P. Besnard, S. Doutre, Checking the acceptability of a set of arguments, in: Proceedings of 10th International Workshop on Non-Monotonic Reasoning (NMR 2004), 2004, pp. 59–64.

[44] B. Fazzinga, S. Flesca, F. Parisi, On efficiently estimating the probability of extensions in abstract argumentation frameworks, International Journal of Approximate Reasoning 69 (2016) 106 – 132. doi:https://doi.org/10.1016/j.ijar.2015.11.009.
URL `http://www.sciencedirect.com/science/article/pii/S0888613X15001760`

[45] D. S. Johnson, C. H. Papadimitriou, M. Yannakakis, On generating all maximal independent sets, Information Processing Letters 27 (3) (1988) 119–123.

[46] F. Cerutti, M. Vallati, M. Giacomin, Where are we now? state of the art and future trends of solvers for hard argumentation problems, in: P. Baroni, T. Gordon, T. Scheffler (Eds.), Proceedings of 6th International Conference on Computational Models of Argument, COMMA 2016, Vol. 287 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2016, pp. 207–218. doi:10.3233/978-1-61499-686-6-207.

[47] M. Vallati, F. Cerutti, M. Giacomin, On the combination of argumentation solvers into parallel portfolios, in: AI 2017: Advances in Artificial Intelligence - 30th Australasian Joint Conference, 2017, pp. 315–327.

[48] B. Hurley, L. Kotthoff, Y. Malitsky, D. Mehta, B. O'Sullivan, Advanced portfolio techniques, in: Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach, 2016, pp. 191–225.

[49] F. Hutter, L. Xu, H. H. Hoos, K. Leyton-Brown, Algorithm runtime prediction: Methods & evaluation, Artif. Intell. 206 (2014) 79–111.

[50] M. Lindauer, H. H. Hoos, K. Leyton-Brown, T. Schaub, Automatic construction of parallel portfolios via algorithm configuration, Artif. Intell. 244 (2017) 272–290.

[51] T. Balyo, C. Sinz, Parallel satisfiability, in: Handbook of Parallel Constraint Reasoning., 2018, pp. 3–29.

[52] F. Cerutti, P. E. Dunne, M. Giacomin, M. Vallati, Computing preferred extensions in abstract argumentation: A sat-based approach, in: E. Black, S. Modgil, N. Oren (Eds.), Theory and Applications of Formal Argumentation, Vol. 8306 LNAI, Springer Verlag, Berlin, Heidelberg, 2014, pp. 176–193.

[53] M. Thimm, S. Villata, The first international competition on computational models of argumentation: Results and analysis, Artificial Intelligence 252 (2017) 267–294.

[54] G. Charwat, W. Dvŏrák, S. A. Gaggl, J. P. Wallner, S. Woltran, Methods for solving reasoning problems in abstract argumentation: A survey, Artificial Intelligence 220 (2015) 28 – 63. doi:https://doi.org/10.1016/j.artint.2014.11.008.
URL `http://www.sciencedirect.com/science/article/pii/S0004370214001404`

[55] F. Cerutti, S. A. Gaggl, M. Thimm, J. P. Wallner, Foundations of implementations for formal argumentation, in: P. Baroni, D. Gabbay, M. Giacomin, L. van der Torre (Eds.), Handbook of Formal Argumentation, College Publications, 2018, Ch. 15.

[56] F. Cerutti, S. A. Gaggl, M. Thimm, J. P. Wallner, Foundations of implementations for formal argumentation, IfCoLog Journal of Logics and their Applications 4 (8) (2017) 2623–2706.

[57] W. Dvŏrák, R. Pichler, S. Woltran, Towards fixed-parameter tractable algorithms for abstract argumentation, Artificial Intelligence 186 (2012) 1 – 37. doi:https://doi.org/10.1016/j.artint.2012.03.005.
URL `http://www.sciencedirect.com/science/article/pii/S0004370212000264`

[58] P. Baroni, M. Giacomin, G. Guida, SCC-recursiveness: a general schema for argumentation semantics, Artificial Intelligence 168 (1-2) (2005) 165–210.

[59] S. Greco, F. Parisi, Efficient computation of deterministic extensions for dynamic abstract argumentation frameworks, in: ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), 2016, pp. 1668–1669. doi:10.3233/978-1-61499-672-9-1668.
URL `https://doi.org/10.3233/978-1-61499-672-9-1668`

[60] S. Greco, F. Parisi, Incremental computation of deterministic extensions for dynamic argumentation frameworks, in: Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings, 2016, pp. 288–304.

[61] G. Alfano, S. Greco, F. Parisi, Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 49–55. doi:10.24963/ijcai.2017/8.
URL `https://doi.org/10.24963/ijcai.2017/8`

[62] D. Niu, L. Liu, S. Lü, A new stochastic local search approach for computing preferred extensions of abstract argumentation, in: ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), 2016, pp. 1652–1653. doi:10.3233/978-1-61499-672-9-1652.
URL `https://doi.org/10.3233/978-1-61499-672-9-1652`

[63] J. Wallner, G. Weissenbacher, S. Woltran, Advanced sat techniques for abstract argumentation, in: Proceedings of CLIMA2013, 2013, pp. 138–154.

Appendix A. Proofs

Theorem 2. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ and $\mathbf{a} \in \mathcal{A}$:

- $\{\mathtt{in}(\mathcal{L}ab) \mid \mathcal{L}ab \in \textbf{EL-PR}(\Gamma))\} = \mathcal{E}_{\mathsf{PR}}(\Gamma)$ (Algorithm 5);
- $\{\mathtt{in}(\mathcal{L}ab) \mid \mathcal{L}ab \in \textbf{EL-PR-withST}(\Gamma))\} = \mathcal{E}_{\mathsf{PR}}(\Gamma)$ (Algorithm 6);
- $\textbf{DC-PR}(\Gamma, \mathbf{a}) \equiv (\exists S \in \mathcal{E}_{\mathsf{PR}}(\Gamma), \mathbf{a} \in S)$ (Algorithm 7);
- $\textbf{DS-PR-straightforward}(\Gamma, \mathbf{a}) \equiv (\forall S \in \mathcal{E}_{\mathsf{PR}}(\Gamma), \mathbf{a} \in S)$ (Algorithm 8);
- $\textbf{DS-PR}(\Gamma, \mathbf{a}) \equiv (\forall S \in \mathcal{E}_{\mathsf{PR}}(\Gamma), \mathbf{a} \in S)$ (Algorithm 9).

