FernUniversität in Hagen
Faculty of Mathematics and Computer Science

AIG
Artificial Intelligence Group

Algorithmic Shortcuts for Skeptical Reasoning under Preferred Semantics in Abstract Argumentation

Bachelor's Thesis

in partial fulfilment of the requirements for
the degree of *Bachelor of Science (B.Sc.)*
in Informatik

submitted by
Julian Sander

First examiner: Prof. Dr. Matthias Thimm
Artificial Intelligence Group

Advisor: Lars Bengel
Artificial Intelligence Group

Statement

I declare that I have written the bachelor´s thesis independently and without unauthorized use of third parties. I have only used the indicated resources and I have clearly marked the passages taken verbatim or in the sense of these resources as such. The assurance of independent work also applies to any drawings, sketches or graphical representations. The work has not previously been submitted in the same or similar form to the same or another examination authority and has not been published. By submitting the electronic version of the final version of the bachelor´s thesis, I acknowledge that it will be checked by a plagiarism detection service to check for plagiarism and that it will be stored exclusively for examination purposes.

I explicitly agree to have this thesis published on the webpage of the artificial intelligence group and endorse its public availability.

Software created for this work has been made available as open source; a corresponding link to the sources is included in this work. The same applies to any research data.

Warendorf, 13.10.2025 Julian Sander Digital unterschrieben von Julian Sander Datum: 2025.10.13 10:43:30 +02'00'

(Place, Date) (Signature)

Zusammenfassung

Skeptische Akzeptanz unter Anwendung der bevorzugten Semantik ist eines der schwierigsten Schlussfolgerungsprobleme in der abstrakten Argumentation. Daher sind effiziente Lösungsverfahren für dieses Problem umso wichtiger. Diese Abschlussarbeit untersucht algorithmische Abkürzungen, mit denen dieses Schlussfolgerungsproblem in einigen Fällen gelöst werden kann und das mit deutlich weniger Ressourcen als von den bekannten vollständigen Algorithmen benötigt werden. Neben einen Überblick über den aktuellen Stand der Technik, besteht der hauptsächliche wissenschaftliche Beitrag dieser Arbeit in der Vorstellung neuer Abkürzungen zur Erweiterung der Anwendbarkeit solcher Techniken und einer Evaluierung aller Abkürzungen bezüglich ihres Potentials zur Steigerung der Laufzeiteffizienz bestehender Lösungsverfahren. Die Ergebnisse zeigen, dass die Laufzeiten von Lösungsverfahren unter Nutzung von algorithmischen Abkürzungen mehr als halbiert werden können.

Abstract

Skeptical acceptance under the preferred semantics is one of the most intricate reasoning problems in abstract argumentation. That is why efficient solving approaches are of utmost importance for this problem. This thesis investigates algorithmic shortcuts that can be used to solve the mentioned reasoning problem for some problem instances with considerable less resources than known complete algorithms. Besides an overview of the state of the art, the main scientific contribution of this thesis are the proposition of new shortcuts to improve the applicability of such techniques and an evaluation of all shortcuts regarding their potential to improve the runtime efficiency of an argumentation solver. The results show that the runtime of argumentation solvers can be more than halved by using algorithmic shortcuts.

Contents

1 Introduction

Argumentation is a fundamental cognitive capacity of human beings, which allows us to structure our reasoning and to handle incomplete or inconsistent information [2]. When facing a complex situation, we are able to abstract pieces of information as single arguments, which support or contradict one another. This structuring helps us to assess the plausibility of received information and to make sense of the situation at hand. We filter out coherent (also called *admissible* [3]) sets of those arguments, which can be seen as possible truths in the given situation. To develop rational agents, *formal argumentation* strives to emulate human argumentation in computer systems as part of an Artificial Intelligence (AI) [2, 15]. The theory of *abstract argumentation frameworks* is part of this research and focuses on modeling the conflicts of arguments, to answer the question which set of arguments prevails in these conflicts [2, 3, 15]. To come to a reasonable conclusion of a given situation it is important to assess the acceptance of an argument. Different notions of acceptance apply different degrees of skepticism, ranging from credulous to skeptical acceptance [2, 17]. An argument is seen as *skeptically accepted* if it holds in every sensible interpretation of the situation at hand. One way of interpreting such an argumentation framework is to apply the *preferred semantics*. This semantics extracts the different admissible sets of arguments of maximum size wrt. set inclusion from the framework. Those sets can be seen as the most comprehensive truths possible in the current situation. Proving that an argument is contained in all those truths renders the argument as skeptically accepted using the preferred semantics. This reasoning problem is denoted as DS-PR. Building upon these skeptically accepted arguments we can conclude a ground truth or common sense, which helps in further advancing the reasoning. It is therefor of great interest to conceive algorithms that allow us to decide whether an argument is skeptically accepted in a given argumentation framework or not. To promote the research on this question is the intention of the *International Competition on Computational Models of Argumentation (ICCMA)*[1]. ICCMA aims at developing computational models for several problems in abstract argumentation, including DS-PR.

The problem of DS-PR is highly intractable and is proven to be part of the class of Π_2^P-complete problems [16]. The use of NP-oracles is therefore a common practice in many state-of-the-art argumentation solvers to solve instances of DS-PR, see the submissions to ICCMA'23 [25] for examples. The questions asked are encoded as propositional formulas and answered by solving the Boolean Satisfiability problem (SAT) for these formulas [6, 7, 12]. A program to solve the SAT problem, called a SAT-solver, serves as the NP-oracle in these approaches. Most of the argumentation solvers follow the SAT-based Counter-Example-Guided Abstraction Refinement (CEGAR) approach [14]. The oracle is iteratively asked for an admissible set of arguments that could contradict the skeptical acceptance of the specified argument [12, 18, 28]. Note that with "NP-oracle" we describe a variant of a formal NP-oracle, which does not only answer YES or NO but also certifies each YES-answer returning a set of arguments. The algorithms ask the oracle for as long as (unvisited) sets exist, that have the potential to be counter-examples to the skeptical acceptance. The first implementations of these solvers, like the solvers CEGARTIX [18], ARGSEMSAT [12], or μ-TOKSIA [28], had to enumerate through all preferred sets of arguments in the worst case. Modern approaches like the Conflict-Driven Admissibility Search algorithm (CDAS) [32], used in the solver FUDGE [32], reduce the number of necessary questions to the orcale by focusing on those sets which are in conflict with the query argument. Obviously, making less calls to the NP-oracle has the prospect of increasing the efficiency of an

[1] https://argumentationcompetition.org/

approach. Having to iteratively consult the NP-oracle is time and resource consuming. That is why recent development in argumentation solvers tries to solve the problem of DS-PR without having to use these approaches. The solver REDUCTO [4] for example uses a two step approach. Iteratively consulting the NP-oracle is only the second step. In the first step the solver makes use of the context given by the argumentation framework to exploit shortcuts in solving the problem. One technique is to calculate a set of arguments using the grounded semantics. It is known that if the argument is contained in the grounded set of arguments, which can be computed in polynomial time [17], then the argument has to be skeptically accepted under the preferred semantics [13]. The use of such context information aims at increasing the efficiency of the solver. The applicability of these techniques is however limited. This means that there are argumentation frameworks in which these techniques alone do not solve the problem. For example if the argument is neither contained nor attacked by the grounded set of arguments, then we do not know if the argument is skeptically accepted or not. In these cases in which a solution cannot be found in the first step, REDUCTO uses an iterative consultation of the NP-oracle in the second step. Considering the motivation to use the techniques of the first step to improve the efficiency of the overall approach and their limited applicability, we call such techniques *algorithmic shortcuts*.

The use of the shortcuts described above is not always beneficial. One well-known example from the literature is the argumentation solver μ-TOKSIA. The solver uses an almost identical algorithm like the argumentation solver CEGARTIX to solve the reasoning problem DS-PR [18]. The only algorithmic difference between the two solvers is that μ-TOKSIA does not apply any shortcuts before starting the iterative consultation of the NP-oracle following the CEGAR approach. μ-TOKSIA has been developed based on the experiences with CEGARTIX several years after CEGARTIX first publication. Even if we take into account that μ-TOKSIA uses much more advanced components, like a more efficient SAT-solver as the NP-oracle, the huge increase in efficiency of μ-TOKSIA compared to CEGARTIX opens an interesting question: Do the shortcuts applied in CEGARTIX really improve the efficiency of the solver? Or are the additional computations necessary because of these shortcuts actually slowing the solver down compared to the approach of μ-TOKSIA? To assess the effects of algorithmic shortcuts on the efficiency of the solving process for the reasoning problem DS-PR is the overarching goal of this thesis.

We will summarize the shortcuts known from the literature and propose some new shortcuts. These new shortcuts can be computed with at most two calls to the NP-oracle, which we will demonstrate with this thesis. The thesis shows how much and if these shortcuts can increase the efficiency of an argumentation solver. We will furthermore investigate the scope of the applicability of these shortcuts, which leads to the following questions for each of the proposed algorithmic shortcuts:

- RQ1 How many times is the algorithmic shortcut able to solve a problem instance of the used datasets?
- RQ2 How much time does it take for the algorithmic shortcut to return a result for a problem instance of the used datasets?
- RQ3 How much faster can the algorithmic shortcut solve a problem instance of the used datasets compared to known complete algorithms from the literature?

This thesis is structured in **6 sections**. In **Section 2** we will explain the theoretical backgrounds which are necessary for the reader to understand the proposed approach. We will highlight the most relevant related works in the field of argumentation solvers in **Section 3**. It follows the description of different shortcuts in **Section 4**. We will furthermore discuss aspects of their implementation as single programs in this section. In **Section 5** we discuss the

setup, conduct, and the results of an empirical evaluation of the algorithmic shortcuts. **Section 6** concludes this thesis.

2 Preliminaries

In this section we introduce the basic concepts which are used in this thesis. In **Section** 2.1 we define the notion of abstract argumentation frameworks and some semantics used to interpret such frameworks. We will then explain in **Section** 2.2 how we can reduce the problem of finding admissible sets of arguments in an abstract argumentation framework to the problem of Boolean Satisfiability of a propositional formula.

2.1 Abstract Argumentation

The following introduction of the basic concepts of abstract argumentation is inspired by Baroni et al. [3] and Thimm et al. [32].

Let us at first introduce the notion of an abstract *argumentation framework*, first proposed by Dung [15]. Such a framework is a directed graph, in which each node represents an argument and each directed edge represents an attack between arguments. We say that argument a attacks argument b, iff argument a contradicts argument b.

Definition 1. *An argumentation framework is a tuple* $\mathsf{AF} = (A, R)$ *in which* A *is a finite set of arguments and* R *is a relation* $R \subseteq A \times A$*.*

The relation R describes the attacks between arguments. $(\mathtt{a}, \mathtt{b}) \in R$ means that argument a attacks argument b, see Example 1.

Example 1. *Let* $\mathsf{AF}_1 = (A, R)$ *be an argumentation framework with* $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ *and* $R = \{(\mathtt{a}, \mathtt{b}), (\mathtt{b}, \mathtt{c})\}$*, see Figure 1. The set* A *contains the arguments* $\mathtt{a}, \mathtt{b}, \mathtt{c}$*. The attack relation* R *describes argument* a *attacking argument* b *and argument* b *attacking argument* c*.*

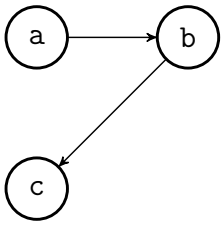

Figure 1: [3] The argumentation framework $\mathsf{AF}_1 = (A, R)$ with $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $R = \{(\mathtt{a}, \mathtt{b}), (\mathtt{b}, \mathtt{c})\}$

If an argument a attacks a member of a set S of arguments, we say that a attacks S and vice versa if a member of S is the attacker. To ease our work with sets of arguments, we define the set of arguments (S^+) attacked by a set S, as well as the set of arguments (S^-) attacking S in the following definition:

Definition 2. *Let* $\mathsf{AF} = (A, R)$ *be an argumentation framework and* $S \subseteq A$ *be a set of arguments.*
$S^+ = \{a \in A \mid \exists b \in S, (b, a) \in R\}$ $\quad$ $S^- = \{a \in A \mid \exists b \in S, (a, b) \in R)\}$

Admissibility The argumentation framework is representing the conflicts between the arguments in a way that enables further analysis about which sets of arguments prevail in these conflicts. A fundamental property of such prevailing sets of arguments is *Admissibility*. At first we note that arguments can only coexist in an admissible set of prevailing arguments if there is no conflict between them, in other words if the set is *conflict-free*. Otherwise the conflict would render at least one of the arguments as defeated. Furthermore, admissible sets of arguments need to *defend* their members against any attackers. This means that each attacker of the set needs to be attacked by at least one member of the set.

Definition 3. *Let $\mathsf{AF} = (A, R)$ be an argumentation framework.*

- *A set $S \subseteq A$ is **conflict-free** if for all $a, b \in S$ it holds that $(a, b) \notin R$, which means that $S \cap S^+ = \emptyset$.*
- *A set $S \subseteq A$ **defends** an argument $a \in A$ if for all $b \in A$ with $(b, a) \in R$ there is $c \in S$ with $(c, b) \in R$, i.e. if $\{a\}^- \subseteq S^+$. The function $F_{\mathsf{AF}} : 2^A \to 2^A$ such that $F_{\mathsf{AF}}(S) = \{a \mid S \text{ defends } a\}$ is called the characteristic function of AF.*
- *A set $S \subseteq A$ is called an **admissible set** iff S is conflict-free and $S \subseteq F_{\mathsf{AF}}(S)$.*

Example 2. *Let $\mathsf{AF}_2 = (A, R)$ be an argumentation framework as depicted in Figure 2. The admissible sets of this framework are $E_{\sigma_{ad}}(\mathsf{AF}_2) = \{\{\mathtt{d}\}, \{\mathtt{a}\}, \{\mathtt{a}, \mathtt{d}\}\}$. $\{\mathtt{d}\}$ is an admissible set because it is unattacked.* `a` *defends itself, which renders $\{\mathtt{a}\}$ to be an admissible set. The combination of both sets $\{\mathtt{a}, \mathtt{d}\}$ is again an admissible set.*

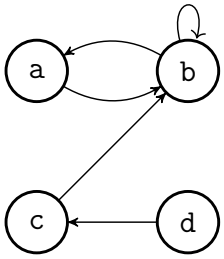

Figure 2: The argumentation framework AF_2.

The last point of Example 2 leads us directly to Dung's Fundamental Lemma [15]:

Lemma 1. *[3] For any argumentation framework $\mathsf{AF} = (A, R)$, let S be an admissible set and a, b be arguments defended by S. Then*

1. *$S' = S \cup \{a\}$ is an admissible set;*
2. *b is defended by S'*

We can conclude from Dung's Fundamental Lemma that the union of admissible sets is itself admissible, if there is no conflict between those sets.

Argumentation Semantics The answer to the question which arguments prevail in an argumentation framework is determined by the argumentation semantics used to interpret these conflicts. One way of representing which arguments prevail in the conflicts of an argumentation framework is to create sets of those arguments. Since there can be situations in which different outcomes of the conflicts are possible, each set represents one of those possible outcomes. A set of prevailing arguments defined by an argumentation semantics is called an *extension*.

Definition 4. *An extension-based semantics σ associates with any argumentation framework $\mathsf{AF} = (A, R)$ a subset of 2^A, denoted as $\mathcal{E}_\sigma(\mathsf{AF})$.*

While there are many different argumentation semantics, only three of them are of interest for the work of this thesis. These are the so called *complete*, *grounded*, and *preferred* semantics.

Dung's Fundamental Lemma naturally leads to the idea of extending admissible sets to make the basis of justification of an argument more comprehensive, which can be seen as more convincing. This is the intuition of the *complete semantics*, where all arguments defended by a set should be contained in the set. If we accept the complete extensions as standpoints with adequate justification in a formal argumentation, then we can ask for a common ground of these standpoints, e.g. the intersection of those sets. Arguments contained in this intersection do always prevail, which means that they cannot be rejected. It is therefore a ground truth, which leads us to the intuition of the *grounded semantics*. The grounded extension can be seen as a standpoint with minimal commitment. It refrains from making statements about an argument if there are only the slightest doubts that the argument could not prevail in one interpretation of the framework. Arguments for which there are good reasons to be accepted or rejected in at least some interpretations might be left as undecided by this standpoint. To include these aspects and to get a more comprehensive interpretation of an argumentation framework is the intuition of the *preferred semantics*. This semantics is based on the complete semantics and aims at maximizing the number of accepted arguments. Alternatively we can define a preferred extension as an admissible set of arguments of maximum size wrt. to set inclusion.

Definition 5. *Let $\mathsf{AF} = (A, R)$ be an argumentation framework.*

- *A set $S \subseteq A$ is called a **complete** extension iff S is conflict-free and $S = F_{\mathsf{AF}}(S)$.*
- *The **grounded** extension $\mathtt{Ext}_{\mathtt{GR}}$ of AF is a minimal (w.r.t. set inclusion) complete extension of AF.*
- *A set $S \subseteq A$ is called a **preferred** extension iff S is a maximal (w.r.t. set-inclusion) complete extension of AF.*

Following these definitions we denote the set of all complete extensions of an argumentation framework AF as $\mathcal{E}_{\mathsf{CO}}(\mathsf{AF})$. The set of all preferred extensions is denoted as $\mathcal{E}_{\mathsf{PR}}(\mathsf{AF})$. Because of its minimality requirement, the grounded extension of an argumentation framework is unique.

Example 3. *Let $\mathsf{AF}_2 = (A, R)$ be an argumentation framework as depicted in Figure 2. Furthermore let $\mathcal{E}_{ad}(\mathsf{AF}_2)$ be the set of all admissible sets and $\mathcal{E}_{CO}(\mathsf{AF}_2)$ be the set of all complete extensions of AF_2. It follows that $\mathcal{E}_{ad}(\mathsf{AF}_2) = \{\{\mathtt{a}\}, \{\mathtt{d}\}, \{\mathtt{a}, \mathtt{d}\}\}$ and $\mathcal{E}_{CO}(\mathsf{AF}_2) = \{\{\mathtt{d}\}, \{\mathtt{a}, \mathtt{d}\}\}$. Note that $\{\mathtt{a}\}$ is an admissible set, but it is not a complete extension, because $\mathtt{d}$ is an unattacked argument and is therefore defended by every set of arguments (including the empty set). Thus we have to add $\mathtt{d}$ to the set $\{\mathtt{a}\}$ to get a complete extension. As $\mathtt{d}$ is contained in all complete extensions, it is also contained in the grounded extension. $\mathtt{d}$ is the only argument with this property. Therefore, the grounded extension is $\mathtt{Ext}_{\mathtt{GR}}(\mathsf{AF}_2) = \{\mathtt{d}\}$. Looking at $\mathcal{E}_{CO}(\mathsf{AF})$ we can see that there is only one preferred extension $\mathcal{E}_{PR}(\mathsf{AF}_2) = \{\{\mathtt{a}, \mathtt{d}\}\}$.*

From Definition 5 we can conclude that for any argumentation framework AF it holds that

$$\mathcal{E}_{\mathsf{PR}}(\mathsf{AF}) \subseteq \mathcal{E}_{\mathsf{CO}}(\mathsf{AF}) \subseteq \mathcal{E}_{\mathsf{ad}}(\mathsf{AF}) \tag{1}$$

The grounded extension is included in each preferred extension because it is included in each complete extension. However, the intersection of all preferred extensions can include more arguments than the grounded extension. Furthermore can a framework have more than one preferred extension. Both points are illustrated in the following example.

Example 4. *Let $\mathsf{AF}_3 = (A, R)$ be an argumentation framework as depicted in Figure 3. The complete extensions of this framework are $\mathcal{E}_{CO}(\mathsf{AF}_3) = \{\emptyset, \{\mathtt{a}, \mathtt{d}\}, \{\mathtt{b}, \mathtt{d}\}\}$. Hence, the preferred extensions of this framework are $\mathcal{E}_{PR}(\mathsf{AF}_3) = \{\{\mathtt{a}, \mathtt{d}\}, \{\mathtt{b}, \mathtt{d}\}\}$. While* d *is contained in each preferred extension, the grounded extension of this framework is the empty set.*

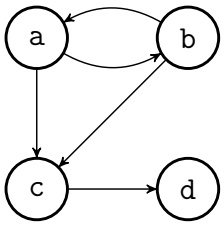

Figure 3: [3] The argumentation framework AF_3.

Labelings Another way of representing the prevailing arguments of an argumentation framework is to use *labelings*. In this representation the semantics gives each argument a label, denoting if the argument prevails or not.