Proof. Considering Algorithm 5, we have to show that $\textbf{EL-PR}(\Gamma) = \mathfrak{L}_{\mathsf{PR}}(\Gamma)$ (the desired result then follows by Proposition 4). Let us first focus on the inner loop (lines 3-5) and let us indicate as $\mathcal{L}ab^0$ the complete labelling assigned to the variable $\mathcal{L}ab$ in line 2 before the inner loop execution. According to the definition of $\underline{\mathsf{FindCL}}^{\sqsupsetneq}$, the labellings assigned to the variable $\mathcal{L}ab$ (line 4) at each iteration of the inner loop satisfy the conditions $\mathcal{L}ab^0 \sqsubsetneq \mathcal{L}ab^1 \sqsubsetneq \ldots$, where each $\mathcal{L}ab^i$ is a complete labelling. By transitivity and anti-symmetry of $\sqsubseteq$, all labellings $\mathcal{L}ab^i$ are distinct, and taking into account that Γ is finite (and so is the number of its possible labellings) this entails that the corresponding sequence is finite as well and the inner loop terminates. In particular, the loop terminates when $\underline{\mathsf{FindCL}}^{\sqsupsetneq}(\Gamma, \mathcal{L}ab) = \bot$, which by the definition of $\underline{\mathsf{FindCL}}^{\sqsupsetneq}$ entails that the labelling in $\mathcal{L}ab$ is maximal w.r.t. $\sqsubseteq$, i.e. a preferred labelling on the basis of Proposition 3. Summing up, the execution of lines 3-6 starts from a complete labelling $\mathcal{L}ab$ and includes in $\mathcal{L}_p$ (line 6) a preferred labelling $\mathcal{L}ab'$ such that $\mathcal{L}ab \sqsubseteq \mathcal{L}ab'$. Let us now consider the outer loop (lines 2-7). First, the loop terminates, since as shown below the labelling $\mathcal{L}ab$ added to $\mathcal{L}_p$ in line 6 does not belong to $\mathcal{L}_p$: taking into account that Γ is finite, $\mathcal{L}_p$ cannot grow indefinitely. To see that $\mathcal{L}ab \notin \mathcal{L}_p$ before the execution of line 6, note that $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_p)$ returns in line 6 a complete labelling which is not less committed than any of the labellings in $\mathcal{L}_p$, thus by transitivity of $\sqsubseteq$ the labelling $\mathcal{L}ab$ added to $\mathcal{L}_p$ in line 6 does not belong to $\mathcal{L}_p$. Now, according to the considerations above, line 6 adds to $\mathcal{L}_p$ a preferred labelling, thus in line 2 it always holds that $\mathcal{L}_p \subseteq \mathfrak{L}_{\mathsf{PR}}(\Gamma)$. Since the loop terminates when $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_p)$ returns $\bot$ in line 2, by the definition of $\underline{\mathsf{FindCL}}^{\not\sqsubseteq}$ it holds that $(\mathfrak{L}_{\mathsf{PR}}(\Gamma) \backslash \mathcal{L}_p) \subseteq \varnothing$, i.e. $\mathfrak{L}_{\mathsf{PR}}(\Gamma) \subseteq \mathcal{L}_p$. As a consequence, $\mathcal{L}_p = \mathfrak{L}_{\mathsf{PR}}(\Gamma)$ in line 8, as desired.

Turning to Algorithm 6, its correctness follows from that of Algorithm 5 taking into account that $\mathfrak{L}_{\mathsf{ST}}(\Gamma) \subseteq \mathfrak{L}_{\mathsf{PR}}(\Gamma)$.

To show the correctness of Algorithm 7, we distinguish two cases. If $\mathbf{a}$ is credulously accepted then there is a preferred labelling where $\mathbf{a}$ is `in`, and since a preferred labelling is by definition a complete labelling $\underline{\mathsf{FindCL}}_{\mathbf{a}}(\Gamma, \mathtt{in})$ is able to return a labelling in line 1, and the algorithm correctly returns $\top$. In the case where $\mathbf{a}$ is not credulously accepted, there is no complete labelling where $\mathbf{a}$ is `in`, since according to Proposition 3 this would entail the presence of a preferred labelling where $\mathbf{a}$ is `in`. As a consequence, the algorithm correctly returns $\bot$.

The correctness of Algorithm 8 follows from the above proof concerning Algorithm 5. In particular, we know that the algorithm terminates and that during the execution of the loop of lines 2-10 the variable $\mathcal{L}ab$ in lines 6-9 iterates over the set of preferred labellings. Thus, if $\mathbf{a}$ is not skeptically accepted then there is an iteration of the loop such that the algorithm returns $\bot$ in line 7, otherwise all preferred labellings assign to $\mathbf{a}$ the label `in` and the loop terminates, thus the algorithm correctly returns $\top$ in line 11.

Let us finally turn to Algorithm 9. It is easy to see that if the algorithm terminates then it either returns $\top$ or $\bot$. We thus show that the algorithm always terminates and whenever it returns a value, this is correct.

As to termination, there are two nested loops in the algorithm. Focusing first on the inner loop (lines 6-8) and denoting as $\mathcal{L}ab^0$ the labelling in the variable $\mathcal{L}ab$ before the execution of the loop, by the definition of $\underline{\mathsf{FindCL}}_{\mathbf{a}}^{\sqsupsetneq}$ the values in variable $\mathcal{L}ab$ at line 6 identify a sequence of labellings $\mathcal{L}ab^0 \sqsubsetneq \mathcal{L}ab^1 \sqsubsetneq \ldots$. As in the proof concerning Algorithm 5, transitivity and anti-symmetry of $\sqsubseteq$ guarantee that the sequence includes distinct elements, and since the argumentation framework is finite then the sequence is finite as well. As a consequence, after a finite number of iterations the inner loop terminates with a labelling in $\mathcal{L}ab$ such that $\mathcal{L}ab^0 \sqsubseteq \mathcal{L}ab$ and $\mathcal{L}ab \in \mathcal{L}_{mu}$ (i.e. $\mathcal{L}ab$ is a maximal complete labelling such that $\mathcal{L}ab(\mathbf{a}) = \texttt{undec}$). As to the loop of lines 5-13, we prove that at each iteration line 12 increases the cardinality of the set in $\mathcal{L}_c$, and again, since Γ is finite, this entails termination of the loop. To prove that $\mathcal{L}ab$ before the execution of line 12 does not belong to $\mathcal{L}_c$, let us consider the case where $\underline{\mathsf{FindCLU}}_{\mathbf{a}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_c)$ in line 5 returns a labelling $\mathcal{L}ab^0$ (i.e. different than $\perp$) which is assigned to the variable $\mathcal{L}ab$. By the definition of $\underline{\mathsf{FindCLU}}_{\mathbf{a}}^{\not\sqsubseteq}$, it holds that $\forall \mathcal{L}ab' \in \mathcal{L}_c, \mathcal{L}ab^0 \not\sqsubseteq \mathcal{L}ab'$, i.e. $\mathcal{L}ab^0$ is not less or equally committed w.r.t. a labelling of $\mathcal{L}_c$. As shown above, the inner loop obtains a labelling $\mathcal{L}ab$ such that $\mathcal{L}ab^0 \sqsubseteq \mathcal{L}ab$, thus it can not be the case that $\mathcal{L}ab \in \mathcal{L}_c$.

We now show that the value returned by the algorithm is always correct. Focusing first on lines 1-3, according to the definition of $\underline{\mathsf{FindCL}}_{\mathbf{a}}$ line 2 is executed if there is no complete labelling where $\mathbf{a}$ is `in` or there is a complete labelling where $\mathbf{a}$ is `out`. It is easy to see that in both cases $\mathbf{a}$ is not skeptically justified, thus the returned value $\perp$ is correct. In particular, since preferred labellings are also complete labellings, in the first case there is no preferred labelling where $\mathbf{a}$ is `in`, and taking into account that a preferred labelling always exists (see e.g. [2]) this entails that $\mathbf{a}$ is not skeptically justified. In the second case, according to Proposition 3 there is a preferred labelling where $\mathbf{a}$ is `out`, and again $\mathbf{a}$ is not skeptically justified.