Definition 6. *[11] Let $\mathsf{AF} = (A, R)$ be an argumentation framework and Λ be a set of labels. A Λ-labeling is a total function* $\mathtt{Lab} : A \to \Lambda$. *The set of all Λ-labelings of AF is denoted as $\mathfrak{L}(\Lambda, \mathsf{AF})$.*

While different labels are conceivable, it is common to use a set of 3 different labels {IN ,OUT ,UNDEC } [3]. An argument labeled IN prevails in the conflicts according to the interpretation of the used semantics. An argument labeled OUT does not prevail in the conflicts. Arguments that are labeled UNDEC are neither known to prevail nor to not to. UNDEC (short for "undecided") describes a situation in which there are reasons for either way (prevail or not), thus no final verdict can be drawn.

To ease our use of labelings we define:

Definition 7. *Let $\mathsf{AF} = (A, R)$ be an argumentation framework and* Lab *be a labeling.*

- $\mathtt{in}(\mathtt{Lab}) = \{a \mid \mathtt{Lab}(a) = \mathsf{IN}\}$
- $\mathtt{out}(\mathtt{Lab}) = \{a \mid \mathtt{Lab}(a) = \mathsf{OUT}\}$
- $\mathtt{undec}(\mathtt{Lab}) = \{a \mid \mathtt{Lab}(a) = \mathsf{UNDEC}\}$

We can describe Lab *with the tuple* $\{\mathtt{in}(\mathtt{Lab}), \mathtt{out}(\mathtt{Lab}), \mathtt{undec}(\mathtt{Lab})\}$

To decide which label to put on an argument, we use a labeling-based argumentation semantics.

Definition 8. *Given an argumentation framework $\mathsf{AF} = (A, R)$ and a set of labels Λ, a labeling-based semantics σ associates with AF a subset of $\mathfrak{L}(\Lambda, \mathsf{AF})$, denoted as $\mathcal{L}_\sigma(\mathsf{AF})$.*

We can define all mentioned extension-based argumentation semantics, as well as the property of admissibility, using labelings. For these definitions we need to define three properties about the legitimacy of a label, e.g. *legally IN*, *legally OUT*, and *legally UNDEC*.

Definition 9. *Let* Lab *be a labeling of an argumentation framework $\mathsf{AF} = (A, R)$.*

- *An* IN*-labeled argument is said to be legally* IN *iff all its attackers are labeled* OUT*.*
- *An* OUT*-labeled argument is said to be legally* OUT *iff it has at least one attacker that is labeled* IN*.*
- *An* UNDEC*-labeled argument is said to be legally* UNDEC *iff not all its attackers are labeled* OUT *and it doesn't have an attacker that is labeled* IN*.*

With these properties we can now define labeling-based notions of admissibility and the argumentation semantics.

Definition 10. *Let* $AF = (A, R)$ *be an argumentation framework.*

- *An* ***admissible labeling*** *is a labeling* Lab *where each* IN*-labeled argument is legally* IN *and each* OUT*-labeled argument is legally* OUT*.*
- *A* ***complete labeling*** *is an admissible labeling where every* UNDEC *labeled argument is legally* UNDEC*.*
- *The* ***grounded labeling*** $\texttt{Lab}_{\texttt{GR}}$ *of* AF *is a complete labeling* Lab *where* $\texttt{in}(\texttt{Lab})$ *is minimal (w.r.t. set inclusion), i.e. there is no complete labeling* Lab′ *such that* $\texttt{in}(\texttt{Lab}') \subsetneq \texttt{in}(\texttt{Lab})$*.*
- *A* ***preferred labeling*** *of* AF *is a complete labeling* Lab *where* $\texttt{in}(\texttt{Lab})$ *is maximal (w.r.t. set-inclusion) among all complete labelings, i.e. there is no complete labeling* Lab′ *such that* $\texttt{in}(\texttt{Lab}') \supsetneq \texttt{in}(\texttt{Lab})$*.*

Example 5. *Let* $AF_2 = (A, R)$ *be an argumentation framework as depicted in Figure 2 and* $\mathcal{L}_{ad}(AF_2)$ *be the set of admissible labelings.* $\mathcal{L}_{ad}(AF_2)$ *contains the labelings* $\texttt{Lab}_1 = \{\{\texttt{d}\}, \{\texttt{c}\}, \{\texttt{a}, \texttt{b}\}\}$, $\texttt{Lab}_2 = \{\{\texttt{a}\}, \{\texttt{b}\}, \{\texttt{c}, \texttt{d}\}\}$, *and* $\texttt{Lab}_3 = \{\{\texttt{a}, \texttt{d}\}, \{\texttt{b}, \texttt{c}\}, \emptyset\}$*. In* $\texttt{Lab}_1$ *the argument* d *is legally labeled* IN*, because it is unattacked. Since* d *is labeled* IN*, the argument* c *is legally labeled* OUT *in* $\texttt{Lab}_1$*. Arguments* a *and* b *are left undecided in* $\texttt{Lab}_1$*. As labeling* $\texttt{Lab}_2$ *shows, it is also legal to label* a *as* IN*, because if* a *is labeled* IN *all of its attackers, which is only* b*, are labeled* OUT*. In this case the arguments* c *and* d *are left undecided. Argument* a *can also coexist with* d*, because there is no attack relation between the two. Therefore an admissible interpretation of* AF_2 *is to label both to* IN*, which means that the arguments* b *and* c *are legally* OUT*, leaving the set of undecided arguments empty. Regarding complete labelings we note that* $\texttt{Lab}_1$ *and* $\texttt{Lab}_3$ *are complete labelings, because* a*,* b *and* c *can be legally labeled* UNDEC*.* $\texttt{Lab}_2$ *however is not a complete labeling. Since* d *has no attackers it holds that all its attackers are labeled* OUT*. Therefore* d *is not legally* UNDEC*, which shows that* $\texttt{Lab}_2$ *is not a complete labeling.*

Note that given a labeling of a framework, we can always deduce its corresponding extension by collecting all arguments labeled IN in one set [3].

Acceptance Problems It should be noted that semantics do not directly assess the acceptance of an argument. In most cases a semantics will return multiple alternative labelings or extensions for a given argumentation framework. Those different labelings can associate different labels for the same argument, as for argument a in Example 5. To clarify when an argument is accepted and when it is not, there are different notions of acceptance based on the level of skepticism demanded. A very skeptical viewpoint would demand the argument to be labeled IN in every labeling associated by the semantics σ used. On the other side of the spectrum, a very credulous viewpoint accepts an argument if it is labeled IN in at least one labeling of σ. The reasoning problem of skeptical acceptance using a specific argumentation semantics σ is denoted as DS-σ. The problem of credulous acceptance using σ is denoted as DC-σ. In this thesis, we are only interested in skeptical acceptance, for which we introduce a formal definition.

As these definitions of acceptance make only sense if there are defined over a non-empty set of labelings or extensions, we start by defining the set of frameworks which fulfill this precondition:

Definition 11. *Given a labeling-based semantics σ, $\mathcal{DL}_\sigma = \{AF \mid \mathcal{L}(AF) \neq \emptyset\}$. Given an extension-based semantics σ, $\mathcal{D}_\sigma = \{AF \mid \mathcal{E}_\sigma(AF) \neq \emptyset\}$.*

With this definition we can know define skeptical acceptance in case of a labeling-based semantics.

Definition 12. *Given a labeling-based semantics σ and an argument framework $AF \in \mathcal{DL}_\sigma$, an argument a is skeptically accepted, iff for all labelings* Lab *in $\mathcal{L}_\sigma(AF)$ it holds that* $\texttt{Lab}(a) = IN$.

In case that we worked with an extension-based semantics we define

Definition 13. *Given an extension-based semantics σ and an argumentation framework $AF \in \mathcal{DE}_\sigma$, an argument a is skeptically accepted, iff for all extensions E in $\mathcal{E}_\sigma(AF)$ it holds that $a \in E$.*

Example 6. *Let $AF_4 = (A, R)$ be an argumentation framework as depicted in Figure 4. Looking at the complete extensions $\mathcal{E}_{CO}(AF_4) = \big\{\{\texttt{a}\}, \{\texttt{a}, \texttt{c}\}, \{\texttt{a}, \texttt{d}\}\big\}$ we see that only argument* a *is skeptically accepted under the complete semantics, since it is contained in each of the discussed extensions.*

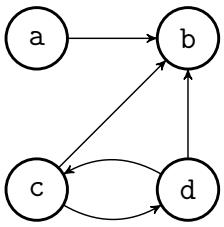

Figure 4: The argumentation framework AF_4.

To simplify our work on the reasoning problem of skeptical acceptance we define the notion of a problem instance.

Definition 14. *Let $AF = (A, R)$ be an argumentation framework. We define a problem instance of the acceptance reasoning problem as a tuple $(AF, \mathbf{a})$ of an argumentation framework AF and an argument $\mathbf{a}$ with $\mathbf{a} \in A$. The argument $\mathbf{a}$ is called the query argument of the problem instance.*

In summary, in this section we introduced argumentation frameworks and some basic semantics to interpret such frameworks, notably the complete, the grounded, and the preferred argumentation semantics. We showed how these semantics can be expressed both as extensions and as labelings. After defining the notion of skeptical acceptance, we now have a clear understanding of the reasoning problem of skeptical acceptance under the preferred semantics.

2.2 Reduction to the Boolean Satisfiability Problem

In this section we briefly recall the basics of the Propositional Logic (PL) and the Boolean Satisfiability Problem of the PL, before we explain how searching for an admissible set in an argumentation framework can be reduced to this problem. After that, we discuss the encoding needed to calculate complete extensions. This section is inspired by Cerutti et al. [12].

There are different ways to encode an argumentation framework as a propositional formula. We will use the encoding proposed by Cerutti et al. [12], which encodes labelings of arguments using two propositional constants for each argument. Other encodings like the one from Besnard and Doutre [7] encode sets of arguments (extensions) and use only one propositional constant per argument. The two-constant encoding has proven to be more efficient [12] and more importantly allows a more fine-grained control over the status of the arguments. Recall that an argument that is not contained in an extension can either be labeled OUT or UNDEC. This subtle difference is of interest for the shortcuts proposed in this thesis.

Let $\mathsf{AF} = (A, R)$ be an argumentation framework. We use a propositional language in which each argument a with $\mathtt{a} \in A$ is represented as two propositional constants $I_{\mathtt{a}}$ and $O_{\mathtt{a}}$. Using these constants we define propositional formulas applying the connectives: $\neg$, $\wedge$, $\vee$. Let φ be such a propositional formula. A truth assignment that evaluates the formula φ to TRUE is called a *model* of the formula φ. The universe of discourse of the proposed language are the labels of the arguments in the argumentation framework. If $I_{\mathtt{a}}$ is TRUE and $O_{\mathtt{a}}$ is FALSE, then the associated label of argument a is $\mathtt{Lab}(\mathtt{a}) = \mathsf{IN}$. Vice versa, if $I_{\mathtt{a}}$ is FALSE and $O_{\mathtt{a}}$ is TRUE, then $\mathtt{Lab}(\mathtt{a}) = \mathsf{OUT}$. If both propositional constants are FALSE, then the label UNDEC is associated with the argument a.

Example 7. *Let $\mathsf{AF}_5 = (A, R)$ be an argumentation framework as depicted in Figure 5 and $\mathcal{L}_p$ be the propositional language to encode the requirements of a labeling in this framework. $\mathcal{L}_p$ contains the propositional constants $I_{\mathtt{a}}, O_{\mathtt{a}}, I_{\mathtt{b}}, O_{\mathtt{b}}, I_{\mathtt{c}}$, and $O_{\mathtt{c}}$*

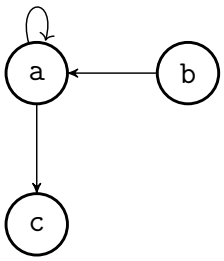

Figure 5: The argumentation framework AF_5.

Let us recall the general definition of a propositional formula being satisfiable:

Definition 15. *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.

To check if such an assignment exists is described as the *Boolean Satisfiability problem (*SAT*)* of the Propositional Logic. Programs that are able to solve SAT for a given formula are called SAT-*solvers*. A common format as input for SAT-solvers is a propositional formula in *Conjunctive Normal Form (CNF)*. A propositional formula in CNF is combining several *clauses* with the conjunctive operator ($\wedge$) of the PL. Each clause is a disjunction ($\vee$) of several *literals*. A literal is either a propositional constant of the used propositional language or the negation ($\neg$) of a propositional constant. Instead of solving the SAT problem by just answering YES or NO, most state-of-the-art SAT-solvers are able to return a model of the formula [22]. While a propositional formula can have several models, SAT-solvers return only one model used to certify the satisfiability of the formula. To enumerate all models of a propositional formula is known as the ALLSAT-Problem [21].

Encoding Admissible Labelings In order to represent a consistent labeling, it must be prevented that both constants ($I_{\mathtt{a}}$ and $O_{\mathtt{a}}$) are TRUE in the same model. We use the following formula to exclude these inaccurate representations:

$$\Psi_{consistent} = \bigwedge_{a \in A} \Big(\neg I_a \vee \neg O_a \Big) \tag{2}$$

All models of Formula (2) set at least one of the two propositional constants associated with each argument to FALSE.

Example 8. *Let $\mathsf{AF}_5 = (A, R)$ be an argumentation framework as depicted in Figure 5 and $\mathcal{L}_p$ be the propositional language to encode the requirements of a labeling in this framework. We define the propositional formula $\Psi_{consistent} = (\neg I_{\mathtt{a}} \vee \neg O_{\mathtt{a}}) \wedge (\neg I_{\mathtt{b}} \vee \neg O_{\mathtt{b}}) \wedge (\neg I_{\mathtt{c}} \vee \neg O_{\mathtt{c}})$. $\Psi_{consistent}$ has 8 different models, each model is a truth assignment in which for each argument $\mathtt{x} \in A$ either $I_{\mathtt{x}}$ or $O_{\mathtt{x}}$ or both are set to* FALSE. *Hence all models can be interpreted as consistent labelings for the arguments in A.*

We use $\mathcal{L}_p$ to define propositional formulas for labelings with different properties. To associate a model of a propositional formula with the labeling of an admissible set, we need to formulate the requirements that arguments have to be legally IN or legally OUT, see Definition 10. At first we define a propositional formula which is evaluated to TRUE if all IN labeled arguments of an assignment are legally IN. We recall from Definition 9 that legally IN means that if an argument is labeled IN, then all of its attackers have to be labeled OUT.

$$\Psi_{LegalIn} = \bigwedge_{(a \in A)} \Big(\bigwedge_{(b \in \{a\}^-)} \neg I_a \vee O_b \Big) \tag{3}$$

The clauses of Formula (3) are either TRUE because the argument a is not labeled IN or (in case a is labeled IN) because every attacker in $\{a\}^-$ is labeled OUT.

Example 9. *Let $\mathsf{AF}_5 = (A, R)$ be an argumentation framework as depicted in Figure 5 and $\mathcal{L}_p$ be the propositional language to encode the requirements of a labeling in this framework. We define the propositional formula $\Psi_{LegalIn} = (\neg I_{\mathtt{a}} \vee O_{\mathtt{a}}) \wedge (\neg I_{\mathtt{a}} \vee O_{\mathtt{b}}) \wedge (\neg I_{\mathtt{c}} \vee O_{\mathtt{a}})$. Note that there is no clause for the argument* b. *Argument* b *has no attackers, therefore the second conjunction of $\Psi_{LegalIn}$ iterates over the empty set. Furthermore, it should be noted that if we were to combine $\Psi_{consistent}$ of Example 8 with $\Psi_{LegalIn}$ by conjunction, then the resulting formula can only be* TRUE *iff the argument* a *is not labeled IN, which is reasonable regarding the argument attacking itself.*

The next propositional formula ensures that all arguments that are labeled OUT in a model are legally OUT. Following Definition 9, an argument is legally OUT iff it has at least one attacker that is labeled IN.

$$\Psi_{LegalOut} = \bigwedge_{a \in A} \Big(\neg O_a \vee \bigvee_{b \in \{a\}^-} I_b \Big) \tag{4}$$

Formula (4) accepts models only if for each argument a in the argumentation framework the argument is either not labeled OUT or (in case a is labeled OUT) if at least one of the attackers in $\{a\}^-$ is labeled IN.

Example 10. *Let $\mathsf{AF}_5 = (A, R)$ be an argumentation framework as depicted in Figure 5 and $\mathcal{L}_p$ be the propositional language to encode the requirements of a labeling in this framework. We define the*

propositional formula $\Psi_{LegalOut} = (\neg O_{\mathtt{a}} \vee I_{\mathtt{a}} \vee I_{\mathtt{b}}) \wedge (\neg O_{\mathtt{b}}) \wedge (\neg O_{\mathtt{c}} \vee I_{\mathtt{a}})$*. We can see from the second clause of this formula that* b *cannot be labeled* OUT *by any model of the formula. It represents the fact that there is no reason to label an unattacked argument as* OUT*. This does not imply however that* b *has to be labeled* IN*, as it can still be labeled* UNDEC*.*

We have now defined all formulas needed to encode an admissible set. However, recall that by definition the empty set is an admissible set, see Definition 3. Hence it would be a trivial solution to label all arguments as UNDEC. To solve the problem of DS-PR we are usually interested in non-empty admissible sets. That is why we have to add another formula so that its models associate to non-empty sets:

$$\Psi_{non-empty} = \bigvee_{a \in A} I_a \tag{5}$$

In all models of Formula (5) at least one argument is labeled IN.

Finally we can give a formula characterizing the non-empty admissible sets of an argumentation framework, cf. [12].

$$\Psi_{\mathsf{ad}} = \Psi_{consistent} \wedge \Psi_{LegalIn} \wedge \Psi_{LegalOut} \wedge \Psi_{non-empty} \tag{6}$$

Example 11. *Let* $\mathsf{AF}_5 = (A, R)$ *be an argumentation framework as depicted in Figure 5 and* $\mathcal{L}_p$ *be the propositional language to encode the requirements of a labeling in this framework. We define the propositional formula* $\Psi_{\mathsf{ad}} = \Psi_{consistent} \wedge \Psi_{LegalIn} \wedge \Psi_{LegalOut} \wedge (I_{\mathtt{a}} \vee I_{\mathtt{b}} \vee I_{\mathtt{c}})$ *using* $\Psi_{consistent}$*,* $\Psi_{LegalIn}$*, and* $\Psi_{LegalOut}$ *from Example 8, 9, and 10. As seen in Example 9* Ψ_{ad} *can only evaluate to* TRUE *iff* a *is not labeled* IN*. With* $\neg I_{\mathtt{a}}$ *being* TRUE*, it follows from the third clause of* $\Psi_{LegalOut}$ $(\neg O_{\mathtt{c}} \vee I_{\mathtt{a}})$ *that* c *cannot be labeled* OUT *in a model of* Ψ_{ad}*. As we have seen in Example 10* b *cannot be labeled* OUT *either (second clause of* $\Psi_{LegalOut}$ $(\neg O_{\mathtt{b}})$*). We can also deduce from* Ψ_{ad} *that* b *has to be labeled* IN *in order for* Ψ_{ad} *to evaluate to* TRUE*. We show this by a short contradiction: Let us assume there exists a model of* Ψ_{ad} *in which* b *is labeled* UNDEC*. Then the first clause of* $\Psi_{LegalOut}$ $(\neg O_{\mathtt{a}} \vee I_{\mathtt{a}} \vee I_{\mathtt{b}})$ *implies that* $\neg O_{\mathtt{a}}$ *has to hold, since* a *cannot be labeled* IN*. Consulting the third clause of* $\Psi_{LegalIn}$ $(\neg I_{\mathtt{c}} \vee O_{\mathtt{a}})$ *we would than deduce that* $\neg I_{\mathtt{c}}$ *has to be* TRUE*. At this point all three arguments would be labeled* UNDEC *which would render the last clause of* Ψ_{ad} $(I_{\mathtt{a}} \vee I_{\mathtt{b}} \vee I_{\mathtt{c}})$ *as* FALSE*, which would contradict our assumption that this truth assignment were a model of* Ψ_{ad}*. Hence all models of* Ψ_{ad} *have to label* b *as* IN*, and can label* c *as* IN *or* UNDEC*, what complies to the admissible sets* $\mathcal{E}_{\mathsf{ad}}(\mathsf{AF}_5) = \{\{\mathtt{b}\}, \{\mathtt{b}, \mathtt{c}\}\}$*.*

Encoding Complete Labelings For the complete semantics, the UNDEC labeled arguments have to be legally UNDEC. We recall from Definition 9 that legally UNDEC means that not all attackers of an argument are labeled OUT and there is no attacker that is labeled IN. We describe the first part of this requirement in the following formula:

$$\Psi_{LegalUndecOut} = \bigwedge_{a \in A} \left(I_a \vee \left(\bigvee_{b \in \{a\}^-} (\neg O_b) \right) \right) \tag{7}$$

In all truth assignments that evaluate Formula (7) to TRUE it holds that if an argument a is not labeled IN, then at least one of its attackers is not labeled OUT. Note that for arguments which are labeled legally OUT this formula is TRUE because at least one of their attacker is labeled IN, hence this attacker is not labeled OUT.

Example 12. *Let* $\mathsf{AF}_5 = (A, R)$ *be an argumentation framework as depicted in Figure 5 and* $\mathcal{L}_p$ *be the propositional language to encode the requirements of a labeling in this framework. We define the*

propositional formula $\Psi_{LegalUndecOut} = (I_{\mathtt{a}} \vee \neg O_{\mathtt{a}} \vee \neg O_{\mathtt{b}}) \wedge (I_{\mathtt{b}}) \wedge (I_{\mathtt{c}} \vee \neg O_{\mathtt{a}})$. *We clearly see that* b *has to be labeled* IN *to evaluate this formula to* TRUE. *In contrast to the encoding for an admissible set in Example 10 labeling* b *to* UNDEC *does not evaluate* $\Psi_{LegalUndecOut}$ *to* TRUE, *which complies with the tighter restrictions of the definition of complete labelings in relation to the definition of an admissible labeling.*

The second part of the definition of legally UNDEC labels can be described by the following formula:

$$\Psi_{LegalUndecIn} = \bigwedge_{a \in A} \left(\bigwedge_{b \in \{a\}^-} \neg I_b \vee O_a \right) \tag{8}$$

For all models of Formula (8) it holds that if an argument is labeled to IN or UNDEC, then none of its attackers is labeled IN, which is compliant with the definition of legally IN and legally UNDEC.

Example 13. *Let* $AF_5 = (A, R)$ *be an argumentation framework as depicted in Figure 5 and* $\mathcal{L}_p$ *be the propositional language to encode the requirements of a labeling in this framework. We define the propositional formula* $\Psi_{LegalUndecIn} = ((\neg I_{\mathtt{a}} \vee O_{\mathtt{a}}) \wedge (\neg I_{\mathtt{b}} \vee O_{\mathtt{a}}) \wedge (\neg I_{\mathtt{a}} \vee O_{\mathtt{c}}))$.

Following Definition 10 we can now give a formula that associates with a complete extension, cf. [12]:

$$\Psi_{\mathsf{CO}} = \Psi_{\mathsf{ad}} \wedge \Psi_{LegalUndecOut} \wedge \Psi_{LegalUndecIn} \tag{9}$$

Note that we are again only interested in non-empty complete sets, which is why we can use Ψ_{ad} without any modifications.

Example 14. *Let* $AF_5 = (A, R)$ *be an argumentation framework as depicted in Figure 5 and* $\mathcal{L}_p$ *be the propositional language to encode the requirements of a labeling in this framework. We define the propositional formula* $\Psi_{CO} = \Psi_{ad} \wedge \Psi_{LegalUndecOut} \wedge \Psi_{LegalUndecIn}$, *using* Ψ_{ad}, $\Psi_{LegalUndecOut}$, *and* $\Psi_{LegalUndecIn}$ *from Example 11, 12, and 13. The complete extensions of the framework are* $\mathcal{E}_{CO}(AF_5) = \{\{\mathtt{b}, \mathtt{c}\}\}$. *We show that in all models of* Ψ_{CO} *the constant* $I_{\mathtt{c}}$ *has to be assigned to* TRUE. *Let us assume that there exists a model of* Ψ_{CO} *in which* $I_{\mathtt{c}}$ *is assigned to* FALSE. *According to the last clause of* $\Psi_{LegalUndecOut}$ $(I_{\mathtt{c}} \vee \neg O_{\mathtt{a}})$ *it follows that* $O_{\mathtt{a}}$ *would have to be* FALSE. *If* $O_{\mathtt{a}}$ *were* FALSE, *then the second clause of* $\Psi_{LegalUndecIn}$ $(\neg I_{\mathtt{b}} \vee O_{\mathtt{a}})$ *would be* FALSE *as well, since* $I_{\mathtt{b}}$ *has to be* TRUE *in all models of* Ψ_{ad} *(see Example 11). Thus* Ψ_{CO} *would evaluate to* FALSE, *which contradicts our assumption that this truth assignment is a model of* Ψ_{CO}. *Since we now know that* $I_{\mathtt{b}}$ *and* $I_{\mathtt{c}}$ *have to be* TRUE *in all models of* Ψ_{CO}, *it follows from the second clause of* $\Psi_{LegalUndecIn}$ $(\neg I_{\mathtt{b}} \vee O_{\mathtt{a}})$ *that* $O_{\mathtt{a}}$ *has to hold in every model of* Ψ_{CO}. *Hence we can see that* Ψ_{CO} *has only one model. The model assigns* $I_{\mathtt{b}}$, $I_{\mathtt{c}}$ *and* $O_{\mathtt{a}}$ *to* TRUE *(and the other constants to* FALSE*), which is compliant to the only complete extension* $\mathcal{E}_{CO}(AF_5) = \{\{\mathtt{b}, \mathtt{c}\}\}$.

In this chapter we have seen how the problem of searching for an admissible set of arguments in an argumentation framework can be reduced to the problem of searching for a model of a formula in Propositional Logic. We showed that the formula created to represent the property of admissibility can be extended to represent the requirements of complete extensions in an argumentation framework. We will use this reduction for the implementation of the proposed algorithmic shortcuts.

3 Related Work

In this section we recall the current state-of-the-art algorithms to solve DS-PR. Besides their general ideas on how to solve the problem, we are interested in any shortcuts that these solvers use and in case of a SAT-based approach, which questions these solvers ask to the NP-oracle, the SAT-solver. In the last years, the International Competitions on Computational Models of Argumentation (ICCMA) showed how much SAT-based approaches are dominating the stage. In all five competitions from 2015 to 2023 the fastest approach to solve the problem DS-PR reduced it to SAT [25, 9, 10, 20, 33]. That is why we focus on SAT-based approaches in this thesis, especially those with excellent results in the last competition in 2023 [25]. Nevertheless, to get a complete understanding of the variety of solving approaches we will first take a look at approaches trying to directly solve the problem, without a reduction to SAT in Section 3.1.

We will then slightly divert from complete argumentation solvers and recall two solvers, which approximate the solution to DS-PR, using shortcuts which are of interest for the following work of this thesis. We will first describe the approach behind the argumentation solver FARGO [24] in Section 3.2. As the second approximating argumentation solver we will take a look at HARPER++ [24] in Section 3.3.

We will then turn our attention to complete SAT-based argumentation solvers. In Section 3.4 we will describe a fundamental technique used in all state-of-the-art SAT-based solvers, which is called the Counter-Example-Guided Abstraction Refinement (CEGAR) [14]. Finally we will discuss the most important SAT-based solvers. We start with CEGARTIX [18] in Section 3.5, followed by ARGSEMSAT [12] in Section 3.6, μ-TOKSIA [28] in Section 3.7, FUDGE [32] in Section 3.8 and REDUCTO [4] in Section 3.9.

3.1 Direct Approaches

There have been different approaches to solve the problem of DS-PR without the reduction to another problem. One of those approaches is a depth-first backtracking procedure by Nofal, Atkinson and Dunne in 2014 [29], which searches through all possible labelings of a framework by a binary tree-search. It uses a labeling-based definition of the preferred argumentation semantics and extends the labeling by two new labels ($BLANK, MUST-OUT$). The algorithm starts in a labeling in which all arguments are labeled as $BLANK$. Subsequently the algorithm labels in each step i one argument a_i as IN, updates the labeling to be coherent with this change and restarts the process for the next argument a_{i+1} labeled $BLANK$. This iterative procedure continues until no argument is labeled $BLANK$ anymore. Having reached such a final state in the search tree, the algorithm evaluates if the found labeling is preferred. If the labeling is admissible and the corresponding extension is not a subset of any other already found preferred extension, then the algorithm marks the labeling as a preferred labeling. In case the labeling was not preferred or labeled the query argument as IN the algorithm backtracks to a inner node n in the search tree. Going back in the iterative procedure, the algorithm relabels the argument a_n of the iteration step at n as UNDEC, creating the second branch of the binary search tree.

In the worst case, the algorithm would have to iterate through all possible labelings to guarantee that all preferred labelings have been enumerated and to certify that the query argument is labeled IN in all those labelings.

In the best case, the first found preferred extension would not contain the query argument. In this case a counter-example was found and the algorithm terminates without having to enumerate further preferred labelings.

Applied Shortcut Before starting the iteration of the preferred labelings, the algorithm checks if any of the attackers of the query argument is credulously accepted, which would render the query argument as not skeptically accepted. To check the credulous acceptance, the algorithm searches for an admissible set containing the attacker. The intention of this shortcut is close to one of the ideas proposed in this thesis, which is described in Section 4.

3.2 FARGO

The argumentation solver FARGO by Matthias Thimm [24] approximates the solution of DS-PR by checking if admissible sets can be found in which either the query argument or any of its attackers are contained [24]. In case that the solver found an admissible set containing the query argument, but did not find an admissible set containing any attacker of the argument, the solver returns that the query argument is skeptically accepted. In all other cases the solver returns that the argument is not skeptically accepted. The search for an admissible set containing a specific argument is realized as a variant of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [8], which is a backtracking search algorithm. In contrast to such a complete search algorithm, FARGO implements a maximum search depth and can thus only approximate the solution of the problem in some cases. Starting from the argument in question FARGO looks if any attacker exists against which the current set is not defended by any other argument in the framework. If so, the algorithm terminates and reports that no admissible set containing this argument exists. In case there exists an argument which defends the current set, it gets added to the set, the search depth gets increased and the procedure repeats with the new set.

If the solver reports that there is an admissible set containing the specified argument, then the answer is always correct, because the solver constructed such an admissible set during the process. If the solver however reports that there is no admissible set containing the specified argument, then the answer can be incorrect since the algorithm terminates preliminary if the search depth exceeds the limit fixed prior to the calculation.

As to performance, the solver is not only faster than all state-of-the-art complete argumentation solvers, but it also ranked the second place for the problem of DS-PR in the "Approximate Track" of ICCMA'23 [25].

Applied Shortcut The two questions about the existence of an admissible set containing the query argument and about the existence of an admissible set containing one of its attackers, can be seen as shortcuts in the sense described in Section 1. We take up the intentions of these shortcuts for the proposed approaches in Section 4.

3.3 HARPER++

The argumentation solver HARPER++ by Matthias Thimm [24] approximates the solution of DS-PR of a query argument by checking if the argument is contained in the grounded extension of the problem's argumentation framework. Since the grounded extension is a subset of all preferred extensions, an argument contained in the grounded extension has to be contained in every preferred extension. However, as seen in Example 4 in Section 2.1, the intersection of all preferred extensions can contain arguments which are not contained in the grounded extension. In this case HARPER++ falsely rejects the skeptical acceptance of the argument.

Though seemingly simple in its design, HARPER++ showed to be the most efficient argumentation solver in the "Approximate Track" of ICCMA'23 for DS-PR [25].

Applied Shortcut HARPER++ showed that the shortcut of using the grounded extension is quite good in approximating the solution to the problem of DS-PR. The potential of this shortcut is further investigated in Section 4.

3.4 CEGAR

Recalling the definition of skeptical acceptance (see Definition 13 in Section 2.1), arguments are only skeptically accepted if they are labeled IN in all labelings or if they are contained in all extensions of an argumentation semantics applied on an argumentation framework. That is why, most approaches to check the skeptical acceptance of an argument enumerate several (if not all) extensions or labelings of an argumentation framework in search of a counter-example. In case of SAT-based approaches, they all follow more or less the SAT-based Counter-Example Guided Abstraction Refinement (CEGAR) approach [14]. The CEGAR approach proposes 4 steps to check a property of a system. In the context of DS-PR these steps could be described as follows:

1. Encode the argumentation framework as a propositional formula, which models associate to an admissible set or a complete extension
2. Use a SAT-solver to check if the propositional formula is satisfiable, if not then terminate the process
3. Decode the returned model of the formula as a labeling (extension) and check if this labeling (extension) is a counter-example to the skeptical acceptance, if it is then terminate the process
4. Refine the propositional formula and return to Step 2

These steps are iterated until a counter-example is found or until the propositional formula is no more satisfiable. If no counter example was found and the formula is unsatisfiable, then the argument is seen as skeptically accepted in most cases. Note that we can only declare the argument as skeptically accepted if we have checked that there is at least one non-empty admissible set in the argumentation framework. Otherwise if no non-empty admissible set exists, what means that no non-empty preferred extension exists, it is common to reject the skeptical acceptance of the argument, see the rules of ICCMA'23 [25].

As we have mentioned in Section 2.2, in most cases SAT-solvers only return one model per invocation. Invoking the SAT-solver again with the same propositional formula there is a considerable chance of receiving the same model. That is why the refinement in Step 4 is crucial for enumerating several admissible (or complete) sets of an argumentation framework. The propositional formula has to be modified in a way, that all of its models still adhere to the requirements necessary for an admissible (or complete) set, but so that a new model is returned by the SAT-solver. To provoke the SAT-solver to return a model different to the models already known, the propositional formula can be extended by requirements that render the already known models as unacceptable. Such an add-on to the propositional formula, which is formulated in CNF, is called a complement clause or *blocking clause* [35]. Let us assume we have an argumentation framework $\mathsf{AF} = (A, R)$ and the last call to the SAT-solver returned a model that can be interpreted as the extension E_1. A common way to design such a blocking clause is to create a disjunction containing a propositional constant I_{a} for each argument $\mathsf{a} \in A \setminus E_1$, used e.g. in CEGARTIX [18]. Following our encoding in Section 2.2, this clause demands that each future model of the now modified formula has to label at least one argument as IN that has not been labeled IN in the last model. Such a clause is added to the propositional formula for each

found extension. It forces the SAT-solver to iterate through all different extensions of the argumentation framework until a counter-example is found or the modified propositional formula becomes unsatisfiable.

3.5 CEGARTIX

One of the earliest and most influential argumentation solvers that follows the CEGAR-approach is CEGARTIX by Dvořák et al. [18]. To solve the problem of DS-PR, the solver encodes the given argumentation framework as a propositional formula, so that the models associate with the complete extensions of the framework. The satisfiability of this propositional formula is then checked by a (extern) SAT-solver.

In addition to the encoding shown in Section 2.2, the propositional formula is extended to search for only those models associated with complete extensions in which neither the query argument, nor any attacker of the query argument are contained. Adding the information that the propositional constant I of each of those arguments is FALSE can have a beneficial impact on the runtime needed by the SAT-solver to check the formula. Most modern SAT-solvers use techniques such as Unit Propagation [8]. Adding more and shorter clauses to an encoding can enable the SAT-solver to exploit these techniques during calculation. Readers interested in this and further techniques about improving the performance of the usage of SAT-solvers are referred to [8].

In the main part of the algorithm, CEGARTIX repeats solving the SAT problem of this formula for as long as the formula is satisfiable. Within each iteration of the loop, the algorithm searches at first for a complete extension fulfilling the requirements mentioned above. If such a complete extension is found, then the algorithm tries to maximize the size of this extension. To do so the algorithm repeatedly calls to the SAT-solver asking for a super set of the already found complete extensions. Note that all these super sets fulfill the requirement that the query argument is not labeled IN. If no larger complete extension without the query argument can be found anymore, then the algorithm calls the SAT-solver again to check if adding the query argument to the set is a valid complete extension. The algorithm rejects the skeptical acceptance of the query argument if the union of the found complete extension and the query argument is not a valid complete extension. In this case it has found a preferred extension not containing the query argument. If the union is a valid complete set and hence the found preferred set contains the query argument, the algorithm adds a blocking clause to the propositional formula. The blocking clause demands from the SAT-solver that any new model of the formula has to label at least one argument outside of the found preferred extension as IN. In case that the so modified propositional formula is no longer satisfiable, the algorithm terminates and reports that the query argument is skeptically accepted.

The repeated calls to the SAT-solver to maximize the size of the complete extensions have shown to be computationally less efficient than modern approaches, which try to limit the number of calls to the SAT-solver. Nevertheless at the ICCMA in 2015 the solver scored the second place after ARGSEMSAT for the problem of DS-PR. The solver also strongly influenced some state-of-the-art argumentation solvers like μ-TOKSIA [28].

Applied Shortcut Before starting to iterate through the complete extensions of the framework, CEGARTIX checks with a single call to the SAT-solver if there exists a complete extension which contains an argument attacking the query argument. Due to the definition of the preferred semantics, it follows from the existence of such a complete extension that there exists a preferred extension attacking the query argument. If that is the case, then the solver rejects

the skeptical acceptance of the query argument. If however there is no such complete extension, then the problem stays unsolved. That is why this technique has limited applicability and matches the former stated definition of a shortcut. Note that in contrast to the shortcuts applied in former described (approximating) argumentation solvers, this shortcut uses a NP-oracle that might not finish in polynomial time. This shortcut is further investigated in Section 4.

3.6 ARGSEMSAT

The first argumentation solver, that won an ICCMA competition in the track for DS-PR in 2015 was ARGSEMSAT created by Cerutti et al. [12]. The solver roughly follows the CEGAR approach, which means it is looking for a counter-example, i.e. a preferred labeling with the query argument not labeled IN. To enumerate the preferred labelings of a framework ARGSEMSAT encodes the framework as a propositional formula, which models associate to complete labelings in the framework.

Because the algorithm is working with labeling-based semantics it uses the term "labeling $\texttt{Lab}_1$ is less committed than $\texttt{Lab}_2$" to describe the fact that $\texttt{in}(\texttt{Lab}_1) \subset \texttt{in}(\texttt{Lab}_2)$. To enumerate preferred labelings, the algorithm calculates a complete labeling $\texttt{Lab}_{\mathsf{CO}}$, which is not less committed than any of the already found preferred labelings. $\texttt{Lab}_{\mathsf{CO}}$ has to label the query argument to UNDEC. ARGSEMSAT then tries to maximize the commitment of labelings based on $\texttt{Lab}_{\mathsf{CO}}$ with repeated calls to the SAT-solver. If the most committed complete labeling $\texttt{Lab}_{COmax}$ (that subsumes $\texttt{Lab}_{\mathsf{CO}}$) has been found, then the algorithm checks if changing the labeling to $\texttt{Lab}_{COmax+}$ so that the query argument is labeled IN results in an even more committed complete labeling. If $\texttt{Lab}_{COmax+}$ is not a valid complete labeling, then the algorithm has found a counter-example ($\texttt{Lab}_{COmax}$) and terminates rejecting the skeptical acceptance of the query argument. If $\texttt{Lab}_{COmax+}$ is a valid complete labeling, and hence a preferred labeling, then the algorithm adds the labeling to its set of already found preferred labelings and restarts the main loop. In case that the SAT-solver cannot find anymore complete labelings that are not less committed than the already found preferred labelings, the algorithm terminates reporting that the query argument is skeptically accepted.

Applied Shortcuts The algorithm makes use of two shortcuts before enumerating all preferred labelings in the main loop. The first used shortcut asks the SAT-solver if there exists a complete labeling, which labels the query argument as OUT. If that is the case, the algorithm rejects the skeptical acceptance of the query argument. This shortcut is similar to the one used in CEGARTIX. The second shortcut asks the SAT-solver if there exists at least one complete labeling in which the query argument is labeled IN. If the answer to this question is NO, then the algorithm rejects the skeptical acceptance. We have seen a similar shortcut used in FARGO, when the solver searched for at least one admissible set containing the query argument. In contrast to the approach used in FARGO, ARGSEMSAT is using the SAT-solver for this search. It ensures that ARGSEMSAT receives a correct answer in every case, but the answer might not be given in polynomial time. In case that there exists a complete labeling with the query argument labeled IN, the decision problem stays unsolved. Both shortcuts of ARGSEMSAT are investigated in Section 4.

3.7 μ-TOKSIA

The argumentation solver μ-TOKSIA, created by Niskanen and Järvisalo [28], is one of the most efficient solvers of the current state of the art. It has ranked first place in many tracks of ICCMA in the years 2023 and 2019 [25, 10]. In case of DS-PR it won both competitions.

To decide about the skeptical acceptance of an argument using preferred semantics, μ-TOKSIA uses a modified version of the CEGARTIX algorithm. The only modifications made is that μ-TOKSIA does not make use of any shortcuts, but only uses the main loop of the algorithm. The algorithm searches for complete extensions, maximizes them in size wrt. set inclusion and checks if they contain the query argument, see Section 3.5.

The fact that μ-TOKSIA performs so much better than CEGARTIX [28], raises the question if the use of shortcuts is justifiable. At first we should note that the way how the two solvers were implemented, which is not identical, might have an impact on these differences in performance. The applied SAT-solver will also have a considerable effect on the runtime efficiency. The currently most efficient version of μ-TOKSIA (version ICCMA'23) uses the SAT-solver GLUCOSE. It was compared in [28] to the newest version of CEGARTIX (version ICCMA'2017) that used the much older SAT-solver MINISAT. There seem to be several reasons for μ-TOKSIA's higher efficiency. One reason could also have been the use of shortcuts in CEGARTIX. As we have discussed in Section 3.5, CEGARTIX uses a shortcut that asks the SAT-solver if there exists a complete extension that attacks the query argument. If such an extension can be found, the shortcut is indeed a considerable way to accelerate the solving process and prevents the solver from having to call the SAT-solver in the main loop of the algorithm. However, if no such extension exists, then there is not much knowledge gained that would accelerate the calculations in the main loop[2]. This means, in cases where the query argument is not attacked by a complete set, CEGARTIX wastes runtime for a call to the SAT-solver, whereas μ-TOKSIA can use this time to begin enumerating the preferred extensions of the framework. This example shows that it is crucial to employ shortcuts prudently and ideally only if one has reason to believe that they will further the solving process. That is exactly one of the reasons why we investigate shortcuts like the one used by CEGARTIX in this thesis. It shows why it is of interest to examine how often and how much these shortcuts actually improve the performance of an argumentation solver.