Let us now consider the case where the algorithm enters line 4. Note that by the considerations above there is no preferred labelling where $\mathbf{a}$ is `out`. Focusing on lines 9-10, let us show that if line 10 is executed then $\mathbf{a}$ is not skeptically justified. According to the definition of $\underline{\mathsf{FindCL}}_{\mathbf{a}}^{\sqsupsetneq}$, this happens if there is no complete labelling $\mathcal{L}ab'$ such that $\mathcal{L}ab'(\mathbf{a}) = \texttt{in}$ and $\mathcal{L}ab \sqsubsetneq \mathcal{L}ab'$. Since $\mathcal{L}ab \in \mathcal{L}_{mu}$ and there are no preferred labellings where $\mathbf{a}$ is `out`, in this case $\mathcal{L}ab$ is a maximal complete labelling, i.e. a preferred labelling by Proposition 3. As a consequence, there is a preferred labelling where $\mathbf{a}$ is `undec` entailing that it is not skeptically justified.

To complete the proof, we first show that, in line 5, $\mathcal{L}_c \subseteq \mathcal{L}_{mu}$ and $\mathcal{L}_c \cap \mathfrak{L}_{\mathsf{PR}}(\Gamma) = \varnothing$. This is obviously true before the first iteration of the outer loop, since $\mathcal{L}_c = \varnothing$. It is also true after each iteration of the loop, since we have shown above that the labelling $\mathcal{L}ab$ added to $\mathcal{L}_c$ in line 12 belongs to $\mathcal{L}_{mu}$, and taking into account lines 9-11 there is a complete labelling $\mathcal{L}ab'$ such that $\mathcal{L}ab \sqsubsetneq \mathcal{L}ab'$, i.e. $\mathcal{L}ab$ is not maximal w.r.t. $\sqsubseteq$ and thus it is not a preferred labelling. Now, the algorithm enters line 14 only if $\underline{\mathsf{FindCLU}}_{\mathbf{a}}^{\not\sqsubseteq}(\Gamma, \mathcal{L}_c)$ in line 5 returns $\perp$. According to the definition of $\underline{\mathsf{FindCLU}}_{\mathbf{a}}^{\not\sqsubseteq}$, this entails that $\{\mathcal{L}ab' \in \mathfrak{L}_{\mathsf{PR}}(\Gamma) \mid \mathcal{L}ab'(\mathbf{a}) = \texttt{undec}\} \subseteq \mathcal{L}_c$, where $\mathcal{L}_c \cap \mathfrak{L}_{\mathsf{PR}}(\Gamma) = \varnothing$. Then $\{\mathcal{L}ab' \in \mathfrak{L}_{\mathsf{PR}}(\Gamma) \mid \mathcal{L}ab'(\mathbf{a}) = \texttt{undec}\}$ is empty, and taking into account that there are no preferred labellings where $\mathbf{a}$ is $\mathtt{out}$, we conclude that $\mathbf{a}$ is skeptically justified, and the value returned in line 14 is correct. $\square$

Proposition 5. Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. A total function $\mathcal{L}ab : \mathcal{A} \mapsto \{\mathtt{in}, \mathtt{out}, \mathtt{undec}\}$ is a complete labelling iff it satisfies any of the following conjunctive constraints for any $\mathbf{a} \in \mathcal{A}$:

$C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \sim}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}}$

$C^{\substack{\leftrightarrow\\ \sim\\ \leftrightarrow}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$

$C^{\substack{\sim\\ \leftrightarrow\\ \leftrightarrow}} = C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$

$C^{\substack{\rightarrow\\ \rightarrow\\ \rightarrow}} = C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$

$C^{\substack{\leftarrow\\ \leftarrow\\ \leftarrow}} = C^{\leftarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}}$

Proof. We prove that each of the 5 conjunctive constraints is equivalent to $C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \leftrightarrow}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$, i.e. the constraint expressed in Definition 7. All of them are obviously entailed by $C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \leftrightarrow}}$, thus we have to prove that each of them entails $C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \leftrightarrow}}$.
As to $C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \sim}}$, $C^{\substack{\leftrightarrow\\ \sim\\ \leftrightarrow}}$ and $C^{\substack{\sim\\ \leftrightarrow\\ \leftrightarrow}}$, the result derives from the fact that $\mathcal{L}ab$ is a function. In particular, let us consider an argument $\mathbf{a}$ which satisfies $C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \sim}}$. Let us first prove $C^{\leftarrow}_{\mathtt{undec}}$. If $\forall \mathbf{b} \in \mathbf{a}^{-} \mathcal{L}ab(\mathbf{b}) \neq \mathtt{in} \wedge \exists \mathbf{c} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{c}) = \mathtt{undec}$, then $\mathcal{L}ab(\mathbf{a}) \neq \mathtt{in}$ because of $C^{\rightarrow}_{\mathtt{in}}$, and $\mathcal{L}ab(\mathbf{a}) \neq \mathtt{out}$ because of $C^{\rightarrow}_{\mathtt{out}}$, thus the only possibility is that $\mathcal{L}ab(\mathbf{a}) = \mathtt{undec}$. As to $C^{\rightarrow}_{\mathtt{undec}}$, if $\mathcal{L}ab(\mathbf{a}) = \mathtt{undec}$ then by $C^{\leftarrow}_{\mathtt{out}}$ $\forall \mathbf{b} \in \mathbf{a}^{-} \mathcal{L}ab(\mathbf{b}) \neq \mathtt{in}$, thus by $C^{\leftarrow}_{\mathtt{in}}$ $\exists \mathbf{c} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{c}) = \mathtt{undec}$. Overall, $C^{\leftrightarrow}_{\mathtt{undec}}$ is satisfied, entailing $C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \leftrightarrow}}$. The proofs for $C^{\substack{\leftrightarrow\\ \sim\\ \leftrightarrow}}$ and $C^{\substack{\sim\\ \leftrightarrow\\ \leftrightarrow}}$ are analogous.
As to $C^{\substack{\rightarrow\\ \rightarrow\\ \rightarrow}}$, with respect to $C^{\substack{\leftrightarrow\\ \leftrightarrow\\ \leftrightarrow}}$ this constraint does not include the terms $C^{\leftarrow}_{\mathtt{in}}$, $C^{\leftarrow}_{\mathtt{out}}$ and $C^{\leftarrow}_{\mathtt{undec}}$. Here we prove that $C^{\leftarrow}_{\mathtt{in}}$ (and, similarly, $C^{\leftarrow}_{\mathtt{out}}$ and $C^{\leftarrow}_{\mathtt{undec}}$) is indeed satisfied. Let us consider an argument $\mathbf{a}$ such that $\forall \mathbf{b} \in \mathbf{a}^{-}$ $\mathcal{L}ab(\mathbf{b}) = \mathtt{out}$, and let us reason by contradiction by assuming that $\mathcal{L}ab(\mathbf{a}) \neq \mathtt{in}$. Since $\mathcal{L}ab$ is a function, if $\mathcal{L}ab(\mathbf{a}) \neq \mathtt{in}$ then either $\mathcal{L}ab(\mathbf{a}) = \mathtt{out}$ or $\mathcal{L}ab(\mathbf{a}) = \mathtt{undec}$. If $\mathcal{L}ab(\mathbf{a}) = \mathtt{out}$, from $C^{\rightarrow}_{\mathtt{out}}$ $\exists \mathbf{b} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{b}) = \mathtt{in} \neq \mathtt{out}$. If $\mathcal{L}ab(\mathbf{a}) = \mathtt{undec}$, from $C^{\rightarrow}_{\mathtt{undec}}$ $\exists \mathbf{b} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{b}) = \mathtt{undec} \neq \mathtt{out}$. The proof for $C^{\leftarrow}_{\mathtt{out}}$ and $C^{\leftarrow}_{\mathtt{undec}}$ is similar.
As to $C^{\substack{\leftarrow\\ \leftarrow\\ \leftarrow}}$, the proof follows the same line. We prove that $C^{\rightarrow}_{\mathtt{in}}$ (and, similarly, $C^{\rightarrow}_{\mathtt{out}}$ and $C^{\rightarrow}_{\mathtt{undec}}$) is indeed satisfied. Given an argument $\mathbf{a}$ such that $\mathcal{L}ab(\mathbf{a}) = \mathtt{in}$, assume by contradiction that $\exists \mathbf{b} \in \mathbf{a}^{-} : \mathcal{L}ab(\mathbf{b}) \neq \mathtt{out}$. Since $\mathcal{L}ab$ is a function, either $\mathcal{L}ab(\mathbf{b}) = \mathtt{in}$ or $\mathcal{L}ab(\mathbf{b}) = \mathtt{undec}$. In the first case, $C^{\leftarrow}_{\mathtt{out}}$ entails that $\mathcal{L}ab(\mathbf{a}) = \mathtt{out} \neq \mathtt{in}$. In the second case, either $C^{\leftarrow}_{\mathtt{out}}$ or $C^{\leftarrow}_{\mathtt{undec}}$ applies, i.e. $\mathcal{L}ab(\mathbf{a}) \in \{\mathtt{out}, \mathtt{undec}\}$ thus $\mathcal{L}ab(\mathbf{a}) \neq \mathtt{in}$. Following the same reasoning line, we can prove that also $C^{\rightarrow}_{\mathtt{out}}$ and $C^{\rightarrow}_{\mathtt{undec}}$ hold. $\square$