3.8 FUDGE

The argumentation solver FUDGE, created by Thimm et al. [32], implements an approach that is different to the former mentioned solvers. The solver avoids having to enumerate preferred extensions and thus does not have to maximize (wrt. to set inclusion) any found admissible set. To decide about the acceptance of an argument, the solver focuses on those admissible sets that are in conflict with any admissible set containing the query argument. Let S be an admissible set containing the query argument and S' be an admissible set attacking S. For each of these potential counter-examples, such as S', the algorithm creates super sets, which contain the query argument. Let S'' be a super set of S', which contains the query argument. The algorithm then checks if any of these supersets is admissible. If S'' is admissible, then all preferred extensions originating from the potential counter-example (S') will contain the query argument, thus the counter-example is defused. If no such admissible superset containing the query argument exists, then a valid counter-example has been found. From this description we can see that FUDGE is also following the CEGAR approach, by calculating an admissible set, checking if the set is an counter-example and refining the propositional formula to enumerate different admissible sets. The main difference to algorithms like CEGARTIX is that FUDGE neither has to compute a preferred extension nor has to enumerate all preferred extensions. However, FUDGE has to enumerate the different potential counter-examples, i.e. all admissible sets attacking an admissible set containing the query argument. For each those sets FUDGE

[2] In this case CEGARTIX learns that the propositional constants of the attacking arguments are to be set FALSE in future calls to the SAT-solver [18].

has to iterate through the space of supersets of these potential counter-examples to come to a verdict.

FUDGE ranked second in the ICCMA competition 2021 for the problem DS-PR in the main track [9].

Applied Shortcut Before enumerating the admissible sets, FUDGE employs a shortcut by calling the SAT-solver asking if at least one admissible set containing the query arguments exists. This first call is necessary and cannot be omitted for the algorithm to work correctly. Imagine there is no non-empty admissible set in the framework. Hence there would be no potential counter-example to be checked in the main loop of the algorithm. If the first check were omitted in this case, the algorithm would have no means of knowing if the query argument is contained in at least one preferred extension or not. We take a closer look at this shortcut in Section 4 . It should be noted that we only consider an auxiliary use of shortcuts in this thesis. It means that shortcuts and the complete algorithms which they precede are considered as being independent from one another.

3.9 REDUCTO

The argumentation solver REDUCTO, created by Bengel et al. [4], is to the best of our knowledge the currently most recent development in argumentation solvers. Thus the solver did not yet participate in any ICCMA competition, but is submitted for ICCMA 2025. For the problem of DS-PR the solver makes use of the notion of *vacuous reduct semantics* [30]. This means that it can be checked if a complete extension Ext_1 is a preferred extension in the framework AF by searching for a complete extension in a reduced version of the framework, called AF_2. AF_2 results from reducing AF by Ext_1. *Reduction* in this context means to remove the arguments of Ext_1, as well as the arguments attacked by Ext_1 from the framework AF. The complete extension Ext_1 is a preferred extension in AF if and only if there is no non-empty complete extension in AF_2. This means that REDUCTO, while enumerating the complete extensions of the framework, can check with one additional call to the SAT-solver if this extension is an preferred extension. The solver can therefor omit the costly maximization of the size wrt. set inclusion of an extension, which is needed in algorithms like CEGARTIX [18] or μ-TOKSIA [28].

Applied Shortcuts Before enumerating the complete extensions of the given framework, REDUCTO calculates the grounded extension in polynomial time. Similar to the intuition that lead to HARPER++ [24], the solver checks if the query argument is contained or attacked by the grounded extension. The solver also checks if the query argument attacks itself, which renders the argument as skeptically unacceptable. Another shortcut used by REDUCTO is to check if the query argument has any attacking arguments at all. If the argument is unattacked, then it has to be skeptically accepted. We will further investigate these shortcuts in Section 4.

This chapter has given an overview of the current state-of-the-art argumentation solvers. We highlighted the algorithmic shortcuts which have been used in some of these solvers. Although shortcuts for DS-PR have been used in argumentation solvers for quite some time (at least since the designs of Nofal et al. [29]), there has been, to the best of our knowledge, no evaluation of their impacts on the runtime efficiency of these solvers.

4 Algorithmic Shortcuts for Deciding Skeptical Preferred Acceptance

In this section we discuss different possible algorithmic shortcuts for the problem of DS-PR. Based on the notion introduced in Section 1, we define an algorithmic shortcut as a *sound* but not a *complete* approach to solve the reasoning problem. The soundness of the proposed approaches guarantees that all answers calculated by these shortcuts correctly solve the problem. However, because these approaches are not complete, we cannot calculate a solution for every problem instance of the reasoning problem. In Section 4.1 we discuss the solving approaches of different shortcuts from the literature as well as new shortcuts. Section 4.2 shows how these shortcuts can be encoded as propositional formulas. In Section 4.3 we discuss how these shortcuts have been implemented as single programs.

4.1 Overview of the Algorithmic Shortcuts

This section describes the algorithmic shortcuts that we investigate in this thesis. Part of these shortcuts is known from the literature, but we also propose some shortcuts which are, to the best of our knowledge, new approaches to solve the problem of DS-PR. Due to the definition of DS-PR (see Section 2.1) which demands that no admissible set of arguments attacks the query argument or is attacked by the query argument, the proposed shortcuts focus on the direct conflicts of the query argument.

All shortcuts discussed in this thesis are summarized in Table 1. Each row in the table describes one shortcut, which is denoted with a name in the first column. The second column describes the characteristic question of the shortcut. Depending on the answer to this question, which is shown in the third column, we can make a conclusion about the skeptical acceptance under preferred semantics of the query argument, which is noted in the forth column of Table 1. Note that not all answers to these questions lead to conclusions about the skeptical acceptance of the query argument. Take S1 as an example. We can only draw a conclusion if the question about the existence of a self-attack of the query argument is answered with YES. If the question is answered with NO, then we cannot immediately draw a conclusion whether the query argument is skeptically accepted or not. Note that Table 1 omitted those cases in which no conclusion can be drawn.

Several shortcuts in Table 1 ask for the existence of a complete labeling with a certain property. Most of the used shortcuts in the literature use complete labelings. We will therefore adhere to this common practice. Note that all of these shortcuts could also be formulated checking the existence of admissible labelings with specific properties[3].

We will discuss each shortcut in detail in the following paragraphs. For each shortcut, we will first present the characteristic question that the shortcut poses to solve the reasoning problem of DS-PR. For those shortcuts taken from the literature, we will then recall in which argumentation solvers the shortcut was applied. Finally, we will show that the shortcut soundly solves DS-PR and give an example of how the shortcut solves the problem. The query argument is denoted with q in the following paragraphs.

Shortcut S1 The shortcut S1 poses the question if the query argument q is attacking itself. The shortcut is based on the fact, that a self-attacking argument cannot be part of an admissible

[3]The differences in runtime efficiency of using a SAT-solver to compute complete or admissible labelings are not yet completely investigated. See [12, 18] for discussions on the usage of complete or admissible labelings/extensions for solving the reasoning problem DS-PR with a complete algorithm.

Name	Characteristic Question	Answer	Conclusion
S1	Is the query argument attacking itself?	YES	Not Accepted
S2	Has the query argument any attackers?	NO	Accepted
S3	Is the query argument contained in the grounded extension?	YES	Accepted
S4	Is the query argument attacked by the grounded extension?	YES	Not Accepted
S5	Does a complete labeling exist in which the query argument is labeled as IN?	NO	Not Accepted
S6	Does a complete labeling exist in which the query argument is labeled as OUT?	YES	Not Accepted
S7	Does a complete labeling exist in which at least one victim of the query argument is labeled as IN?	YES	Not Accepted
S8	Does a complete labeling exist in which all defenders against at least one attacker of q are labeled OUT?	YES	Not Accepted
S9	In case that a non-empty complete labeling exists, does a non-empty complete labeling exist in which the query argument is labeled as NOT IN?	NO	Accepted
S10	In case that a non-empty complete labeling exists, does a non-empty complete labeling exist in which all self-defending arguments are either labeled IN or OUT and in which q is labeled NOT IN?	NO	Accepted

Table 1: The proposed algorithmic shortcuts investigated in this thesis. The shortcuts are represented by their name, their characteristic question, and the necessary answer to the characteristic question to draw a conclusion about the skeptical acceptance of a query argument.

set of arguments.

This shortcut is a very basic check and used for example in the argumentation solver REDUCTO, see Section 3.9.

In case that the question is answered with YES, then q is indeed attacking itself. Due to the conflict-freeness of admissible sets of arguments, we can therefore conclude that q cannot be part of any admissible set in the argumentation framework of the problem instance given. We recall at this point that preferred extensions are defined as admissible sets of maximum size, see

Section 2.1. Since q is not contained in any admissible set, we can conclude that it will not be contained in any preferred extension. Due to the definition of skeptical acceptance, see Section 2.1, and because q is not contained in any preferred extension, we can conclude that it is not skeptically accepted. We demonstrate this situation in an example.

Example 15. *Let $AF_6 = (A, R)$ be an argumentation framework as depicted in Figure 6. We can clearly see that* q *is attacking itself. The only non-empty admissible set in this framework is* {b}*. It is also the only preferred extension of this argumentation framework.* q *is not contained in this preferred extension, therefore it is not skeptically accepted under the preferred semantics in AF_6.*

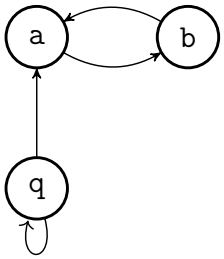

Figure 6: The argumentation framework AF_6.

Shortcut S2 In shortcut S2 we ask if there are any arguments that are attacking the query argument q. The basic idea of this shortcut is that unattacked arguments are contained in each preferred extension.

An application of this shortcut can be found in the argumentation solver REDUCTO, see Section 3.9.

If the posed question is answered with NO, then we know that q is an unattacked argument. Recalling the definition of preferred extensions in Section 2.1, we know that preferred extensions are admissible sets of maximum size wrt. set inclusion. Due to Dungs fundamental Lemma (see Lemma 1) we know that we can add the (unattacked) admissible set {q } to each other admissible set S and receive an admissible super set of S. Therefore each admissible set of maximum size wrt. set inclusion has to contain q. Which means that every preferred extension has to contain q and that is why q has to be skeptically accepted under preferred semantics in this case.

Example 16. *Let $AF_7 = (A, R)$ be an argumentation framework as depicted in Figure 7.* q *has no attackers in this framework. The only preferred extension of this framework is* {b, q}*. We see that* q *is skeptically accepted under the preferred semantics.*

Shortcut S3 and S4 The third and forth shortcut are both using the grounded extension of an argumentation framework. They ask if the query argument q is either contained or attacked by the grounded extension.

This shortcut is well known in the literature, see its use in HARPER++ (see Section 3.3) and in REDUCTO (see Section 3.9).

If q is contained in the grounded extension, then the argument is skeptically accepted, since the grounded extension is a subset of all preferred extensions (see Section 2.1). In case that q is attacked by the grounded extension, then the query argument cannot be skeptically accepted, because in this case each preferred extension contains a subset attacking q.

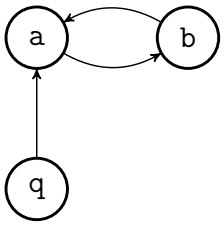

Figure 7: The argumentation framework AF_7.

Example 17. *Let $\mathsf{AF}_7 = (A, R)$ be an argumentation framework as depicted in Figure 7. Since* q *is unattacked, it is contained in the grounded extension $E_{GR}(\mathsf{AF}_7) = \{\mathtt{q}\}$, as well as in the only preferred extension $\mathcal{E}_{PR}(\mathsf{AF}_7) = \{\{\mathtt{b}, \mathtt{q}\}\}$. Therefore* q *is skeptically accepted under the preferred semantics.*

Example 18. *Let $\mathsf{AF}_8 = (A, R)$ be an argumentation framework as depicted in Figure 8. In this framework the grounded extension $E_{GR}(\mathsf{AF}_8) = \{\mathtt{a}\}$ is attacking* q*. The only preferred extension $\{\mathtt{a}\}$ does not contain* q*. We can conclude that* q *is not skeptically accepted under the preferred semantics.*

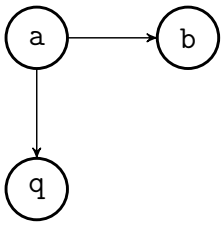

Figure 8: The argumentation framework AF_8.

Shortcut S5 Shortcut S5 in Table 1 checks if at least one complete labeling $\mathtt{Lab}_{\mathtt{s5}}$ exists, in which $\mathtt{Lab}_{\mathtt{s5}}(\mathtt{q}) = \mathsf{IN}$.

This shortcut has been used in several argumentation solvers in the literature, notably in FARGO (see Section 3.2), in ARGSEMSAT (see Section 3.6), and in FUDGE (see Section 3.8). It should be noted that FUDGE asks for an admissible set containing the query argument. Recalling that all complete extensions are admissible sets, and that extensions can be transformed into labelings and vice versa, we can easily see that the existence of $\mathtt{Lab}_{\mathtt{s5}}$ concludes the existence of an admissible set which contains q and vice versa.

In case that the question is answered with NO, we can conclude that $\mathtt{Lab}_{\mathtt{s5}}$ does not exist in the framework. Following Definition 5 we know that preferred extensions are complete extensions of maximum size wrt. set inclusion. Since there is no complete extension containing q, we can conclude that there cannot be a preferred extension in the framework which contains q. That is why q cannot be skeptically accepted.

Example 19. *Let $\mathsf{AF}_8 = (A, R)$ be an argumentation framework as depicted in Figure 8. The only complete labeling in this framework $\{\{\mathtt{a}\}, \{\mathtt{b}, \mathtt{q}\}, \emptyset\}$ labels* q *as* OUT*. In this framework exists no complete labeling which labels* q *as* IN*. It is easy to see that* q *is not skeptically accepted in AF_8.*

Shortcut S6 The sixth shortcut S6 in Table 1 checks if a complete labeling $\mathtt{Lab}_{\mathtt{s6}}$ exists, with $\mathtt{Lab}_{\mathtt{s6}}(\mathtt{q}) = \mathsf{OUT}$. The intuition of this shortcut is identical with the question if there exists a complete extension containing any attacker of q.

This shortcut was used in FARGO (see Section 3.2), in CEGARTIX (see Section 3.5), in ARGSEMSAT (see Section 3.6) and in the direct approach of Nofal et al. (see Section 3.1).

If the question is answered with YES, then we can conclude that $\texttt{Lab}_{\texttt{S6}}$ exists in the framework. We can interpret the set $\mathsf{IN}(\texttt{Lab}_{\texttt{S6}})$ as a complete extension which is attacking q. We recall from Section 2.1 that every complete extension is a subset of at least one preferred extension. Therefore we can conclude that there has to be a preferred extension which is attacking q. Due to the conflict-freeness of preferred extensions, we can conclude that this extension does not contain q. That is why q cannot be skeptically accepted in this case.

A framework in which this shortcut solves the problem of DS-PR is discussed in Example 19.

Shortcut S7 For shortcut S7 in Table 1 we ask if a complete labeling $\texttt{Lab}_{\texttt{S7}}$ exists, in which any argument that is attacked by q, is labeled IN. The intuition of this shortcut is that, if the query argument q attacks an admissible set S, then q also attacks the preferred extension, which is a superset of S.

This shortcut has not been used in any of the mentioned argumentation solvers.

If the question is answered with YES, we can conclude that q attacks an admissible set. We recall that for each admissible set S there exists a superset of maximum size wrt. set inclusion S_{max} with $S \subseteq S_{max}$. Following Definition 5 S_{max} is a preferred extension. This means that a preferred extension exists, which is attacked by q. Due to the conflict-freeness of preferred extensions, this preferred extension does not contain q. We conclude that not all preferred extensions contain q, which renders q as not skeptically accepted.

Example 20. *Let $\mathsf{AF}_9 = (A, R)$ be an argumentation framework as depicted in Figure 9. The set* {a} *is an admissible set because it defends itself against its attacker* q*. The set is also one of the preferred extensions of the framework, which renders* q *as not skeptically accepted.*

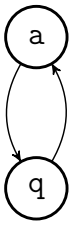

Figure 9: The argumentation framework AF_9.

Shortcut S8 For shortcut S8 in Table 1 we ask if there exists a complete labeling in which q is not defended against at least one attacker. This shortcut can be seen as a more nuanced version of S6, in which we ask if any complete labeling labels q as OUT. The motivation for this shortcut can be seen in the following example. For the following we call an argument d a defender of q against x, iff d is attacking x while x is attacking q and d and x are not the same argument.

Example 21. *Let $\mathsf{AF}_{10} = (A, R)$ be an argumentation framework as depicted in Figure 10. The complete extensions of this framework are $\mathcal{E}_{CO} = \{\{\texttt{c}\}, \{\texttt{b}, \texttt{q}\}, \emptyset\}$. The preferred extension are $\mathcal{E}_{PR} = \{\{\texttt{c}\}, \{\texttt{b}, \texttt{q}\}\}$. In S6 we require that* q *has to be labeled legally OUT, due to the definition of a complete labeling in Section 2.1. Figure 10 shows a framework, in which* q *is never labeled legally OUT, but is still not skeptically accepted. S6 would return as answer* NO*, leading to no conclusion. However, asking for a complete labeling in which all defenders against an attacker of* q *are labeled OUT we receive*

YES, *since in Figure 10* b *is labeled* OUT *in one complete labeling and* b *is defending* q *against* a. *Note that* a *cannot defend* q *against itself.*

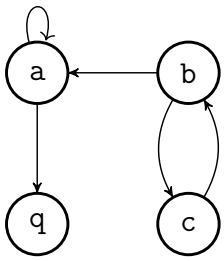

Figure 10: The argumentation framework AF_{10}.

Note that asking the characteristic question of S8 is only reasonable if there are both attackers and defenders of the query argument in the framework. If there are no attackers of the query argument we can directly conclude that it is skeptically accepted, see S2. In case there is at least one attacker without any defender, then we can conclude that the query argument is attacked by the grounded extension and hence not skeptically accepted, see S4. A query argument that is only attacked by itself is also a case in which no defender compliant to the stated definition exists. Having no other attacks, the query argument is labeled UNDEC in all labelings and hence not skeptically accepted.

If the characteristic question of S8 is answered with YES we know that a labeling $\mathtt{Lab}_{\mathtt{S8}}$ exists, in which the set of all defenders S_D of q against an attacker x are labeled (legally) OUT. This means that there has to be a set of arguments S_{atk} with $S_{atk} \subseteq \mathtt{in}(\mathtt{Lab}_{\mathtt{S8}})$ which is attacking the defenders S_D. Therefore we know that the complete extension $E_1 = \mathtt{in}(\mathtt{Lab}_{\mathtt{S8}})$ attacks S_D. We recall that for each complete extension there has to be a preferred extension which is a superset of the complete extension. It follows that a preferred extension E_2 exists with $E_1 \subseteq E_2$ and that E_2 attacks S_D. We consider now two different cases. In case one the attacker x is not attacking itself, in case two x is attacking itself. In the first case we can conclude that E_2 has to contain x, because of the maximality wrt. set inclusion of E_2 and because all attackers of x are attacked by E_2. Due to the conflict-freeness of preferred extensions, we now know that E_2 cannot contain q in this case. If x is attacking itself, then x has to be labeled UNDEC in the preferred labeling of E_2, because all attackers of x (bot not itself) are labeled OUT. Because x is labeled UNDEC we can conclude that q has also to be labeled UNDEC or OUT in this labeling. Hence q is not labeled IN in the preferred labeling. It follows that in both cases q cannot be skeptically accepted.

Improving the Recognition of Skeptical Acceptance The most important motivation to use algorithmic shortcuts is to prevent the argumentation solver from having to make multiple calls to a NP-oracle to finally solve the problem (see Section 1). Due to the general scheme of the CEGAR approach, problem instances in which the query argument is indeed skeptically accepted under preferred semantics provoke the most calls to the NP-oracle and can be seen as the most difficult problem instances for CEGAR-based argumentation solvers. To solve these instances the argumentation solver has to iterate through all candidates which could be counter-examples (see Section 3.4). It is therefore especially interesting to investigate shortcuts which can conclude with a single call to a NP-oracle that a given query argument is indeed skeptically accepted under preferred semantics.

Of the proposed shortcuts so far only S3 has lead to this conclusion. However, there are cases in which S3 does not allow to draw this conclusion, as we will see in the following example.

Example 22. *Let $AF_{11} = (A, R)$ be an argumentation framework as depicted in Figure 11. The complete extensions of this framework are $\mathcal{E}_{CO}(AF_{11}) = \{\emptyset, \{\mathtt{q}\}\}$. The grounded extension is the empty set. That is why the query argument is not contained in the grounded extension. However,* q *is contained in each non-empty complete extension, as well as in each preferred extension. Therefore* q *is skeptically accepted.*

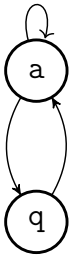

Figure 11: The argumentation framework AF_{11}.

One idea to solve frameworks like the one in Example 22 is to ask the question if q is contained in each non-empty complete extension. We denote this question as Sx. If the question Sx were answered with YES, one could be tempted to conclude that the query argument has to be skeptically accepted. However, Sx alone does not allow a sound conclusion about the skeptical acceptance of an argument, see Example 23. In case that no non-empty complete labeling exists in an argumentation framework, Sx can be answered with YES. The only preferred extension in such a framework is the empty set, which does not contain q. Hence q is not skeptically accepted in such a case, although Sx was answered with YES. If we add the necessary check to ensure that at least one non-empty complete extension exists to the question Sx we get to the intuition of the next proposed shortcut S9.

Example 23. *Let $AF_{12} = (A, R)$ be an argumentation framework as depicted in Figure 12. The only complete and therefore preferred extension is the empty set.* q *is attacking itself and cannot be contained in any admissible set. Hence* q *is not skeptically accepted.*

Figure 12: The argumentation framework AF_{12}.

Shortcut S9 Shortcut S9 in Table 1 asks two questions. The first questions asks if there exists a (non-empty) complete labeling $\mathtt{Lab}_{\mathtt{S9empt}}$ with $\mathtt{in}(\mathtt{Lab}_{\mathtt{S9empt}}) \neq \emptyset$. In case that $\mathtt{Lab}_{\mathtt{S9empt}}$ exists, the shortcut asks the second question. In the second question the shortcut checks if a non-empty complete labeling $\mathtt{Lab}_{\mathtt{S9}}$ exists, in which $\mathtt{Lab}_{\mathtt{S9}}(\mathtt{q}) \neq \mathsf{IN}$. The intuition of this shortcut is to check if the query argument is labeled IN in all non-empty complete labelings, in case that at least one such labeling exists.

If the first question is answered with YES and the second question is answered with NO, we can conclude that at least one non-empty labeling exists, but no labeling $\mathtt{Lab}_{\mathtt{S9}}$ exists. It means that all (non-empty) complete labelings of the given argumentation framework label q as IN. We infer the complete extensions from those labelings by collecting their IN-labeled

arguments in sets. All those complete extensions contain the query argument. Those extensions of maximum size wrt. set inclusion are the preferred extensions of the framework. The only complete extension which is not containing q is the empty set. Because the empty set is subset of all other sets of arguments we can conclude that all preferred extensions contain q. It follows that q is skeptically accepted in this case.

We demonstrate the application of S9 in an example.

Example 24. *Let $\mathsf{AF}_{13} = (A, R)$ be an argumentation framework as depicted in Figure 13. The complete extensions of this framework are $\mathcal{E}_{CO}(\mathsf{AF}_{13}) = \{\emptyset, \{\mathtt{a}, \mathtt{q}\}, \{\mathtt{b}, \mathtt{q}\}\}$. It is easy to see that* q *is contained in all non-empty complete extensions and therefore also contained in all preferred extensions. It follows that* q *is skeptically accepted under the preferred semantics.*

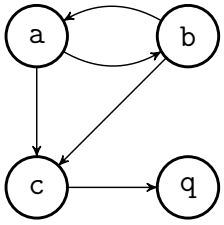

Figure 13: The argumentation framework AF_{13}.

Shortcut S10 Shortcut S10 again asks two questions. The first question asks if a non-empty complete labeling exists. If so the shortcut poses the second question. The second question asks if in all labelings in which all self-defending arguments are either labeled IN or OUT the query argument q is labeled IN. Shortcut S10 tries to relax the restrictions of S9. Instead of having to be labeled IN in all non-empty complete labelings, S10 asks if q is labeled IN in a distinct subset S of the (non-empty) complete labelings. It is important to note that each preferred labeling is contained in S. To see this relation we use some observations on self-defending arguments.

We start with a definition of the set of *self-defending* arguments.

Definition 16. *Let $\mathsf{AF} = (A, R)$ be an argumentation framework. We define $S_{SD} = \{a \in A \mid \{a\}^- \setminus \{a\}^+ = \emptyset, (a, a) \notin R\}$ as the set of all self-defending arguments.*

The self-defending arguments can be part of the grounded extension, but they do not have to be.

Example 25. *The arguments* a *and* b *in Figure 13 are both self-defending, but the grounded extension of the framework AF_{13} is the empty set.*

Since each self-defending argument alone is an admissible set, there has to be a preferred extension which contains the argument. But the relation between the preferred extensions and the self-defending arguments is even stronger, as we can see in the following theorem.

Theorem 1. *Every preferred extension of an abstract argumentation framework has to contain or attack every self-defending argument.*

Proof. As proof for Theorem 1 we recall that a preferred extension is an admissible set of maximum size wrt. set inclusion. Which means that all admissible sets of an argumentation framework have to be either attacked or contained by the preferred extension. If an admissible set

ext_{ad} is not attacked or contained by a preferred extension ext_{PR}, then recalling Dung's fundamental Lemma (see Section 2.1) $ext'_{\mathsf{PR}} = ext_{\mathsf{PR}} \cup ext_{\mathsf{ad}}$ would be an admissible set and $ext_{\mathsf{PR}} \subset ext'_{\mathsf{PR}}$ would hold, so that ext_{PR} could not have been a preferred extension in the first place. According to Definition 16, self-defending arguments are admissible sets, which concludes that a preferred extension has to contain or attack each self-defending argument. □

We add a definition for those labelings in which potential conflicts between the self-defending arguments are resolved.

Definition 17. *Let $\mathsf{AF} = (A, R)$ be an argumentation framework and S_{SD} be the set of self-defending arguments. We call a labeling in which every argument $\mathtt{a} \in S_{SD}$ is either labeled IN or OUT as a* ***labeling with settled self-defense****.*

Theorem 1 already implies our initial proposition that every preferred labeling is a labeling with settled self-defense. We highlight this conclusion in the following corollary.

Corollary 1. *Every preferred labeling is a labeling with settled self-defense.*

We recall that the set of all preferred extensions of a framework is a subset of the set of all complete extension of the framework (see Section 2.1). We also know that we can transform each complete extension into a complete labeling [3]. It follows Corollary 2.

Corollary 2. *Every preferred labeling is contained in the set of all complete labelings with settled self-defense.*

The intuition of S10 is to use the set of labelings with settled self-defense to limit the number of labelings in which we have to check if q is labeled IN. We only focus on those labelings essential for the reasoning problem DS-PR. With this reduction, we allow non-essential labelings to judge q as undecided, while we still maintain the ability to draw a conclusion about the skeptical acceptance of q. This means, that in difference to S9, using S10 we can draw a conclusion even if q has been labeled UNDEC in a labeling in which some self-defending arguments are also labeled UNDEC.

Note that the set of self-defending arguments can be distinguished from an abstract argumentation framework in polynomial time. We only need to iterate through the arguments of the framework and check if the argument is countering each attack.

The soundness of S10 is now easy to see. If the characteristic question is answered with NO, we can conclude that q is labeled IN in all complete labelings with settled self-defense. We denote the set of those labelings as $S_{\mathtt{Lab}_{\mathtt{SSD}}}$. Following Corollary 2, every preferred labeling is contained in $S_{\mathtt{Lab}_{\mathtt{SSD}}}$. Thus, q has to be labeled IN in each preferred labeling and is therefore skeptically accepted.

In case that no self-defending argument exists, S10 would naturally fall back to S9 and check if q is labeled IN in all complete labelings.

Example 26. *Let $\mathsf{AF}_{14} = (A, R)$ be an argumentation framework as depicted in Figure 14. The complete extensions of this framework are $\mathcal{E}_{PR}(\mathsf{AF}_{14}) = \big\{\emptyset, \{\mathtt{c}\}, \{\mathtt{f}\}, \{\mathtt{c}, \mathtt{f}, \mathtt{q}\}\big\}$. Not all non-empty complete extensions contain* q*. The former shortcut S9 would not be able to conclude that* q *is skeptical accepted under preferred semantics. Neither would S3, which checks if* q *is contained in the grounded extension come to this conclusion, because the grounded extension is the empty set. The self-defending arguments in this framework are* c *and* f*. It is easy to see, that these arguments cannot be labeled legally OUT. Therefore each complete labeling, which does not label one of these arguments as IN, has to label the argument in question as UNDEC. Thus the complete extensions $\{\mathtt{c}\}$ and $\{\mathtt{f}\}$ are associated with complete*

labelings in which one self-defending argument is labeled UNDEC*. That is why these labelings are non-essential in the sense described above for the assessment if* q *is skeptical accepted. It does therefore not limit the applicability of S10, that* q *is also labeled* UNDEC *in these labelings. The only complete labeling with settled self-defense is the one associated with the complete extension* $\{c, f, q\}$*. Since* q *is labeled* IN *in this labeling the shortcut S10 correctly resolves* q *as skeptical accepted.*

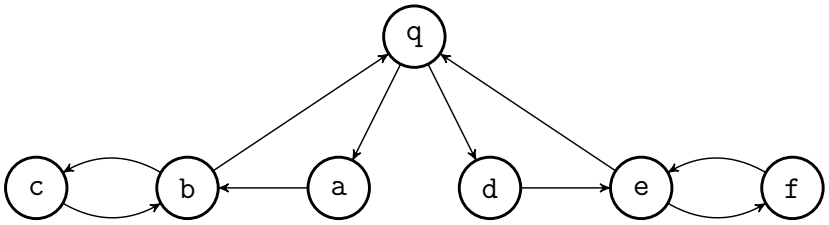

Figure 14: The argumentation framework AF_{14}.

Further Shortcuts We conclude this section discussing why any additional shortcuts using the well-known labels UNDEC and OUT do not seem to be of interest in regards of the goal of this thesis.

First, let us discuss the use of an algorithmic shortcut, that asks about the existence of a complete labeling $\mathtt{Lab}_{\mathtt{undec}}$ that labels q as UNDEC. If such a labeling exists, then we cannot directly conclude the acceptance status of q. While q is labeled UNDEC in $\mathtt{Lab}_{\mathtt{undec}}$ it might be labeled different in a preferred labeling $\mathtt{Lab}_{\mathtt{PR}}$ with $\mathtt{in}(\mathtt{Lab}_{\mathtt{undec}}) \subset \mathtt{in}(\mathtt{Lab}_{\mathtt{PR}})$. If $\mathtt{Lab}_{\mathtt{undec}}$ does not exist, then the existence of a labeling in which q is labeled OUT is still possible, but not guaranteed. Therefore we can again draw no conclusion about he skeptical acceptance of q.

Further shortcuts asking for labelings $\mathtt{Lab}_{\mathtt{NotOut}}$ that label q as NOT OUT or $\mathtt{Lab}_{\mathtt{NotUndec}}$ that label q as NOT UNDEC also seem to be unproductive.

If $\mathtt{Lab}_{\mathtt{NotOut}}$ exists, we only know that some complete labelings do not refute the query argument. We neither know if there is one complete labeling that does label q as OUT, nor if there is a preferred labeling that labels q as UNDEC. Both cases would render q as not skeptically accepted. Hence we can draw no immediate conclusion from this new knowledge. If $\mathtt{Lab}_{\mathtt{NotOut}}$ does not exist, which means that all complete labelings label q as OUT, we would get the same answer which we get by using S5. Recall that by using S5 we ask if any complete labeling labels q as IN. We use S5 in this thesis because S5 is more informative. In case that all labelings label q as UNDEC, S5 would return NO and solve the problem. Asking for $\mathtt{Lab}_{\mathtt{NotOut}}$ will return YES and would give us no indication wether the query argument is skeptically accepted or not.

Knowing about the existence of $\mathtt{Lab}_{\mathtt{NotUndec}}$ which labels q as NOT UNDEC does also not gain us any new information. In case that $\mathtt{Lab}_{\mathtt{NotUndec}}$ exists, we only know that not all complete labelings declare q as UNDEC. We do not know if one of them is refuting the argument neither do we know if all the preferred labelings label q as IN. If $\mathtt{Lab}_{\mathtt{NotUndec}}$ does not exist, it means that all labelings label q as UNDEC. Again this knowledge can be drawn from asking the question in S5, which would be answered with NO in this case.

4.2 SAT-Encodings for Acceptance Shortcuts

Since we are interested in improving the runtime efficiency of an argumentation solver, it is of importance to use the most efficient method possible to answer the characteristic questions

of the proposed algorithmic shortcuts. We already mentioned that the computation of the grounded extension is well-known in the literature (see [15]) and can be done in polynomial time [17]. We will therefore use this proven algorithm (for S3 and S4), which was used for example in the solver REDUCTO [4]. The shortcuts S1 and S2 are also easy to calculate.

As for the other proposed shortcuts (S5 to S10) we can see a similarity of these questions to the problem of credulous acceptance of an argument under the complete semantics (DC-CO). To check DC-CO we ask if at least one complete labeling exists, that labels the query argument to IN. Which means that similar to the shortcuts S5 to S10 we ask for the existence of at least one complete labeling with a certain property. It is therefore easy to see that we can reduce the problems of S5 to S10 to the problem of DC-CO. The problem of DC-CO has a complexity that is NP-complete [17], which is the same complexity than the problem SAT. As seen in Section 2.2, it is possible to reduce the problem of DC-CO in polynomial time to the problem of SAT [7]. It is therefore justifiable to use the sophisticated SAT-solvers to solve the problem of DC-CO as well as the shortcuts S5 to S10. The results of the last ICCMA competitions have shown, that SAT-based solving processes for the problem of DC-CO are more efficient than other approaches known from the literature [25], like for example the direct approach from Nofal et al. [29]. That is why we use a SAT-solver to answer the questions of the shortcuts S5 to S10.

In the following paragraphs we discuss the propositional clauses that are added to the clauses discussed in Section 2.2 to encode each characteristic question of these shortcuts as a propositional formula.

Encoding of S5 To answer S5 we ask the NP-oracle if a complete labeling exists, in which the query argument is labeled as IN. The NP-oracle can be a SAT-solver. Recall that the encoding Ψ_{CO} of Formula (9) in Section 2.2 asks for a (non-empty) complete extension. We can use this encoding to translate the characteristic question of S5 to a propositional formula. The non-emptiness of the extensions does comply with our intention to find a complete labeling in which at least one argument (here q) is labeled IN. To enforce the requirement that the query argument q has to be labeled as IN we add a clause demanding that I_{q} has to be set to TRUE

$$\Psi_{S5} = \Psi_{\mathsf{CO}} \wedge I_{\mathsf{q}} \tag{10}$$

In case that Formula (10) is unsatisfiable, then the characteristic question of S5 is answered with NO, which renders q as not skeptically accepted (see Section 4.1).

Encoding of S6 In S6 we ask if there exists a complete labeling in which the query argument is labeled as OUT. We will base the encoding of this question again on the encoding for a non-empty complete labeling Ψ_{CO} (see Formula (9) in Section 2.2). Recall that in an empty labeling all arguments are labeled as UNDEC. Note that we search for a complete labeling in which q is labeled as legally OUT (see 2.1), which implies that at least one attacker of q has to be labeled as IN. It is therefore valid to search for only those complete labelings which are non-empty. We can encode the characteristic question of S6 as the following propositional formula:

$$\Psi_{S6} = \Psi_{\mathsf{CO}} \wedge O_{\mathsf{q}} \tag{11}$$

If Formula (11) is satisfiable, then there exists a truth assignment which sets O_{q} to TRUE, which means that the argument q is labeled as OUT in the associated complete labeling. Hence q is not skeptically accepted (see Section 4.1).

Encoding of S7 S7 raises the question if there exists a complete labeling in which at least one victim of q is labeled as IN. Note that we can compute the set of victims $\{q\}^+$ in polynomial time by checking the attack relations of a given framework. To encode the characteristic question of S7 we use Ψ_{CO} (see Formula (9) in Section 2.2) as a basis. The non-emptiness of Ψ_{CO} aligns well with the intuition to search for those labelings in which at least one argument of $\{q\}^+$ is labeled as IN. In order to correctly solve the shortcut we have to check at first if the query argument q attacks any other argument. If q does not attack another argument, then S7 cannot assess the skeptical acceptance of q. If q does indeed attack other arguments, then we proceed with the following formula that encodes the characteristic question of S7:

$$\Psi_{S7} = \Psi_{\mathsf{CO}} \wedge \Big(\bigvee_{b \in \{q\}^+} I_b \Big) \tag{12}$$

If Formula (12) is satisfiable (and q attacks other arguments in the framework), then there exists a complete labeling which labels a victim of q as IN. In this case q cannot be skeptically accepted, because it attacks an admissible set (see Section 4.1).

Encoding of S8 To conclude the skeptical acceptance from S8, we need to check if there exists a complete labeling, in which all defenders against at least one attacker of q are labeled as OUT. Computing the set of defenders for each attacker of the query argument can be done in polynomial time by iterating through the attack relations of the given argumentation framework. It should be noted that attackers, which attack themselves, are not eligible as defenders in the sense used for this shortcut. These arguments are not able to defend the query argument.

To answer this question correctly with the hereby proposed encoding we have to take three special cases into account: the case of an unattacked query argument, the case of an unattacked attacker of q, and of a self-attacking query argument that has no other attackers. In the first special case there is no argument attacking q. Since there are no attackers, there can be no labeling with the defenders labeled OUT. We conclude that S8 cannot solve the problem for this instance and that we cannot draw any conclusion about the acceptance of q (at least not by answering the characteristic question of S8).

In the second special case there is an unattacked argument which is attacking q. It means that there are no defenders against this attacker. In this case the question has to be answered with YES, which lets us conclude that the argument q is not skeptical accepted.

The third special case deals with those problem instances in which the query argument attacks itself and has no other attackers. Because the query argument will always be UNDEC in this case we can conclude that the argument cannot be skeptically accepted.

We can catch these three special cases in polynomial time before checking the satisfiability of the encoded formula of the characteristic question.

To translate the characteristic question to a propositional formula we use the encoding associated with nonempty complete extensions Ψ_{CO} of Formula (9) in Section 2.2. The requirement of non-emptiness does not hinder our solving process. In all cases left to consider there are attackers of q and for each attacker at least one defender. Therefore, we are only interested in labelings which label the defenders as legally OUT, which implies that at least one argument has to be labeled as IN. The following propositional formula encodes the characteristic question of S8:

$$\Psi_{S8} = \Psi_{\mathsf{CO}} \wedge \left(\bigvee_{b \in \{q\}^-} \left(\bigwedge_{d \in \{\{b\}^- \setminus \{b\}\}} O_d \right) \right) \tag{13}$$

Formula (13) becomes TRUE iff for at least one attacker b in $\{\mathtt{q}\}^-$ it holds that for all arguments d attacking b the variable O_d is set to TRUE.

However, Formula (13) is not in Conjunctive Normal Form (CNF). In order to use this formula as an input for a SAT-solver, we need to transform the formula to an CNF with equivalent satisfiability. Note that equivalence in satisfiability is sufficient here, because for this specific shortcut we are only interested in the existence of a complete labeling with the properties described above. We do not have to calculate the labeling itself. Let $n = \mid \{\mathtt{q}\}^- \mid$ be the number of attackers of q. We enumerate each attacker b_i of q with $i = 1...n$ and introduce the propositional constants $\mathrm{Z} = \{z_1, ..., z_n\}$.

The following formula in CNF has an equivalent satisfiability as Formula (13):

$$\Psi'_{S8} = \Psi_{\mathsf{CO}} \wedge \left(\bigvee_{z \in \mathrm{Z}} z\right) \wedge \left(\bigwedge\nolimits_{i=1}^{n} \left(\bigwedge_{d \in \{\{b_i\}^- \setminus \{b_i\}\}} (\neg z_i \vee O_d)\right)\right) \tag{14}$$

We have chosen this method of transformation into a satisfiability equivalent formula in CNF over other techniques such as a Tseitin transformation [34], because there is no risk of exponential blow up of the formula and other methods seem to result in larger formulas in our use case.

To sum it up, if there are arguments attacking q and for each attacker of q exists at least one defender (and q is not only attacked by itself), and Formula (14) is satisfiable, then the characteristic question is answered with YES, which means that q is not skeptically accepted (see Section 4.1).