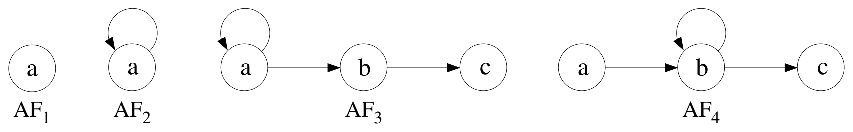

Figure A.26: Identifying some weak encodings.

Proposition 6. The following 6 encodings are weak:

$C^{\substack{\rightarrow\\\leftarrow\\\leftrightarrow}} = C^{\leftrightarrow}_{\mathtt{undec}} \wedge C^{\rightarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}}$
$C^{\substack{\leftarrow\\\rightarrow\\\leftrightarrow}} = C^{\leftrightarrow}_{\mathtt{undec}} \wedge C^{\leftarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}}$
$C^{\substack{\rightarrow\\\leftrightarrow\\\leftarrow}} = C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{undec}}$
$C^{\substack{\leftarrow\\\leftrightarrow\\\rightarrow}} = C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{undec}}$
$C^{\substack{\leftrightarrow\\\rightarrow\\\leftarrow}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}}$
$C^{\substack{\leftrightarrow\\\leftarrow\\\rightarrow}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$

Proof. For each encoding, we identify an argumentation framework and a non-complete labelling which satisfies the relevant constraint. In particular, referring to Figure A.26: for $C^{\substack{\rightarrow\\\leftarrow\\\leftrightarrow}}$, see the labelling $\{(\mathbf{a}, \mathtt{out})\}$ of AF_1; for $C^{\substack{\leftarrow\\\rightarrow\\\leftrightarrow}}$, see the labelling $\{(\mathbf{a}, \mathtt{in})\}$ of AF_2; for $C^{\substack{\rightarrow\\\leftrightarrow\\\leftarrow}}$, see the labelling $\{(\mathbf{a}, \mathtt{undec})\}$ of AF_1; for $C^{\substack{\leftarrow\\\leftrightarrow\\\rightarrow}}$, see the labelling $\{(\mathbf{a}, \mathtt{undec}), (\mathbf{b}, \mathtt{in}), (\mathbf{c}, \mathtt{out})\}$ of AF_3; for $C^{\substack{\leftrightarrow\\\rightarrow\\\leftarrow}}$, see the labelling $\{(\mathbf{a}, \mathtt{in}), (\mathbf{b}, \mathtt{undec}), (\mathbf{c}, \mathtt{undec})\}$ of AF_4; for $C^{\substack{\leftrightarrow\\\leftarrow\\\rightarrow}}$, see the labelling $\{(\mathbf{a}, \mathtt{undec}), (\mathbf{b}, \mathtt{out}), (\mathbf{c}, \mathtt{in})\}$ of AF_3. □

Theorem 3. All the encodings of cardinality 0, 1, and 2 are weak. Among the encodings of cardinality 3, $C^{\substack{\rightarrow\\\rightarrow\\\rightarrow}} = C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$ and $C^{\substack{\leftarrow\\\leftarrow\\\leftarrow}} = C^{\leftarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}}$ are correct and non-redundant, the other 18 encodings are weak. Among the encodings of cardinality 4, $C^{\substack{\leftrightarrow\\\leftrightarrow\\\sim}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}}$, $C^{\substack{\leftrightarrow\\\sim\\\leftrightarrow}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$ and $C^{\substack{\sim\\\leftrightarrow\\\leftrightarrow}} = C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$ are correct and non-redundant, the 6 encodings $C^{\substack{\leftrightarrow\\\rightarrow\\\rightarrow}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$, $C^{\substack{\rightarrow\\\leftrightarrow\\\rightarrow}} = C^{\rightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$, $C^{\substack{\rightarrow\\\rightarrow\\\leftrightarrow}} = C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$, $C^{\substack{\leftrightarrow\\\leftarrow\\\leftarrow}} = C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}}$, $C^{\substack{\leftarrow\\\leftrightarrow\\\leftarrow}} = C^{\leftarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}}$, $C^{\substack{\leftarrow\\\leftarrow\\\leftrightarrow}} = C^{\leftarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$ are correct and redundant, the other 6 encodings are weak. All the encodings of cardinality 5 and 6 are correct and redundant.