Encoding of S9 For shortcut S9 we ask if a non-empty complete labeling exists, and in case that it does, if in at least one of those non-empty complete labelings q is labeled NOT IN. We will use two separate formulas for S9, each answers one of the two parts of this characteristic question. The first part is the question if a non-empty complete labeling exists. The propositional formula for this question is Ψ_{CO} of Formula (9). We discussed its application in Section 2.2. If this formula is unsatisfiable, which means there are no nonempty complete labelings, then S9 cannot solve the problem instance. In case that the formula is satisfiable we proceed to the next part of the question.

The second part of the characteristic question of S9 asks for a nonempty complete labeling which labels q as NOT IN. Again we use Ψ_{CO} of Formula (9) in Section 2.2 as a base to ask for non-empty complete labelings. We enrich this formula by the following clauses to encode the second part of the characteristic question of S9:

$$\Psi_{S9} = \Psi_{\mathsf{CO}} \wedge \neg I_{\mathtt{q}} \tag{15}$$

In case that Ψ_{CO} is satisfiable, but Formula (15) is not, we can conclude that all complete labelings label q as IN, hence q is skeptically accepted (see Section 4.1).

Encoding of S10 The shortcut S10 solves problem instances by asking if a non-empty complete labeling exists, and in case that it does, if there is a complete labeling with settled self-defense that labels q as NOT IN. Labelings with settled self-defense are those labelings which label each self-defending argument as either IN or OUT (see Definition 17 in Section 4.1). Note that the set of self-defending arguments, which we denote with S_{SD}, can be computed in polynomial time by checking the attack relations of the argumentation framework of the problem instance. As in S9 we use Ψ_{CO} of Formula (9) in Section 2.2 to check if a non-empty complete

labeling exists. If Ψ_{CO} is unsatisfiable, then we cannot solve the given problem instance using S10. If Ψ_{CO} is satisfiable, then we proceed by making a second call to the SAT-solver. This time we ask for a complete labeling with settled self-defense that labels q as NOT IN. We use the following clause to limit our search to those labelings of settled self-defense:

$$\Psi_{SD} = \bigwedge_{b \in S_{SD}} (I_b \vee O_b) \tag{16}$$

Formula (16) becomes only TRUE iff all arguments in S_{SD} are either labeled IN or OUT. Using Formula (16) we can know formulate the second part of the characteristic question of S10 as a propositional formula:

$$\Psi_{S10} = \Psi_{\mathsf{CO}} \wedge \Psi_{SD} \wedge \neg I_{\mathsf{q}} \tag{17}$$

Again, if Ψ_{CO} is satisfiable but Formula (17) is not, we can conclude that all preferred labelings label q as IN, hence q is skeptically accepted (see Section 4.1).

4.3 Implementation of the Shortcuts

To further investigate the capabilities of the mentioned shortcuts, we implement each shortcut as a single program, which we can then evaluate independently in further experiments.

Definition We implement each shortcut as an individual program that takes a single problem instance as input. We recall that a problem instance consists of an abstract argumentation framework and a query argument (see Definition 14 in Section 2.1). The program returns 3 different answers: YES, NO and UNKNOWN. The program returns YES iff the implemented shortcut could conclude that the query argument of the problem instance is skeptically accepted under the preferred semantics in the argumentation framework of the problem instance. Iff the shortcut concluded that the query argument is not skeptically accepted, then the program returns NO. In case that the shortcut could not decide whether the query argument is accepted or not, the program returns UNKNOWN.

Design The aforementioned programs were implemented in the program language C11, a variant of the program language C. Most of the code was written in C++, an object-oriented extension of C. Using C11 enabled using proven and reliable data structures to model and process the problem instance. To model an abstract argumentation framework, each argument was encoded as an integer ranging from one to the number of arguments in the framework. The attack relations of the framework were represented in two-dimensional arrays, called "vectors" in C11. One such "vector" Arr_{atck} was created to access the attackers of each argument. Each entry a_i of Arr_{atck} contains a list of arguments which attack the argument i in the framework. A similar vector is created listing the victims of each argument of the framework. This design allows direct access ($O(1)$) to both the attackers as well as the victims of an argument. This direct data access is very efficient for the encoding of e.g. complete labelings, because such an encoding requires to iterate over attack relations of specific arguments (see Section 2.2). Furthermore can such a direct access be used to efficiently check the existence of a particular attacker of an argument or if an argument has any attackers at all. Since we use at least one of these operations for all described shortcuts, we will use the same data model for all implemented shortcuts. Of course it is worth noting that one trade-off of such a data-model is the larger space needed to store the data-structure, compared to a single-list data model for example. However, during

the conduction of the experiments we could not observe any problem of running out of space. To restrict the space needed to represent the argumentation frameworks we represent each argument as an integer of type "uint32t". The type "uint32t" represents an unsigned integer with a fixed number of 32 bits. It restricts the programs to problem instances with 2^{32} arguments at maximum, which is well-above the maximum number of arguments in the used datasets.

In case of the shortcuts S5 to S10 the problem instance gets encoded as a propositional formula according to the characteristic question of the shortcut (see Section 4.2). To solve the SAT-problem of these formulas the SAT-solver Kissat [19] version 4.0.3 was used. Kissat was the fastest SAT-solver in the latest competition for SAT-solvers in 2024 [22] [4]. To use the Application-Programming-Interface (API) of Kissat, all literals of the propositional formula were represented as signed integers. Negative integers represent negated propositional constants.

The implementation of the algorithmic shortcuts is open source and available on GitHub[5].

In this chapter we described the algorithmic shortcuts, which we investigate in this thesis. In addition to the shortcuts known from the literature, we proposed new shortcuts and showed that all shortcuts soundly solve the problem of DS-PR. We showed how the characteristic questions of these shortcuts can be encoded as propositional formulas and discussed how these shortcuts can be implemented as single programs.

5 Empirical Evaluation

Many state-of-the-art argumentation solvers use algorithmic shortcuts (see Section 3). The example of μ-TOKSIA (see Section 3.7) seems to indicate that not all these shortcuts really improve the overall runtime efficiency of the solvers. It is therefor of interest to investigate the impact of the different shortcuts on the performance of a solver. In order to improve the runtime behavior of a solver a shortcut needs to be computed efficiently and solve many problem instances. Especially the second point has not been explored systematically yet. It leads to the following questions for each of the proposed algorithmic shortcuts:

- RQ1 How many times is the algorithmic shortcut able to solve a problem instance of the used datasets?
- RQ2 How much time does it take for the algorithmic shortcut to return a result for a problem instance of the used datasets?
- RQ3 How much faster can the algorithmic shortcut solve a problem instance of the used datasets compared to known complete algorithms from the literature?

In this section we describe the setup, conduction, and results of the experiments used to assess the potential of the shortcuts to improve the runtime efficiency of solving approaches for skeptical reasoning under the preferred semantics.

5.1 Setup

To describe the setup of the experiments we will first discuss the datasets of problem instances which were used for the evaluation. After that we discuss the application Probo2 [26], which was used to conduct the experiments. Finally we conclude this section with a quick overview of the computer system on which the experiments were calculated.

[4] https://satcompetition.github.io/2024/results.html
[5] https://github.com/jlianSander/ascDSPR

Datasets We use the datasets known from the ICCMA competitions to answer the research questions RQ1 to RQ3. In the following paragraphs we use the term "ICCMA datasets" to describe the sets of problem instances of all ICCMA competitions from 2015 to 2023 for the reasoning problem DS-PR. These datasets have been assembled to assess the problem solving capacities and efficiencies of argumentation solvers. We make use of their variety of problem instances to cover as many different problem instances as possible. Some of the key statistics of these datasets have been summarised in Table 2.

There are 1677 problem instances in total in all used data sets. In the competition ICCMA'17 we have the special case that 50 frameworks were reused in a secondary problem instance with a different query argument. It is the only competition in which the number of problem instances differs from the number of frameworks. We noted the number of problem instances in parenthesis. It is also worth noting that the ICCMA'21 dataset contains problem instances with frameworks of particularly large size compared to all other considered datasets. Furthermore, we see in Table 2 that the dataset of ICCMA'23 has a substantially higher average number of attacks (Avg. $|R|$) and average node degree than the sets used in the years before 2021. It means that on average a single argument was part of more attack relations, which indicates that the complexity of the frameworks in ICCMA'23 seems to have been higher on average than those before 2021.

Dataset	#AFs	Avg. $\|A\|$	Med. $\|A\|$	Std. $\|A\|$	Avg. $\|R\|$	Avg. D	#YES	#NO
ICCMA'15	192	1,980	675	2,424	105,396	68	14	178
ICCMA'17	300 (350)	14,544	474	132,416	311,277	245	36*	293*
ICCMA'19	326	826	196	1,784	97,639	239	39	287
ICCMA'21	480	87,331	48,200	92,881	7,239,611	161	0*	394*
ICCMA'23	329	29,791	796	203,719	1,002,470	299	32*	273*

Table 2: Statistics for all considered datasets (extended table from [5]). #AFs describes the number of instances in the dataset; $|A|$ describes the number of arguments per instance and the table displays its average, its median, and its standard deviation; Avg. $|R|$ describes the average number of attacks in an instance of the dataset; Avg. D describes the average node degree of the argumentation frameworks in the dataset. #YES and #NO represent the number of instances in which the query argument is skeptically accepted or not. '*' denotes that not for all instances a solution was ascertainable.

Table 2 also shows in the columns #YES and #NO how often a query argument was skeptically accepted or rejected in the problem instances of a benchmark.

We could not ascertain the solutions for all instances in the data sets ICCMA'17, ICCMA'21 and ICCMA'23, what is marked with an asterisk. Usually the solutions to the problem instances are concluded by solving the problem instances with a proven argumentation solver. Since ICCMA'21 this benchmark solver has been μ-TOKSIA [9]. It is good practice to apply the benchmark solver with a time limit for the computation of a single instance that is ten times higher than the time limit used for the competition, which has been 1200 seconds in the years 2021 and 2023 [9, 25]. Nevertheless, applying a timeout of 12 000 seconds did not suffice on the used Intel Xeon E5 with 3.4 GHz CPU to calculate answers for all instances of the datasets ICCMA'17, ICCMA'21 and ICCMA'23.

Looking at Table 2 we can see a clear bias with more instances solved with NO than with YES. The number of known problem instances that are solved with YES is quite low, in total 121

of 1677 (7%), compared to the known problem instances solved with NO, 1425 of 1677 (85%). If we recall the general idea that proving the skeptical acceptance means to check that a query argument is contained in every preferred extension, we see why solving a problem instance with solution YES is often more time-consuming than taking the same framework and asking about an argument that is not skeptically accepted. That is why we presume that among those problem instances for which we could not compute a solution, there are more problem instances with solution YES than NO.

Probo2 - Software Setup The evaluation of the research questions over the ICCMA datasets is done with the help of the program PROBO2 version 1.1 [26][6]. The program allows to compare argumentation solvers by evaluating their runtime efficiency on datasets of argumentation frameworks. To evaluate a solver, PROBO2 iterates through the different problem instances of a given dataset. For each problem instance PROBO2 asks the solver to compute a solution to the problem instance and takes the time that the solver needs for the computation. In compliance with the rules of the ICCMA competitions, PROBO2 aborts the calculation of a solver, if the runtime of the computation exceeded a time limit.

Computer System - Hardware Setup All experiments were conducted under Ubuntu 20.04 using an Intel Xeon E5 3.4 GHz CPU and up to 192 GB of RAM. A time-limit of 1200 seconds was set for the computation of a single problem instance, which is equivalent to the rules of the most recent ICCMA in 2023.

5.2 Experiment Descriptions

In the following section we will discuss how the answers to the proposed research questions RQ1 to RQ3 can be empirically evaluated.

RQ1 The first research question points at the applicability of the different proposed algorithmic shortcuts of this thesis. We ask how often a given shortcut can solve a problem instance of the ICCMA datasets. The more problem instances the shortcut can solve, the more often it can improve the runtime efficiency of the argumentation solver and the less often it slows the solver down.

To answer this question we implement each shortcut as a program described in Section 4.2. These programs implement the same interface as the complete argumentation solvers of the ICCMA competitions. Therefore we can use the tool PROBO2 to iterate through the ICCMA datasets and count the number of problem instances in which the program returned an answer that is not UNKNOWN. The settings for this experiment are summarised in Table 3.

Test objects	S1 to S10
Metric	#instances for which program output not "UNKNOWN "
Datasets	ICCMA'15 - ICCMA'23
System	Intel Xeon E5 3.4 GHz CPU 192 GB RAM

Table 3: Experiment to answer RQ1

[6] https://github.com/aig-hagen/probo2

RQ2 The second research question focuses on the efficiency of the algorithmic shortcuts. For this question we take the time that passes from calling the shortcut to receiving a result from the shortcut function. In case that the shortcut solved the problem instance it will return its judgment if the query argument is skeptically accepted or not. If the shortcut was not able to solve the problem, then the result consists of a signal indicating the inability of the shortcut.

To measure the runtime of a algorithmic shortcut, we implement the shortcut as a program described in Section 4.2. For each problem instance we record the time from calling the program to receiving an answer (T_{Si}). Again we use the tool PROBO2 to iterate through all problem instances of the ICCMA datasets. We denote the program runtime in seconds of an algorithmic shortcut Si as T_{Si}. Table 4 summarises the settings of this experiment.

Test objects	S1 to S10
Metric	T_{Si} for $i = 1..10$
Datasets	ICCMA'15 - ICCMA'23
System	Intel Xeon E5 3.4 GHz CPU 192 GB RAM

Table 4: Experiment to answer RQ2

RQ3 The third research question examines the potential of an algorithmic shortcut to actually improve the runtime efficiency of an argumentation solver. We answer this question by comparing the performance of an algorithmic shortcut with the performance of a complete argumentation solver, namely μ-TOKSIA in its release version ICCMA'23 [7].

Choosing μ-TOKSIA as a benchmark for a complete solver for DS-PR has several reasons. The first reason is that μ-TOKSIA ranked first place in the last two ICCMA competitions in 2023 and 2021 [25, 9] and is therefore the most efficient implementation of a complete algorithm to solve DS-PR that has taken part in ICCMA. What is even more important for our experiments is the fact that μ-TOKSIA does not apply any algorithmic shortcuts. It only uses a complete algorithm, which is very close to the original idea of the CEGAR approach.

For a fair comparison of a shortcut and a complete solver such as μ-TOKSIA we have to account the fact that the shortcut cannot solve each problem instance. Simply comparing the runtimes of both test subjects over all instances of the ICCMA datasets would be unjust. Imagine a shortcut that does not calculate anything but rather directly replies the answer "UNKNOWN ". This shortcut would be extremely fast, but useless.

There are two ways to create a meaningful comparison. We can either restrict the comparison to those instances for which the shortcut was able to compute a solution or we design a more robust metric. Restricting the set of evaluated instances to the subset of those instances for which all solving approaches could calculate a solution allows a fair comparison of those approaches. However, the significance of such a comparison can be quite limited by the small number of instances of the considered subset. That is why we also need to think of a way to compare an algorithmic shortcut with a complete solver over all instances of the ICCMA data sets.

To come to a more meaningful comparison we design a reasonable use case for algorithmic shortcuts. In this use case we apply consecutively first the algorithmic shortcut and then a

[7] The newest version of μ-TOKSIA (version 2025.06.06) with the SAT-solver CADICAL v. 2.1.3, showed a less efficient runtime behavior on the used data sets than version ICCMA'23.

complete solver to solve the problem instance. In case that the algorithmic shortcut was able to solve the problem the use case finishes prematurely and returns the solution of the problem. If the shortcut was not able to solve the problem, then we start the complete solver after the shortcut has finished to finally solve the problem. Due to the linear character of this process of applying one solving approach after the other until the problem is solved we call this strategy a *cascading combination* of solving approaches. The runtime of such a cascading combination is either the runtime of the shortcut alone or the sum of the runtime of the shortcut and the runtime of the complete solver.

The following definition generalizes the cascading combination to a chain of several consecutively applied shortcuts.

Definition 18. *Let $S_S = (S_1, S_2, ..., S_n)$ be a series of algorithmic shortcuts and $T_{Si}(p)$ be the runtime in seconds that the ith shortcut in the series needs to return an answer to a given problem instance p. Let $T_c(p)$ be the runtime in seconds that a complete argumentation solver C needs to calculate an answer for the problem instance p. Let CC be the cascading combination of S_S and C. We define the runtime of the cascading combination as $T_{CC}(p)$*

$$T_{CC}(p) = \begin{cases} \sum_{S_i \in S_S} T_{Si}(p) + T_c(p) & S_i(p) = \text{UNKNOWN} : \forall S_i \in S_S \\ \sum_{i=1}^{k} T_{Si}(p) & S_i(p) = \text{UNKNOWN} : i = 1..k-1 \end{cases} \tag{18}$$

The runtime of the cascading combination is slower than its complete solver if none of its shortcuts is able to solve the problem instance. It is a way to "punish" failed attempts of the shortcuts to solve a problem instance. Furthermore it describes a realistic use case for a serial computation.

Of course this model of applying shortcuts underlies the assumption that we cannot calculate several shortcuts at the same time. In case that we use a computer with multiple cores it is conceivable that every core computes one shortcut, by which the runtime comes down to the minimum runtime of all shortcuts and the complete solver. Furthermore we neglect with this model the possibility that shortcuts or solvers can learn from the failed attempts of preceding shortcuts.

For this experiment we will use cascading combinations of a single shortcut and the complete solver μ-TOKSIA. We can then compare the runtime of these combinations with the runtime of μ-TOKSIA alone on all instances of the ICCMA datasets.

Table 5 summarizes the settings of this experiment.

Test objects	S1 to S10, μ-TOKSIA
Metric	$T_{CC(Si,\mu\text{-TOKSIA})} - T_{\mu\text{-TOKSIA}}$ for $i = 1..10$
Datasets	ICCMA'15 - ICCMA'23
System	Intel Xeon E5 3.4 GHz CPU 192 GB RAM

Table 5: Experiment to answer RQ3

5.3 Results

We will discuss the results of our empirical evaluation following the order of the research questions. We will first discuss how many instances the different algorithmic shortcuts could solve

(RQ1), what is called the applicability of the shortcuts. We will then discuss the overlap between the solved instances of the different shortcuts. In answer to RQ2 we will then take a look at the runtimes of the shortcuts for those instances they could solve. We will set these runtimes in relation to the runtimes that μ-TOKSIA needed on the same subsets of problem instances. Finally we will discuss different cascading combinations of shortcuts and μ-TOKSIA. We will compare the runtime of these combinations to the runtime of μ-TOKSIA over all instances of the ICCMA data sets. This comparison answers the third and last research question RQ3 and concludes this section.

The instances of the ICCMA datasets can be split into two disjunct groups according to their solution. An instance of these datasets is either correctly solved by returning the answer YES or by returning the answer NO. In the following we will refer to these two groups as YES-instances and NO-instances.

To validate our experimental results, we compared the answers returned by the different solving approaches to the known solutions of the ICCMA data sets. All solving approaches returned the correct answer if they were able to solve a problem instance.

Applicability Because of the incompleteness of algorithmic shortcuts it is of particular interest to investigate how many different problem instances they were able to solve. Table 6 and Table 7 give an overview of the number of problem instances that each algorithmic shortcut was able to solve. By design all algorithmic shortcuts in this thesis can only solve either NO-instances or YES-instances but not both types of instances. That is why we grouped the results of the applicability in two tables.

	ICCMA'15	ICCMA'17	ICCMA'19	ICCMA'21	ICCMA'23	Total	#TO
S1	26	45	57	0	0	128	0
S4	84	68	119	0	62	333	1
S5	175	194	149	356	161	1,035	32
S6	123	201	284	450	216	1,274	67
S7	71	164	224	443	151	1,053	76
S8	126	200	284	448	215	1,273	72
μ-TOKSIA	178	273	287	377	267	1,382	174
solution	178	293	287	394	273	1,425	0

Table 6: Applicability over the NO-instances of the data sets. The table displays the number of instances solved by each solving approach.

Table 6 and Table 7 show for each solving approach (row) how many problem instances they could solve in each data set (columns), followed by the total number of all solved instances over all data sets in the column 'Total'. Note that in the column 'Total' we also count problem instances for which there is no solution known, because μ-TOKSIA could not solve these instances with a timeout of 12 000 seconds. The last column '#TO' shows the number of problem instances for which the calculation was aborted because the time limit was reached.

In Table 6 we can first note that no solving approach, including μ-TOKSIA, was able to compute the solutions to all 1677 problem instances in time. Of course the incompleteness of the shortcuts S1-S8 prevents them by design to solve all instances. For example S1 could only solve 9% of the problem instances for which we know a solution. Because the computation of S1 never reached the time limit, we can conclude that the limited applicability of S1 is purely due

to its characteristic question, which is easy to see if we recall that S1 simply asks if the query argument is attacking itself. The poor applicability of this shortcut is also due to the design of the data sets. Simple solving strategies like S1 and S4 have been anticipated by the designers of the ICCMA competitions and easy problem instances were excluded from the data sets in order to challenge the argumentation solvers [25, 9].

For the other solving approaches we can see in the last column of Table 6 that some of their computations were aborted, because they took longer than the allowed time limit. In case of μ-TOKSIA this is the only reason why the complete solver was not able to compute all solutions. Main reason for this inefficiency is the usage of a SAT-solver as a NP-oracle. The same problem applies for the shortcuts S5 to S8, which do also use a SAT-solver. Interestingly even shortcut S4, which calculates the grounded extension and applies an algorithm of polynomial time, timed out at the instance 'admbuster_2000000', which is an extremely large framework with two million arguments.

The usage of a SAT-solver does however improve the applicability of shortcuts significantly. The most applicable shortcut of this group S6 uses the SAT-solver to ask for a complete labeling in which the query argument is labeled OUT. This shortcut could solve 1274 of the 1677 problem instances (76%) and found 1205 (85%) of the 1425 known solutions.

If we look at the numbers of solved instances per benchmark we note that some shortcuts solved more instances than the known solutions of this benchmark, see for example S8 in ICCMA'21. We recall that the solutions to the data sets have been calculated using μ-TOKSIA with a time out of 12 000 seconds. These results show that the shortcuts can calculate answers for instances for which μ-TOKSIA is not able to solve these instances even with ten times more computation time.

	ICCMA'15	ICCMA'17	ICCMA'19	ICCMA'23	Total	#TO
S2	5	7	16	0	28	0
S3	8	32	34	19	93	1
S9	9	35	36	30	110	37
S10	9	37	37	32	115	37
μ-TOKSIA	14	36	39	32	121	174
solution	14	36	39	32	121	0

Table 7: Applicability over the YES-instances of the data sets. The table displays the number of instances solved by each solving approach.

The results in Table 7 show a similar picture to those in Table 6. It is worth noting that the applicability of the shortcuts S9 and S10 is quite high with more than 90% of those problem instances solved, for which we know the solution. However, one should keep in mind that the overall number of YES-instances in the solutions of the ICCMA datasets is quite low. In fact we discussed in Section 5.1 that due to the complexity of solving YES-instances, we presume that among those problem instances for which we could not calculate the solution, there are more YES-instances than NO-instances. The complexity of solving YES instances comes mainly from the underlying CEGAR approach of μ-TOKSIA which was used with a time limit of 12 000 seconds to compute the solutions. The CEGAR approach iterates through all preferred extensions of a framework, before a solver comes to the conclusion that an argument is indeed skeptically accepted. The shortcuts S9 and S10 work with a different approach and only need two calls to the SAT-solver to decide about the acceptance of an argument. On the one hand,

comparing S9 and S10 to μ-TOKSIA with the same time limit, we can see that μ-TOKSIA was able to solve more problem instances, despite its CEGAR approach. On the other hand, it is interesting to see that S10 solved one problem instance in ICCMA'17 for which no solution was known, which means that μ-TOKSIA could not solve this problem instances with ten times more computation time.

Overlap To reasonably compare the runtime of different shortcuts it is of importance to regard the subset of those problem instances which could be solved by all shortcuts of the comparison (see Section 5.2). We call such a subset of problem instance which were solved by several solving approaches as their overlap.

Observing the overlap is also of interest if one wants to combine shortcuts in a cascading combination. There is no use of applying shortcut B consecutively after shortcut A, if shortcut A already solves all problem instances shortcut B would be able to. In other words, if shortcut A overlaps shortcut B by 100%.

Table 8 and Table 9 show the overlap between the different shortcuts and μ-TOKSIA. Since by design the overlap between a shortcut that solves NO-instances and a shortcut that solves YES-instances is 0, we showed the overlap for each group in a seperate table.

	S1	S4	S5	S6	S7	S8	μ-TOKSIA
#	128	333	1,035	1,274	1,053	1,273	1,382
S1	-	81	128	95	74	97	127
S4	81	-	333	333	192	333	325
S5	128	333	-	800	622	802	925
S6	95	333	800	-	1,044	1,263	1,183
S7	74	192	622	1,044	-	1,041	970
S8	97	333	802	1,263	1,041	-	1,183
μ-TOKSIA	127	325	925	1,183	970	1,183	-

Table 8: Overlap over the NO-instances. The table displays the number of instances solved by both solving approaches.

Table 8 shows the overlap between the different shortcuts that solve NO-instances. In the first row it shows the number of instances, which the shortcuts of the columns could solve. Beginning with the second row the table shows for each row how many of these instances the shortcuts of the rows could solve as well. For example, shortcut S4 in the third column solved 333 problem instances over all data sets. S1 in the second row could solve 81 of these 333 problem instances.

We can see in Table 8 that μ-TOKSIA, being a complete solver, has a high overlap over all other shortcuts (>89%). Yet again we see that the overlap never reaches 100%. The overlap is linked to the applicability, which we can see observing S6 for example. S6 has a very good applicability, which means that the shortcut can solve many problem instances. Hence it also solved many problem instances that other shortcuts were able to solve. That explains why the overlap of S6 is above 74% for all other solving approaches, including μ-TOKSIA with 86%. The overlap of S6 as well as the overlap of S5 over S4 is complete (100%). That is also no surprise if we look at the design of the shortcuts. S4 asks if the grounded extension attacks the query argument (see Section 4.1). S6 asks if there is an admissible set that attacks the query argument. We recall from Section 2.1 that the grounded extension is an admissible set. Therefore S6 will always be able to

solve all problem instances that S4 can solve. A similar relationship exists between S5 and S4. S5 asks if a complete extension exists that contains the query argument. We recall from Section 2.1 that the grounded extension is the minimal complete extension of a framework. Hence if an argument is attacked by the grounded extension, then it is attacked by all complete extensions and therefore it cannot be contained in any complete extension. It means that S5 will detect such a case and therefore solve each and every problem instance that S4 can solve. There will be also no complete extension containing the query argument, if the argument is attacking itself. That is why the overlap of S5 over S1 is again 100%. Note that this relation is not true the other way around, which can be seen by the poor overlap of S4 over S5 (32%) and S6 (26%).

Due to its design and the small number of solved instances overall, the overlap of S1 over the other solving approaches is very small and never bigger than 24% (S4).

Another very interesting case is the shortcut S8. We can see again a 100% overlap over S4, which is no surprise. S8 asks if a complete labeling exists in which all defenders against at least one attacker of the query argument are labeled OUT. In case that the grounded extension attacks the query argument, it will also attack all defenders of the query argument because of its admissibility. The overlap of S8 over S6 however is with 99% lower than expected. Due to their design we expected S8 to completely overlap S6. In case that a problem instance can get solved by S6, then there is a complete labeling that labels the query argument q as legally OUT. This implies that there has to be at least one attacker of q that is labeled legally IN. Hence all arguments attacking the attacker have to be labeled OUT, which would imply that S8 can also solve the problem instance. Regarding the 1% of cases in which S8 could not solve a problem instance we refer to the number of aborted calculation due to the time limit of the calculation shown in Table 6. S8 has slightly more timeouts than S6, which could explain the incomplete overlap. The overlap of S6 over S8 of 99% shows that those situations for which S8 was created, when S6 cannot solve a problem instance but S8 does, are very few (10 in total).

	S2	S3	S9	S10	μ-TOKSIA
#	28	93	110	115	121
S2	-	28	28	28	28
S3	28	-	93	93	93
S9	28	93	-	110	110
S10	28	93	110	-	114
μ-TOKSIA	28	93	110	114	-

Table 9: Overlap over the YES-instances. The table displays the number of instances solved by both solving approaches.

The results shown in Table 9 clearly reflect the logical implications of the design process. We see that S10 completely overlaps S9 and that S9 complete overlaps S3, which completely overlaps S2. S2 asks if the query argument is unattacked. If so then the query argument is contained in the grounded extension which is exactly what S3 asks for. We recall that the grounded extension is the minimum complete extension. Which means if the query argument is contained in the grounded extension, then it is contained in all complete extension, hence there is no complete labeling that does not contain the query argument, which is what S9 asks for. And if there is no complete labeling that labels q as NOT IN, there is also no complete labeling with settled self-defense that does label q as NOT IN, which is the characteristic question of S10 (see Section 4.1). We can also see that there are five problem instances (from 115) that S9 cannot solve but

S10 does. Compared to S3 the new proposed shortcut S10 could increase the applicability by more than 20% (22 instances in total). This improved applicability is not a huge effect, but still of interest in light of the small number of YES-instances overall.

Runtime of solved Instances To improve the runtime efficiency of argumentation solvers, shortcuts do not only need to solve many problem instances, they also need to do it fast. That is why in this paragraph we compare the runtime of each shortcut with the runtime of μ-TOKSIA. To ensure a fair comparison we will consider only those problem instances for the comparison that could be solved by both test subjects, the shortcut and μ-TOKSIA (see Section 5.2). It means that for each pair of a shortcut and μ-TOKSIA, we will observe the runtime on a different subset of instances.

	RT	RT μ-TOKSIA	#VBS	#
S1	49.03	1,011.87	127	127
S2	1.35	2.58	23	28
S3	7.12	11.38	73	93
S4	284.63	351.74	307	325
S5	15,657.71	85,236.32	816	925
S6	54,971.19	96,027.46	931	1,183
S7	53,884.31	91,833.91	727	970
S8	57,860.26	93,977.10	926	1,183
S9	52.74	69.07	81	110
S10	56.19	69.24	96	114

Table 10: Runtime comparison of the different shortcuts with μ-TOKSIA. Cumulated runtime (RT) is measured in seconds. # marks number of instances of the comparison and #VBS number of those in which the shortcut contributed to the Virtual Best Solver.

The first column of Table 10 shows for each shortcut the sum of its runtime followed by the sum of μ-TOKSIA's runtime in the second column on their subset of solved instances. The third column shows how many instances the shortcut calculated faster than μ-TOKSIA and the last column shows the number of problem instances used for the comparison. Note that the term VBS stands for Virtual Best Solver. It describes the model of a theoretical solving approach that does always choose the fastest solving approach for a instance.

We can see in Table 10 that all shortcuts have been faster than μ-TOKSIA in computing the solutions. Shortcuts with the least reduction were S10 and S4 with 81% of the runtime of μ-TOKSIA. The rather bad result of S4 is surprising, since S4 does not call a SAT-solver, but applies a polynomial algorithm to calculate the grounded extension.

S1 has the highest reduction in runtime and only needs 5% of the runtime of μ-TOKSIA to solve the same problem instances. This result is no surprise. μ-TOKSIA uses a call to a SAT-solver to solve these problems, while S1 simply checks if the query argument is attacking itself. Due to the implementation of S1 this check can be done in O(1). Whats more surprising than the result of S1 is the result of S2. Again S2 does only check if the argument is attacked or not, which is an O(1) operation in the current implementation. Still μ-TOKSIA needs only about twice this time to solve these problem instances, making use of a SAT-solver. However, this comparison was only made over 28 problem instances which is less than 2% of all 1677 problem instances.

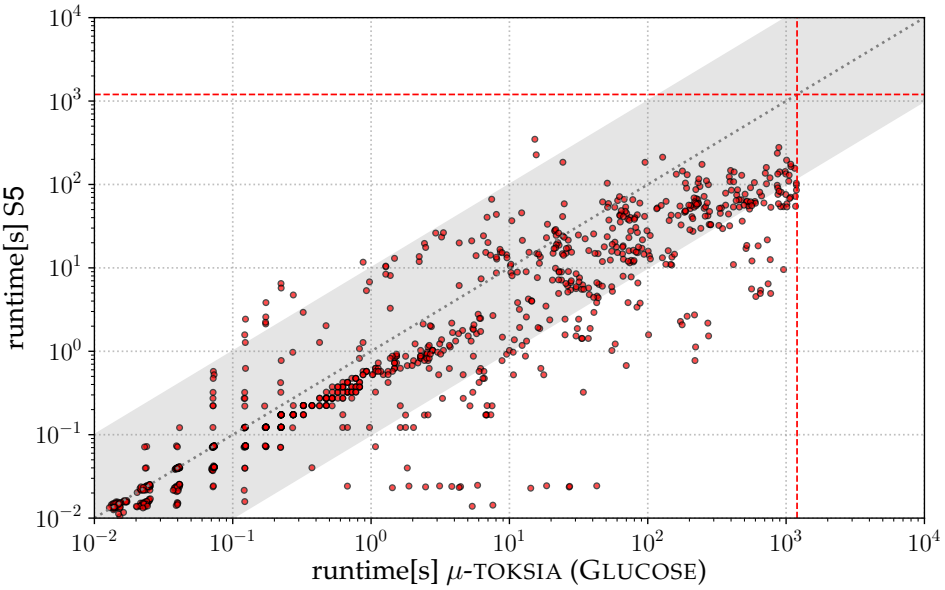

Figure 15: Scatter plot of the runtimes of single problem instances of the algorithmic shortcut S5 and μ-TOKSIA. Each problem instance is represented as one point. Points on the diagonal line indicate a similar runtime of both solving approaches for one problem instance. Points within the gray area differ only up to one magnitude from runtime equivalence. The red horizontal and vertical lines mark the time limit that was set for the computation of one instance.

A remarkable result is also the runtime of S5, which needs only 18% of the runtime that μ-TOKSIA requires to solve the same instances. In this case we considered 925 of the overall 1677 problem instances (55%). Figure 15 shows a scatter plot of the runtime behavior of the two solving approaches. We can clearly see that μ-TOKSIA has a higher runtime for most instances. There are also very few instances in which S5 has a significantly higher runtime, shown by the points above the gray diagonal area.

Runtime of all Instances To compare the runtime of a shortcut with μ-TOKSIA on only those instances that the shortcut could solve does not allow to assess if it is worth applying the shortcut before calling μ-TOKSIA. To consider also the failed attempts of a shortcut we create a cascading combination of each shortcut and μ-TOKSIA. The runtime of such a cascading combination is defined by the ability of the shortcut to solve the problem instance at hand (see Section 5.2). Because we want to compare these new combinations with μ-TOKSIA itself, we subtract for each problem instance the runtime of μ-TOKSIA from the runtime of the cascading combination. Doing so we receive the difference of the runtime, denoted as Δ. If Δ is positive, then the cascading combination needed more time than μ-TOKSIA to try to solve an instance. If Δ is negative then the cascading combination needed less time.

Table 11 shows for each solving approach plus the VBS the cumulated runtime of all instances in the first column. The second column shows the cumulated Δ over all instances. The third column shows the number of instances which could not be solved (#TO), because the runtime reached the time limit. In the forth column we can see the PAR2 score for each solving approach.

	RT	Δ	#TO	PAR2	#VBS
(S1, μ-TOKSIA)	320,445.51	386.96	173	314.12	193
(S2, μ-TOKSIA)	322,654.71	2,596.15	174	316.16	100
(S3, μ-TOKSIA)	325,940.27	5,881.71	175	318.82	145
(S4, μ-TOKSIA)	316,012.25	-4,046.31	167	307.18	347
(S5, μ-TOKSIA)	196,903.18	-123,155.37	72	141.32	589
(S6, μ-TOKSIA)	293,654.36	-26,404.19	87	184.27	478
(S7, μ-TOKSIA)	318,432.08	-1,626.47	98	201.01	349
(S8, μ-TOKSIA)	313,990.29	-6,068.26	92	197.53	590
(S9, μ-TOKSIA)	465,487.68	145,429.13	204	368.49	71
(S10, μ-TOKSIA)	462,930.49	142,871.94	203	366.65	84
(μ-TOKSIA)	320,058.55	0	174	315.36	369
VBS	90,478.90	-229,579.65	39	81.86	-

Table 11: Evaluation of the cascading combinations of a shortcut and μ-TOKSIA over all 1677 problem instances. Cumulated runtime (RT) and cumulated Δ are measured in seconds. #TO marks number of instances which exceeded the time-limit of computation and #VBS number of those instances for which the combination contributed to the Virtual Best Solver. PAR2 marks the PAR2 score for each solver.

The PAR2 score is the average runtime over all instances, with the particularity that the runtime of instances which reached the time limit are multiplied by two. This way we punish the solving approach for each problem instance it could not solve. The fifth column shows for how many instances the solving approach was the fastest approach. The Virtual Best Solver (VBS) always chooses the fastest solving approach for a problem instance. The fastest solving approach gets marked as a contribution to the VBS and its counter in #VBS gets increased. Note that solving approaches with a runtime within a 5% margin of the fastest solution found are also counted as contributions to the VBS. Doing so we dampen the effects of uncontrollable variations in the execution of the programs by the used computer system.

We can see in Table 11 that the cascading combination of S5 and μ-TOKSIA is 38% faster than applying μ-TOKSIA alone. This solving approach has with 72 also the smallest number of timeouts and consequently the lowest PAR2 score of all approaches. It offered one of the fastest methods to solve a problem instance in 35% of the cases. We see in the column of the PAR2 score of Table 11 that the cascading combinations using the shortcuts S5, S6, S7, and S8 show a clear improvement compared to μ-TOKSIA. We can draw the same conclusion by looking hat a cactus plot of the cumulated runtimes of all solving approaches in Figure 16.

In Figure 16 we sorted all instances according to their runtime in an ascending order for each solving approach. We then created a data point for each instance, marking on the x-axis their runtime and on the y-axis their number in the sorted list of instances. The red vertical line marks the time limit for the calculation. Note that we omitted the data points of the first 800 instances. There was no difference visible between the solving approaches. We also hide all instances which reached the time limit during calculation. As a result we see that the cascading combination of S5 and μ-TOKSIA has the steepest rise and is fastest in reaching the limit of solved problem instances. We can also see that the combinations of S6, S7 and S8 with μ-TOKSIA do also have a significantly better runtime behavior than μ-TOKSIA. The cascading combination

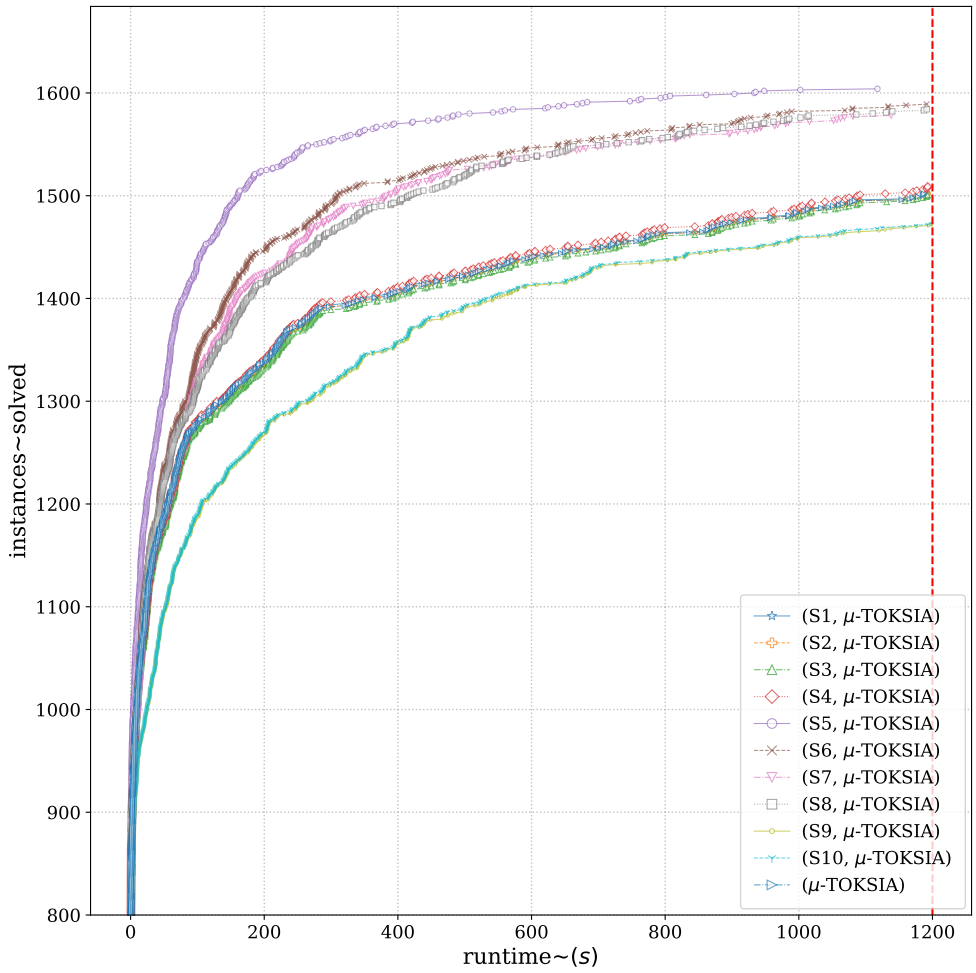

Figure 16: Cactus plot displaying the runtimes for solving single instances of each cascading combinations of a shortcut with μ-TOKSIA and μ-TOKSIA alone. Each data point represents the runtime of a solver for a single problem instance on the x-axis, and the number of instances with equal or faster runtimes of the solver on the y-axis.

of S4 and μ-TOKSIA has also a negative Δ in Table 11, but due to the high number of timeouts its PAR2 score is much worse than the scores of the best performing combinations. In Figure 16 the combination of S4 and μ-TOKSIA shows a similar performance as applying μ-TOKSIA alone and is one of several combinations in the middle field.