Proof. As to the first claim, it is easy to see that any encoding having cardinality 0, 1 and 2 is a strict subset of at least one (weak) encoding introduced in Proposition 6, thus it is obviously weak too. As to the encodings having cardinality 3, $(C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}})$ and $(C^{\leftarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}})$ are correct by Proposition 5, and they are non-redundant since any strict subset has a cardinality strictly lower than 3 (thus it is weak as shown above). The remaining 18 encodings of cardinality 3 can take one of the following two forms: (i) 12 encodings include $C^{\leftrightarrow}_{\mathtt{in}}$, $C^{\leftrightarrow}_{\mathtt{out}}$ or $C^{\leftrightarrow}_{\mathtt{undec}}$ and another single term; (ii) 6 encodings include two "left" and one "right" terms, or vice versa. In both cases, it is easy to check that any of these encodings is a strict subset of one of the weak encodings of Proposition 6. As to the encodings of cardinality 4, $(C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}})$, $(C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\leftrightarrow}_{\mathtt{undec}})$ and

$(C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{undec}})$ are correct by Proposition 5, and they are non-redundant since they do not contain $(C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}})$ nor $(C^{\leftarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}})$, thus any subset is weak according to the considerations above concerning the encodings of cardinality 3. Moreover, the 6 encodings $C^{\substack{\leftrightarrow\\\rightarrow\\\rightarrow}}$, $C^{\substack{\rightarrow\\\leftrightarrow\\\rightarrow}}$, $C^{\substack{\rightarrow\\\rightarrow\\\leftrightarrow}}$, $C^{\substack{\leftrightarrow\\\leftarrow\\\leftarrow}}$, $C^{\substack{\leftarrow\\\leftrightarrow\\\leftarrow}}$ and $C^{\substack{\leftarrow\\\leftarrow\\\leftrightarrow}}$ are supersets of $(C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}})$ or $(C^{\leftarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}})$, and thus correct and redundant, while the other 6 encodings are the weak ones identified in Proposition 6. Finally, all encodings of cardinality 5 and 6 contain at least one of the correct encodings of Proposition 5, thus they are correct and redundant. □

Lemma 1. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab_1, \mathcal{L}ab_2 \in \mathfrak{L}(\Gamma)$ complete labellings. $\exists \mathbf{a} \in \mathcal{A}$ such that $\mathcal{L}ab_1(\mathbf{a}) = \mathtt{out}$ and $\mathcal{L}ab_2(\mathbf{a}) \neq \mathtt{out}$ iff $\exists \mathbf{b} \in \mathcal{A}$ such that $\mathcal{L}ab_1(\mathbf{b}) = \mathtt{in}$ and $\mathcal{L}ab_2(\mathbf{b}) \neq \mathtt{in}$.

Proof. As to the first direction of the proof, assume that $\mathcal{L}ab_1(\mathbf{a}) = \mathtt{out}$ and $\mathcal{L}ab_2(\mathbf{a}) \neq \mathtt{out}$. According to Definition 7, $\exists \mathbf{b} \in \mathbf{a}^-$ such that $\mathcal{L}ab_1(\mathbf{b}) = \mathtt{in}$. Since $\mathcal{L}ab_2(\mathbf{a}) \neq \mathtt{out}$, according to Definition 7 $\nexists \mathbf{c} \in \mathbf{a}^- : \mathcal{L}ab_2(\mathbf{c}) = \mathtt{in}$, thus it must be the case that $\mathcal{L}ab_2(\mathbf{b}) \neq \mathtt{in}$.
As to the other direction of the proof, assume that $\mathcal{L}ab_1(\mathbf{b}) = \mathtt{in}$ and $\mathcal{L}ab_2(\mathbf{b}) \neq \mathtt{in}$. We consider two possible cases for $\mathcal{L}ab_2(\mathbf{b})$. If $\mathcal{L}ab_2(\mathbf{b}) = \mathtt{out}$, according to Definition 7 $\exists \mathbf{a} \in \mathbf{b}^-$ such that $\mathcal{L}ab_2(\mathbf{a}) = \mathtt{in}$. Since $\mathcal{L}ab_1(\mathbf{b}) = \mathtt{in}$, according to Definition 7 $\forall \mathbf{c} \in \mathbf{b}^-, \mathcal{L}ab_1(\mathbf{c}) = \mathtt{out}$, and this also holds for $\mathbf{a}$. If, in the other case, $\mathcal{L}ab_2(\mathbf{b}) = \mathtt{undec}$, according to Definition 7 $\exists \mathbf{a} \in \mathbf{b}^-$ such that $\mathcal{L}ab_2(\mathbf{a}) = \mathtt{undec}$, and again it must also hold that $\mathcal{L}ab_1(\mathbf{a}) = \mathtt{out}$. □

Proposition 7. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab, \mathcal{L}ab' \in \mathfrak{L}(\Gamma)$ stable labellings. All of the following statements are equivalent to $\mathcal{L}ab \neq \mathcal{L}ab'$:

1. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \mathtt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$

2. $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \mathtt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \mathtt{out}$

3. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \mathtt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$ and $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \mathtt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \mathtt{out}$

Proof. Taking into account that stable labellings do not assign the label `undec` to any argument, the condition $\mathcal{L}ab \neq \mathcal{L}ab'$ is equivalent to $\exists \mathbf{b} \in \mathtt{out}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$ or $\exists \mathbf{c} \in \mathtt{in}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{c}) = \mathtt{out}$. According to Lemma 1, this entails all of the three conditions. The fact that each of the three conditions entails $\mathcal{L}ab \neq \mathcal{L}ab'$ is obvious. □

Proposition 8. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab' \in \mathfrak{L}_{\mathsf{PR}}(\Gamma)$ a preferred labelling. Then, $\forall \mathcal{L}ab \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$, all of the following statements are equivalent to $\mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$:

1. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \mathtt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$

2. $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \mathtt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \mathtt{out}$

3. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \mathtt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$ and $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \mathtt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \mathtt{out}$

Proof. Each of the three conditions obviously entails $\mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$, therefore we can focus on the other direction of the proof.

Let us first prove that if $\mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$ then the first condition holds.

Letting $S \equiv \mathtt{in}(\mathcal{L}ab) \backslash \mathtt{in}(\mathcal{L}ab')$, it is easy to see that $S \neq \varnothing$. Indeed, $S = \varnothing$ would entail $\mathtt{in}(\mathcal{L}ab) \subseteq \mathtt{in}(\mathcal{L}ab')$ and according to Definition 7 we would also have $\mathtt{out}(\mathcal{L}ab) \subseteq \mathtt{out}(\mathcal{L}ab')$, i.e. $\mathcal{L}ab \sqsubseteq \mathcal{L}ab'$.

We now prove that $S \cap \mathtt{out}(\mathcal{L}ab') \neq \varnothing$, i.e. $\exists \mathbf{b} \in \mathtt{out}(\mathcal{L}ab')$ such that $\mathbf{b} \in S \subseteq \mathtt{in}(\mathcal{L}ab)$, entailing the first condition as desired.

For the sake of contradiction, let us assume $S \cap \mathtt{out}(\mathcal{L}ab') = \varnothing$. We prove that $\mathtt{in}(\mathcal{L}ab') \cup S$ is admissible. Let us note that, by Proposition 4, $\mathtt{in}(\mathcal{L}ab)$ is a complete extension and $\mathtt{in}(\mathcal{L}ab')$ is a preferred extension, i.e. maximal admissible. First, we prove that $\mathtt{in}(\mathcal{L}ab') \cup S$ is conflict-free. Indeed, if $S \rightarrow \mathtt{in}(\mathcal{L}ab')$ then $\mathtt{in}(\mathcal{L}ab') \rightarrow S$, since $\mathtt{in}(\mathcal{L}ab')$ is admissible. If $\mathtt{in}(\mathcal{L}ab') \rightarrow S$ then according to Definition 7 we would have $S \cap \mathtt{out}(\mathcal{L}ab') \neq \varnothing$, but this is not the case. Second, we show that $\mathtt{in}(\mathcal{L}ab') \cup S$ defends all of its arguments. In particular, $\forall \mathbf{a} \in \mathcal{A}$ with $\mathbf{a} \rightarrow \mathtt{in}(\mathcal{L}ab') \cup S$ we distinguish two cases:

1. if $\mathbf{a} \rightarrow \mathtt{in}(\mathcal{L}ab')$, then $\mathtt{in}(\mathcal{L}ab') \rightarrow \mathbf{a}$ since $\mathtt{in}(\mathcal{L}ab')$ is admissible.

2. if $\mathbf{a} \rightarrow S$ then $\mathbf{a} \rightarrow \mathtt{in}(\mathcal{L}ab)$, since $S \equiv \mathtt{in}(\mathcal{L}ab) \backslash \mathtt{in}(\mathcal{L}ab')$. This entails that $\mathtt{in}(\mathcal{L}ab) \rightarrow \mathbf{a}$, since $\mathtt{in}(\mathcal{L}ab)$ is a complete extension and thus admissible. Now, if $\mathcal{L}ab'(\mathbf{a}) = \mathtt{out}$ then by definition $\mathtt{in}(\mathcal{L}ab') \rightarrow \mathbf{a}$, otherwise $\mathtt{in}(\mathcal{L}ab) \backslash \mathtt{in}(\mathcal{L}ab') \rightarrow \mathbf{a}$, i.e. $S \rightarrow \mathbf{a}$.

Summing up, $\forall \mathbf{a} \in \mathcal{A}$ with $\mathbf{a} \rightarrow (\mathtt{in}(\mathcal{L}ab') \cup S)$, $(\mathtt{in}(\mathcal{L}ab') \cup S) \rightarrow \mathbf{a}$ holds, i.e. $(\mathtt{in}(\mathcal{L}ab') \cup S)$ is admissible. However, since $S \neq \varnothing$ and by definition $S \cap \mathtt{in}(\mathcal{L}ab') = \varnothing$, it holds that $\mathtt{in}(\mathcal{L}ab') \subsetneq (\mathtt{in}(\mathcal{L}ab') \cup S)$. Quod est absurdum because $\mathtt{in}(\mathcal{L}ab') \cup S$ is admissible contradicting the fact that $\mathtt{in}(\mathcal{L}ab')$ is a preferred extension.

Turning to the second condition, we know from the first one that $\exists \mathbf{b} \in \mathcal{A}$ such that $\mathcal{L}ab'(\mathbf{b}) = \mathtt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$. According to Definition 7, $\mathcal{L}ab'(\mathbf{b}) = \mathtt{out}$ entails that $\exists \mathbf{c} \in \mathcal{A}$ (with $\mathbf{c} \rightarrow \mathbf{b}$) such that $\mathcal{L}ab'(\mathbf{c}) = \mathtt{in}$, and $\mathcal{L}ab(\mathbf{b}) = \mathtt{in}$ entails that $\mathcal{L}ab(\mathbf{c}) = \mathtt{out}$.

The third condition obviously follows from the first two conditions. □

Proposition 9. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework and $\mathcal{L}ab, \mathcal{L}ab' \in \mathfrak{L}_{\mathsf{CO}}(\Gamma)$. All of the following statements are equivalent to $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab$:

1. $\mathtt{in}(\mathcal{L}ab') \subseteq \mathtt{in}(\mathcal{L}ab) \wedge \mathtt{out}(\mathcal{L}ab') \subseteq \mathtt{out}(\mathcal{L}ab) \wedge \exists \mathbf{b} \in \mathtt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{b}) = \mathtt{out}$

2. $\mathtt{in}(\mathcal{L}ab') \subseteq \mathtt{in}(\mathcal{L}ab) \wedge \mathtt{out}(\mathcal{L}ab') \subseteq \mathtt{out}(\mathcal{L}ab) \wedge \exists \mathbf{c} \in \mathtt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{c}) = \mathtt{in}$

3. $\mathtt{in}(\mathcal{L}ab') \subseteq \mathtt{in}(\mathcal{L}ab) \wedge \mathtt{out}(\mathcal{L}ab') \subseteq \mathtt{out}(\mathcal{L}ab) \wedge \exists \mathbf{b} \in \mathtt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{b}) = \mathtt{out} \wedge \exists \mathbf{c} \in \mathtt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{c}) = \mathtt{in}$

Proof. Each of the three conditions obviously entails $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab$. Let us then assume that $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab$, i.e. $\texttt{in}(\mathcal{L}ab') \subseteq \texttt{in}(\mathcal{L}ab) \wedge \texttt{out}(\mathcal{L}ab') \subseteq \texttt{out}(\mathcal{L}ab) \wedge \exists \mathbf{b} \in \texttt{undec}(\mathcal{L}ab') : \mathcal{L}ab(\mathbf{b}) \in \{\texttt{out}, \texttt{in}\}$. We distinguish two cases for $\mathcal{L}ab(\mathbf{b})$.
If $\mathcal{L}ab(\mathbf{b}) = \texttt{out}$, then the first condition holds. Moreover, by Lemma 1 (with $\mathcal{L}ab_1 = \mathcal{L}ab$ and $\mathcal{L}ab_2 = \mathcal{L}ab'$) there is an argument $\mathbf{c}$ such that $\mathcal{L}ab(\mathbf{c}) = \texttt{in}$ and $\mathcal{L}ab'(\mathbf{c}) \neq \texttt{in}$. Since $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab$, it cannot be the case that $\mathcal{L}ab'(\mathbf{c}) = \texttt{out}$, thus $\mathcal{L}ab'(\mathbf{c}) = \texttt{undec}$, entailing the second and third conditions.
In the other case, $\mathcal{L}ab(\mathbf{b}) = \texttt{in}$ and the second condition holds. Similarly to the previous case, by Lemma 1 there is an argument $\mathbf{c}$ such that $\mathcal{L}ab(\mathbf{c}) = \texttt{out}$ and $\mathcal{L}ab'(\mathbf{c}) \neq \texttt{out}$. Taking into account that $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab$, it must be the case that $\mathcal{L}ab'(\mathbf{c}) = \texttt{undec}$, entailing the first and third conditions. □

Proposition 10. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework, $\mathbf{a}$ an argument in $\mathcal{A}$ and $\mathcal{L}ab' \in \mathfrak{L}(\Gamma)$ a maximal complete labelling such that $\mathcal{L}ab'(\mathbf{a}) = \texttt{undec}$, i.e. $\nexists \mathcal{L}ab^* \in \mathfrak{L}(\Gamma)$ such that $\mathcal{L}ab^*(\mathbf{a}) = \texttt{undec}$ and $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab^*$. Then, $\forall \mathcal{L}ab \in \mathfrak{L}_{\mathsf{PR}}(\Gamma)$ such that $\mathcal{L}ab(\mathbf{a}) = \texttt{undec}$, all of the following statements are equivalent to $\mathcal{L}ab \neq \mathcal{L}ab'$:

1. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \texttt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \texttt{in}$
2. $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \texttt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \texttt{out}$
3. $\exists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \texttt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \texttt{in}$ and $\exists \mathbf{c} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{c}) = \texttt{in} \wedge \mathcal{L}ab(\mathbf{c}) = \texttt{out}$

Proof. Each of the three conditions obviously entails $\mathcal{L}ab \neq \mathcal{L}ab'$, therefore we can focus on the other direction of the proof.