The cascading combinations of S9 and S10 with μ-TOKSIA are slower in solving DS-PR than μ-TOKSIA alone. An observation that can also be seen in Table 11. Applying S9 or S10 upstream of μ-TOKSIA slows the argumentation solver down by 45% compared to the cumulated runtime of μ-TOKSIA. This might be surprising if we look at Figure 17, in which we can see that S10 is faster than μ-TOKSIA in solving most instances. However, Figure 17 considers only the very small number of instances (114 of 1677) that the shortcut was able to solve. For the other 93% of the problem instances the cascading combination 'lost' runtime, because while S10 was trying

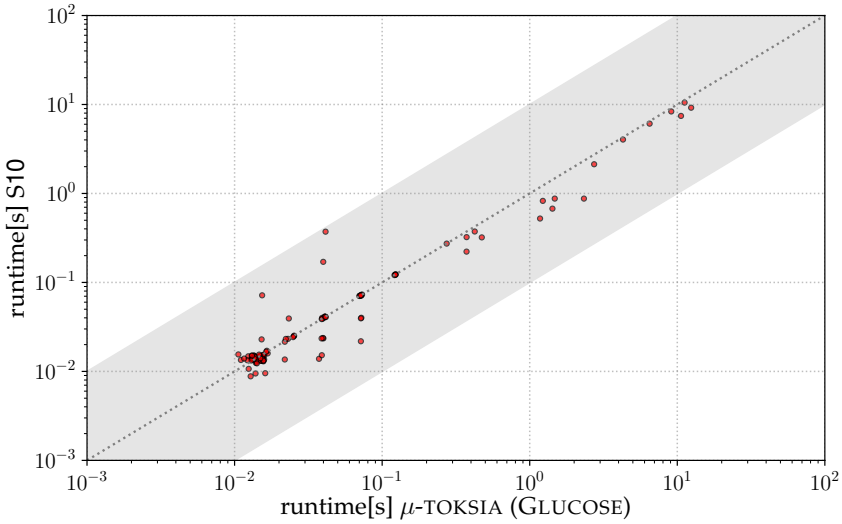

Figure 17: Scatter plot of the runtimes of single problem instances of the algorithmic shortcut S10 and μ-TOKSIA. Each problem instance is represented as one point. Points on the diagonal line indicate a similar runtime of both solving approaches for one problem instance. Points within the gray area differ only up to one magnitude from runtime equivalence. The runtimes of all evaluated instances of this subset are far from reaching the time limit of 1200 seconds.

to solve the problem instance in vain, the test subject that only applied μ-TOKSIA could already solve the problem instance.

At this point it should be noted that μ-TOKSIA is surprisingly fast in solving the known YES-instances. While it needs 80.26 seconds on average to solve the NO-instances, it only needs 2.76 seconds on average for YES-instances. Although the standard deviation is high in both cases (204.71 for NO and 14.2 for YES) the median of the runtime of the solved YES-instances (0.02 seconds) is also lower than the median of the runtime of the solved NO-instances (0.82 seconds). Again we have to keep in mind that there are much less YES-instances than NO-instances, but it is interesting to see that these YES-instances did not seem to be a big challenge for μ-TOKSIA, despite having to iterate through all preferred extensions. This might be an indication, that the solvable YES-instances of the data sets are too easy to solve, while those YES-instances which are presumably in the set of unsolved problem instances are too hard for any current solver to solve in a reasonable time.

Cascading several Shortcuts The model of cascading combinations allows to calculate the theoretical runtime of different combinations of shortcuts. A limit for the runtime reduction that can be achieved with these shortcuts is the runtime of the VBS. Its design always chooses the best approach for each problem instance and could be theoretically reached if all solving approaches were calculated in parallel. With only 39 time-outs and a PAR2 score of 81.88 the VBS shows an remarkable reduction compared to μ-TOKSIA in its original design.

If we focus on a purely sequential execution on only one processing unit, we can use the insights of the above observations to create cascading combinations of more than one algorithmic shortcut with μ-TOKSIA. Based on the results in Table 11 and Figure 16 it seems worth to look

at combinations using the shortcuts S5, S6, S7 and S8. Because the cascading combination of S5 and μ-TOKSIA showed the best results, we will use this combination as a benchmark. Regarding S6, S7 and S8 we notice that the overlap (see Table 8) between these shortcuts is significant. S7 overlaps S8 by 82%. Both S6 and S8 overlap S7 and each other by 99%. If we take the runtimes on the solved instances from Table 10 into account, we see that S6 needs 57%, S7 59%, and S8 62% of the runtime of μ-TOKSIA. This advantage in runtime efficiency of S6 compared to S7 and S8 is also visible in Table 12.

	S1	S4	S5	S6	S7	S8
S1						
S4	-0.50					
S5	11.32	-201.68				
S6	34.30	-201.73	30,990.32			
S7	4.21	90.88	31,762.59	2,025.34		
S8	30.94	-211.48	41,809.66	12,411.91	9,748.52	
μ-TOKSIA	962.84	67.11	69,578.61	41,056.27	37,949.60	36,116.85

Table 12: Δ of the cumulated runtimes in seconds in a pairwise comparison of different solving approaches on sets of NO-instances that both could solve

Table 12 shows a pairwise comparison of the cumulated runtimes of different shortcuts on the subset of problem instances that both shortcuts could solve. The table shows for each pair of shortcuts the Δ of their cumulated runtimes. The runtime of the shortcut in the row is subtracted by the runtime of the shortcut in the column. We can see that both S7 and S8 have a positive Δ to S6, which means that their runtime is higher than the runtime of S6 on the set of problem instance that both could solve.

Since S6 is the fastest of these three shortcuts and because it gets only overlapped by S5 by 63%, we choose to apply it after S5 in a cascading combination.

	RT	Δ	#TO	PAR2	#VBS
(S5, μ-TOKSIA)	196,903.18	-123,155.37	72	141.32	930
(S5, S6, μ-TOKSIA)	231,125.49	-88,933.06	59	124.10	1070
(S5, S6, S8, μ-TOKSIA)	280,450.00	-39,608.56	59	124.19	1070
(S5, S6, S3, μ-TOKSIA)	233,725.30	-86,333.26	60	126.23	1071
(S5, S6, S10, μ-TOKSIA)	250,422.82	-69,635.73	58	122.67	1072
(S5, S6, S3, S10, μ-TOKSIA)	253,026.96	-67,031.60	59	124.80	1071
(μ-TOKSIA,)	320,058.55	0	174	315.36	714

Table 13: Evaluation of some cascading combinations and μ-TOKSIA over all 1677 problem instances. RT marks the cumulated runtime in seconds of the solving approach. Δ denotes the difference to the runtime of μ-TOKSIA. #TO marks number of instances which exceeded the time-limit of computation and #VBS number of those instances for which the combination contributed to the Virtual Best Solver. PAR2 marks the PAR2 score for each solver.

Table 13 shows the results for a runtime comparison of some new cascading combinations

with μ-TOKSIA. We see that the combination of S5, S6 and μ-TOKSIA reduces the runtime over all instances less than the combination of only S5 and μ-TOKSIA (-28% compared to -38%). However, the number of timeouts is smaller, which leads to a better PAR2 score (124.1 compared to 141.32). Adding S8 after S6 in the cascade does further slow down the runtime and it does not improve the number of time outs. This is not surprising if we recall that due to the high overlap of S6 over S8 the probability that S8 can solve a problem instance that S6 could not is really low (around 1%). Both S5 and S6 can only solve NO-instances. Because the majority of problem instances are NO-instances it is only reasonable to apply shortcuts for NO-instances first, before trying to solve any YES-instances. We see in Table 13 that applying a shortcut for YES-instances after these capable shortcuts for NO-instances can have a further positive effect on the number of timeouts. Adding S10 to the combination of S5 and S6 increases the runtime over all instances, but it reduces the number of timeouts by one, which reduces the PAR2 score. Though this effect is not huge, it should be seen in light of the small number of solvable YES-instances in the data sets.

In this chapter we described the setup of the experiments used to answer the aforementioned research questions. We described the conduction of the experiments and discussed the results of the empirical evaluation.

6 Conclusion

Skeptical acceptance under the preferred semantics is one of the most intricate reasoning problems in abstract argumentation [17]. Hence there is great interest in runtime efficient solving approaches for this problem. Algorithmic shortcuts have been used in the past to improve the efficiency of argumentation solvers [24], but their effects have never been investigated in a thorough comparison. With this thesis we filled this gap by providing a runtime comparison of several algorithmic shortcuts that have been used in the biennial ICCMA competitions [9, 20, 25, 33]. We created, to our knowledge, the first summary of algorithmic shortcuts that have been used over the years in different argumentation solvers. In addition to the six shortcuts known from the literature we proposed four new shortcuts, which employ a SAT-solver as a NP-oracle. All those shortcuts make at most two calls to the NP-oracle. This limited and constant number of calls is particularly interesting in case of certifying skeptical acceptance of an argument. Using other solving approaches, like the well known CEGAR approach [14], the number of calls to the NP-oracle can grow polynomial in the worst case. CEGAR uses an iterative procedure, enumerating all possible counter-examples to certify skeptical acceptance. We showed the correctness for all algorithmic shortcuts and discussed the corresponding SAT-encodings, which are necessary to use a SAT-solver. Due to the non-completeness of algorithmic shortcuts, we designed the model of a cascading combination of a shortcut and a complete solver. The model represents a serial execution of the shortcut followed by the solver if needed to solve the problem. With this model we were able to conduct a fair and sound comparison of the runtime behavior of all algorithmic shortcuts and the complete solver μ-TOKSIA [28] over all problem instances.

The overarching goal of this thesis was to assess the effects of algorithmic shortcuts on the efficiency of the solving process for the reasoning problem DS-PR. The empirical evaluation of the different shortcuts showed huge differences in their effects on the runtime efficiency of a solving approach. While the shortcut S6 was the most applicable, solving 85% of the problem instances used for verification, the shortcut S5 was most efficient during computation, calculating the same results as μ-TOKSIA but in less than 18% of μ-TOKSIA's runtime. Both of these shortcuts could only solve NO-instances. The new proposed shortcut S10 was also slightly faster than μ-TOKSIA in solving YES-instances and needed less than 81% of the solver's runtime. S10 makes use of a relation between the self-defending arguments and the preferred extensions of an abstract argumentation framework, which we worked out in this thesis. The shortcut has a high applicability, solving 94% of the problem instances used for verification and is able to certify skeptical acceptance with only two calls to the NP-oracle.

In this thesis we could show that the gain in runtime efficiency of μ-TOKSIA compared to CEGARTIX [18] is not due to the fact that CEGARTIX applies S6 before using the same CEGAR-like algorithm as μ-TOKSIA. We showed that μ-TOKSIA would even profit from applying S6 with a PAR2 score reduction of about 42%. Focusing on a serial execution of shortcuts as components of an argumentation solver, we showed that both selection and arrangement of the shortcuts impact the runtime of the solver. A careful design of such a solver can half the PAR2 score compared to μ-TOKSIA. The best found solution in this thesis is a combination of first applying S5, then S6, S10 and finally μ-TOKSIA as the last step in this solving approach. This combination is significantly more efficient in terms of runtime compared to only using μ-TOKSIA, reducing the runtime by 22% and the PAR2 score by more than 60%. It shows that using algorithmic shortcuts before applying complete algorithms has the potential of increasing the runtime efficiency of an argumentation solver significantly. This potential is even higher if we think of a parallel execution of several shortcuts at the same time. We showed that based on our empirical data a theoretical reduction of the cumulated runtime of 72% compared to μ-TOKSIA is achievable.

Our data also shows that some shortcuts are able to solve problem instances within the time limit, that μ-TOKSIA was not able to solve even with a ten times higher time limit. The usage of a NP-oracle for an algorithmic shortcut increased their applicability significantly and did, to our surprise, not worsen the overall runtime efficiency. We could see in our results that the implementation of a polynomial algorithm to compute the grounded extension, used for shortcuts S3 and S4, performed worse than applying a SAT-solver to give answers that subsumed the information that could be drawn from the grounded extension for solving DS-PR, as used in the shortcuts S5 and S6.

Future work could explore the problem instances which the shortcuts could not solve more in detail. It could be interesting to develop new shortcuts that complement the ones proposed in this thesis, so that in the end a (serial) composition of these shortcuts can solve all problem instances of DS-PR. This could be particularly interesting for those problem instances which are hard to solve for CEGAR-like complete argumentation solvers. It would be also interesting to investigate more serial combinations of the discussed algorithmic shortcuts. The model of cascading combinations and the runtime that can be calculated for each problem instance allow an automated assessment of the many possible serial arrangements of the discussed algorithmic shortcuts in this thesis. There are also other more sophisticated models conceivable, which set the sequence of the shortcuts individually for each problem instance. One idea is to use inconsistency measures (see [1, 23, 27]) or other (graph) properties of the abstract argumentation framework to assess which shortcuts to use in which order for a given problem instance. The application of algorithmic shortcuts to improve the runtime efficiency of argumentation solvers follows the idea of using semantical information of a problem instance to solve the problem. In this regard it could be interesting to investigate if measures about the structure of a framework correlate with the applicability and efficiency of different shortcuts. The frequency of self-attacks for example could give hints about the probability that the shortcut S1 is able to solve a problem. More so, it could be interesting if further semantical information, e.g. extension-based measures, can help in choosing the most efficient order of shortcuts for a problem instance. Further research could also include the development of a argumentation solver that computes the different shortcuts in parallel. We have shown the huge potential to further increase the runtime efficiency this way and it would be interesting to see if our theoretical benchmark can be met. In this thesis we compared the shortcuts to the complete solver μ-TOKSIA. It would be also interesting to compare the performance of the different shortcuts with heuristic algorithms like in [31]. Doing so we could compare the shortcuts S5 and S6 with FARGO, a solver that uses a similar approach to solve a problem instance, but does not use a SAT-solver. The study in [31] includes the solver HARPER++. Comparing the shortcuts to this solver could confirm our results for S3 and S4, which use the same algorithmic idea. While the algorithmic shortcuts of this thesis make use of semantical information of the abstract argumentation framework, it would be interesting to compare these shortcuts with other heuristic algorithms which are more focused on the properties of the graph of an abstract argumentation framework, like ARIPOTER and AFGCNV2 (see [31]).

Though the reasoning problem DS-PR is highly intractable and one of the most difficult reasoning problems in abstract argumentation, this thesis shows that argumentation solvers to solve this problem are far from having reached their limits in runtime efficiency.

References

[1] Leila Amgoud and Jonathan Ben-Naim. Measuring disagreement in argumentation graphs. In Serafín Moral, Olivier Pivert, Daniel Sánchez, and Nicolás Marín, editors, *Scalable Uncertainty Management - 11th International Conference, SUM 2017*, volume 10564 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2017.

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

[3] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. Abstract argumentation frameworks and their semantics. In *Handbook of formal argumentation*, volume 1, pages 159–236. College Publications, 2018.

[4] Lars Bengel, Julian Sander, and Matthias Thimm. A reduct-based approach to skeptical preferred reasoning in abstract argumentation. In *Proceedings of the 22th International Conference on Principles of Knowledge Representation and Reasoning, KR 2025*, 2025.

[5] Lars Bengel, Julian Sander, and Matthias Thimm. A reduct-based approach to skeptical preferred reasoning in abstract argumentation (extended version). July 2025.

[6] Philippe Besnard and Sylvie Doutre. Checking the acceptability of a set of arguments. In James P. Delgrande and Torsten Schaub, editors, *10th International Workshop on Non-Monotonic Reasoning, NMR 2004*, pages 59–64, 2004.

[7] Philippe Besnard, Sylvie Doutre, and Andreas Herzig. Encoding argument graphs in logic. In Anne Laurent, Olivier Strauss, Bernadette Bouchon-Meunier, and Ronald R. Yager, editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems - 15th International Conference, IPMU 2014*, volume 443 of *Communications in Computer and Information Science*, pages 345–354. Springer, 2014.

[8] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

[9] Stefano Bistarelli, Lars Kotthoff, Jean-Marie Lagniez, Emmanuel Lonca, Jean-Guy Mailly, Julien Rossit, Francesco Santini, and Carlo Taticchi. The third and fourth international competitions on computational models of argumentation: Design, results and analysis. *Argument & Computation*, 16(2):236–299, 2025.

[10] Stefano Bistarelli, Lars Kotthoff, Francesco Santini, and Carlo Taticchi. Summary report for the third international competition on computational models of argumentation. *AI Magazine*, 42(3):70–73, 2021.

[11] Martin W. A. Caminada and Dov M. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145, 2009.

[12] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. How we designed winning algorithms for abstract argumentation and which insight we attained. *Artificial Intelligence*, 276:1–40, 2019.

[13] Federico Cerutti, Matthias Thimm, and Mauro Vallati. An experimental analysis on the similarity of argumentation semantics. *Argument & Computation*, 11(3):269–304, 2020.

[14] Edmund M. Clarke, Anubhav Gupta, and Ofer Strichman. SAT-based counterexample-guided abstraction refinement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(7):1113–1123, 2004.

[15] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.

[16] Paul E. Dunne and Trevor J. M. Bench-Capon. Coherence in finite argument systems. *Artificial Intelligence*, 141(1/2):187–203, 2002.

[17] Wolfgang Dvořák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. *Journal of Applied Logics- IfCoLog Journal of Logics and their Applications (FLAP)*, 4(8), 2017.

[18] Wolfgang Dvořák, Matti Järvisalo, Johannes Peter Wallner, and Stefan Woltran. Complexity-sensitive decision procedures for abstract argumentation. *Artificial Intelligence*, 206:53–78, 2014.

[19] Mathias Fleury and Armin Biere. Mining definitions in Kissat with Kittens. *Formal Methods in System Design*, 60(3):381–404, 2022.

[20] Sarah Alice Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran. Summary report of the second international competition on computational models of argumentation. *AI Magazine*, 39(4):77–79, 2018.

[21] Orna Grumberg, Assaf Schuster, and Avi Yadgar. Memory efficient all-solutions SAT solver and its application for reachability analysis. In Alan J. Hu and Andrew K. Martin, editors, *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004*, volume 3312 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2004.

[22] Marijn J.H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Proceedings of SAT competition 2024: Solver, benchmark and proof checker descriptions. 2024.

[23] Anthony Hunter. Measuring inconsistency in argument graphs. *Computing Research Repository*, abs/1708.02851, 2017.

[24] Matti Järvisalo, Tuomo Lehtonen, and Andreas Niskanen. *Solver and Benchmark Descriptions of ICCMA 2023 : 5th International Competition on Computational Models of Argumentation*, volume B-2023-3. Department of Computer Science Series of Publications B, 2023.

[25] Matti Järvisalo, Tuomo Lehtonen, and Andreas Niskanen. ICCMA 2023: 5th international competition on computational models of argumentation. *Artificial Intelligence*, 342:104311, 2025.

[26] Jonas Klein and Matthias Thimm. Probo2: A benchmark framework for argumentation solvers. In Francesca Toni, Sylwia Polberg, Richard Booth, Martin Caminada, and Hiroyuki Kido, editors, *Computational Models of Argument - Proceedings of COMMA 2022*, volume 353 of *Frontiers in Artificial Intelligence and Applications*, pages 363–364. IOS Press, 2022.

[27] Isabel Müßig. *Characteristics of Inconsistency Measures in Argument Graphs in Abstract Argumentation*. Bachelor's Thesis, FernUniversität in Hagen, 2025.

[28] Andreas Niskanen and Matti Järvisalo. μ-toksia: An efficient abstract argumentation reasoner. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020*, pages 800–804, 2020.

[29] Samer Nofal, Katie Atkinson, and Paul E. Dunne. Algorithms for decision problems in argument systems under preferred semantics. *Artificial Intelligence*, 207:23–51, 2014.

[30] Matthias Thimm. On undisputed sets in abstract argumentation. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, pages 6550–6557. AAAI Press, 2023.

[31] Matthias Thimm. Heuristic algorithms for credulous and sceptical reasoning problems in abstract argumentation. *Journal of Logic and Computation*, 35(5), 2025.

[32] Matthias Thimm, Federico Cerutti, and Mauro Vallati. Skeptical reasoning with preferred semantics in abstract argumentation without computing preferred extensions. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021*, pages 2069–2075. ijcai.org, 2021.

[33] Matthias Thimm, Serena Villata, Federico Cerutti, Nir Oren, Hannes Strass, and Mauro Vallati. Summary report of the first international competition on computational models of argumentation. *AI Magazine*, 37(1):102, 2016.

[34] Grigori S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483. Springer, 1983.

[35] Yinlei Yu, Pramod Subramanyan, Nestan Tsiskaridze, and Sharad Malik. All-SAT using minimal blocking clauses. In *2014 27th International Conference on VLSI Design, VLSID 2014*, pages 86–91. IEEE Computer Society, 2014.