Let us first prove that if $\mathcal{L}ab \neq \mathcal{L}ab'$ then the first condition holds.

Since $\mathcal{L}ab$ is a preferred labelling, it must be the case that $\mathcal{L}ab \not\sqsubseteq \mathcal{L}ab'$ (otherwise we would have $\mathcal{L}ab \sqsubsetneq \mathcal{L}ab'$ contradicting maximality w.r.t. $\sqsubseteq$). This in turn entails that $\texttt{in}(\mathcal{L}ab) \nsubseteq \texttt{in}(\mathcal{L}ab')$. Indeed, according to Definition 7, $\texttt{in}(\mathcal{L}ab) \subseteq \texttt{in}(\mathcal{L}ab')$ would entail $\texttt{out}(\mathcal{L}ab) \subseteq \texttt{out}(\mathcal{L}ab')$, i.e. $\mathcal{L}ab \sqsubseteq \mathcal{L}ab'$, which is not the case.

Let us now consider the set $\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab)$. Since $\texttt{in}(\mathcal{L}ab) \nsubseteq \texttt{in}(\mathcal{L}ab')$, it must be the case that $\texttt{in}(\mathcal{L}ab') \subsetneq (\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$. Moreover, since $\mathcal{L}ab'(\mathbf{a}) = \texttt{undec}$ and $\mathcal{L}ab(\mathbf{a}) = \texttt{undec}$, $\mathbf{a} \notin (\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ and it is not the case that $(\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab)) \rightarrow \mathbf{a}$ (otherwise either $\mathcal{L}ab'(\mathbf{a}) = \texttt{out}$ or $\mathcal{L}ab(\mathbf{a}) = \texttt{out}$). We reason by contradiction, by proving that if $\nexists \mathbf{b} \in \mathcal{A} : \mathcal{L}ab'(\mathbf{b}) = \texttt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \texttt{in}$ then $(\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ is a complete extension. This leads to a contradiction, since by Proposition 4 the labelling $\mathcal{L}ab^* \equiv \texttt{Ext2Lab}(\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ would be a complete labelling and, by the conditions above, $\mathcal{L}ab' \sqsubsetneq \mathcal{L}ab^*$ and $\mathcal{L}ab^*(\mathbf{a}) = \texttt{undec}$, contradicting the maximality of $\mathcal{L}ab'$.

Let us then prove that $(\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ is a complete extension. Let us note that, by Proposition 4, $\texttt{in}(\mathcal{L}ab)$ is a preferred extension and $\texttt{in}(\mathcal{L}ab')$ is a complete extension.

First, $(\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ is conflict-free. Indeed, if $\texttt{in}(\mathcal{L}ab) \rightarrow \texttt{in}(\mathcal{L}ab')$ then $\texttt{in}(\mathcal{L}ab') \rightarrow \texttt{in}(\mathcal{L}ab)$, since $\texttt{in}(\mathcal{L}ab')$ is admissible. If $\texttt{in}(\mathcal{L}ab') \rightarrow \texttt{in}(\mathcal{L}ab)$

then according to Definition 7 there would be an argument $\mathbf{b}$ such that $\mathcal{L}ab(\mathbf{b}) = \texttt{in}$ and $\mathcal{L}ab'(\mathbf{a}) = \texttt{out}$, but this has been excluded by assumption.

Second, we show that $(\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ defends all of its arguments. For all $\mathbf{b} \in \mathcal{A}$ with $\mathbf{b} \rightarrow (\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$, it holds that $\mathbf{b} \rightarrow \texttt{in}(\mathcal{L}ab')$ or $\mathbf{b} \rightarrow \texttt{in}(\mathcal{L}ab)$. Since both $\texttt{in}(\mathcal{L}ab')$ and $\texttt{in}(\mathcal{L}ab)$ are admissible, there must be an argument $\mathbf{c} \in (\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ such that $\mathbf{c} \rightarrow \mathbf{b}$.

Finally, to show that $(\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ is complete we take into account that $\texttt{in}(\mathcal{L}ab)$ is a preferred extension. Since $\texttt{in}(\mathcal{L}ab) \subseteq (\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ and $(\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$ has been proved to be admissible, it must then be the case that $\texttt{in}(\mathcal{L}ab) = (\texttt{in}(\mathcal{L}ab') \cup \texttt{in}(\mathcal{L}ab))$. Since any preferred extension is a complete extension, the conclusion follows.

Turning to the second condition, as in the proof of Proposition 8 we know from the first one that $\exists \mathbf{b} \in \mathcal{A}$ such that $\mathcal{L}ab'(\mathbf{b}) = \texttt{out} \wedge \mathcal{L}ab(\mathbf{b}) = \texttt{in}$. According to Definition 7, $\mathcal{L}ab'(\mathbf{b}) = \texttt{out}$ entails that $\exists \mathbf{c} \in \mathcal{A}$ (with $\mathbf{c} \rightarrow \mathbf{b}$) such that $\mathcal{L}ab'(\mathbf{c}) = \texttt{in}$, and $\mathcal{L}ab(\mathbf{b}) = \texttt{in}$ entails that $\mathcal{L}ab(\mathbf{c}) = \texttt{out}$.

The third condition obviously follows from the first two conditions. □

Appendix B. Summary of parameters used in algorithms

Parameter	Domain	Algorithm	Comment
$C*$	$\mathfrak{C} = \{C_{\substack{\rightarrow\\\rightarrow\\\rightarrow}}, C_{\substack{\leftarrow\\\leftarrow\\\leftarrow}}, C_{\substack{\rightarrow\\\rightarrow\\\leftrightarrow}}, C_{\substack{\rightarrow\\\leftrightarrow\\\rightarrow}}, C_{\substack{\leftarrow\\\leftarrow\\\leftrightarrow}}, C_{\substack{\leftarrow\\\leftrightarrow\\\leftarrow}}, C_{\substack{\leftrightarrow\\\rightarrow\\\rightarrow}}, C_{\substack{\leftrightarrow\\\leftarrow\\\leftarrow}}, C_{\substack{\leftrightarrow\\\leftrightarrow\\\sim}}, C_{\substack{\leftrightarrow\\\sim\\\leftrightarrow}}, C_{\substack{\sim\\\leftrightarrow\\\leftrightarrow}}, C_{\substack{\rightarrow\\\leftrightarrow\\\leftrightarrow}}, C_{\substack{\leftarrow\\\leftrightarrow\\\leftrightarrow}}, C_{\substack{\leftrightarrow\\\rightarrow\\\leftrightarrow}}, C_{\substack{\leftrightarrow\\\leftarrow\\\leftrightarrow}}, C_{\substack{\leftrightarrow\\\leftrightarrow\\\rightarrow}}, C_{\substack{\leftrightarrow\\\leftrightarrow\\\leftarrow}}, C_{\substack{\leftrightarrow\\\leftrightarrow\\\leftrightarrow}}\}$	All	The set of all the correct encodings as identified by Theorem 3
$\pi^{\mathsf{S}}_{\mathsf{All}}$	$\{\top, \bot\}$	Alg. 10, 13, 15	Parameter enabling using a AllSAT solver
$\pi^{\mathsf{S}}_{\mathsf{All;O}}$	$\{\top, \bot\}$	Alg. 10, 13, 15	When using a AllSAT solver, this parameter forces, for each found labelling, the creation of a blocking clause based on arguments labelled `in`, requiring one of them to be labelled `out`

$\pi^{\mathsf{S}}_{\mathsf{All;I}}$	$\{\top, \bot\}$	Alg. 10, 13, 15	When using a AllSAT solver, this parameter forces, for each found labelling, the creation of a blocking clause based on arguments labelled `out`, requiring one of them to be labelled `in`
$\pi^{\mathsf{S}}_{\mathsf{O}}$	$\{\top, \bot\}$	Alg. 10, 13, 15	When not using a AllSAT solver, this parameter forces, for each found labelling, the creation of a blocking clause based on arguments labelled `in`, requiring one of them to be labelled `out`
$\pi^{\mathsf{S}}_{\mathsf{I}}$	$\{\top, \bot\}$	Alg. 10, 13, 15	When not using a AllSAT solver, this parameter forces, for each found labelling, the creation of a blocking clause based on arguments labelled `out`, requiring one of them to be labelled `in`
$\pi^{\mathsf{P}}_{\mathsf{S}}$	$\{\top, \bot\}$	Alg. 13, 15	Selector between Algorithms 5 and 6 which exploits the knowledge that stable extensions are also preferred
$\pi^{\mathsf{P}}_{\mathsf{iO}}$	$\{\top, \bot\}$	Alg. 13, 15, 16	Further to Proposition 9, to identify a complete labelling which is strictly more committed w.r.t. a previously found complete labelling, an argument labelled `undec` should then become `out`

$\pi^{\mathsf{P}}_{\mathsf{iI}}$	$\{\top, \bot\}$	Alg. 13, 15, 16	Further to Proposition 9, to identify a complete labelling which is strictly more committed w.r.t. a previously found complete labelling, an argument labelled `undec` should then become `in`
$\pi^{\mathsf{P}}_{\mathsf{eO}}$	$\{\top, \bot\}$	Alg. 13, 15, 16	Further to Proposition 8, to identify a non-empty complete labelling that is not less committed w.r.t. any found preferred labelling, an argument labelled `in` should then become labelled `out`
$\pi^{\mathsf{P}}_{\mathsf{eI}}$	$\{\top, \bot\}$	Alg. 13, 15, 16	Further to Proposition 8, to identify a non-empty complete labelling that is not less committed w.r.t. any found preferred labelling, an argument labelled `out` should then become labelled `in`

Table B.5: Table summarising the parameters used in algorithms presented in Section 5.

Appendix C. Pilot study: performance comparison with minimal number of propositional variables

For stable semantics, the CNFs identified in Proposition 11 can be modified as presented in Proposition 12, by taking into consideration the constraint $\bigwedge_{\mathbf{a}\in\mathcal{A}} \neg U_{\mathbf{a}}$ (cf. Algorithm 10 for instance). In Proposition 12, we use one propositional variable $I_{\mathbf{a}}$ for argument $\mathbf{a} \in \mathcal{A}$, implicitly defining two labels: if $I_{\mathbf{a}}$ is assigned $\top$, $\mathbf{a}$ is `in`, otherwise it is `out`. Proof of Proposition 12 is omitted as it follows from straightforward manipulations.

Proposition 12. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$:

$$C^{\leftarrow}_{\mathtt{in}} \equiv \bigwedge_{\{\mathbf{a}\in\mathcal{A}\}} \left(I_{\mathbf{a}} \vee \left(\bigvee_{\{\mathbf{b} \ \mid \ \mathbf{b}\to\mathbf{a}\}} I_{\mathbf{b}} \right)\right)$$

$$C^{\rightarrow}_{\mathtt{in}} \equiv \bigwedge_{\{\mathbf{a}\in\mathcal{A}\}} \left(\bigwedge_{\{\mathbf{b} \ \mid \ \mathbf{b}\to\mathbf{a}\}} \neg I_{\mathbf{a}} \vee \neg I_{\mathbf{b}} \right)$$

	Mean runtime (s)		**Median runtime (s)**		**Wilcoxon**
	Prop. 12	Prop. 11	Prop. 12	Prop. 11	
`st_small`	27.91	75.93	7.97	6.02	Yes Z = -1.126 p = 0.260
`st_medium`	19.33	19.46	13.38	13.85	No Z = -2.057 p = 0.040

Table C.6: Statistics for the pilot study comparing computing EE-ST using three variables per argument (cf. Prop. 11) and using one variable per argument (cf. Prop. 12. The **Wilcoxon** column refers to Wilcoxon signed-rank tests aimed at assessing whether there is a statistical significant change in the execution time.

$$C^{\leftarrow}_{\texttt{out}} \equiv \bigwedge_{\{\mathbf{a}\in\mathcal{A}\}} \left(\bigwedge_{\{\mathbf{b} \ | \ \mathbf{b}\rightarrow\mathbf{a}\}} \neg I_{\mathbf{b}} \vee \neg I_{\mathbf{a}} \right)$$

$$C^{\rightarrow}_{\texttt{out}} \equiv \bigwedge_{\{\mathbf{a}\in\mathcal{A}\}} \left(I_{\mathbf{a}} \vee \left(\bigvee_{\{\mathbf{b} \ | \ \mathbf{b}\rightarrow\mathbf{a}\}} I_{\mathbf{b}} \right)\right)$$

In this pilot study, we considered a portion of the ICCMA-15 benchmark, namely the groups `st_small` and `st_medium`, which consist of graphs which feature many complete/preferred/stable extensions.[16] `st_small` features AFs with up to 300 arguments, while `st_medium` features AFs with 400 arguments.

We ran Algorithm 10 using $C^{\rightarrow}_{\texttt{in}} \wedge C^{\leftarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\leftarrow}_{\texttt{out}}$ with the CNFs specified in Proposition 11, and we compared that against the same algorithm using $C^{\rightarrow}_{\texttt{in}} \wedge C^{\leftarrow}_{\texttt{in}} \wedge C^{\rightarrow}_{\texttt{out}} \wedge C^{\leftarrow}_{\texttt{out}}$ with the CNFs specified in Proposition 12.[17] We used a 6 core machine Intel(R) Core(TM) i7-5820K CPU @3.30GHz with 8GB RAM.

Table C.6 summarises the statistics for this study. First of all, according to a Wilcoxon signed-rank test, there is a significant change in the execution time for AFs in `st_small`, but there is no significant change for the larger AFs in `st_medium`. Moreover, considering the AFs in `st_small`, the average runtime for the version using the CNFs of Proposition 11 is higher than for the version using the CNFs of Proposition 12, but this is reversed when considering median values. Indeed, the median runtime of the implementation running on the CNFs specified by Proposition 11 is smaller than the median runtime of the implementation running on the *minimal* CNFs specified by Proposition 12.

[16]`http://argumentationcompetition.org/2015/results.html`

[17]In this case, references to $U_{\mathbf{a}}$ in Algorithm 10 must be ignored.

Therefore, this pilot study seems to suggest that performance does not necessary improve when considering fewer propositional variables when transforming the constraints imposed by (stable) semantics into CNFs.